

PyCharm 2017.1 Help

Meet PyCharm

Welcome to PyCharm help!

PyCharm is a dedicated Python and Django IDE providing a wide range of essential tools for Python developers, tightly integrated together to create a convenient environment for productive Python development and Web development.

For the newbies

- Before starting with PyCharm, take a look at the [essentials](#), since they are useful for more productive usage of PyCharm.
- If you want to learn about the PyCharm UI, then take a [guided tour](#). The parts [Tool Windows](#) and [PyCharm Editor](#) tell you everything you need to know to get a grip of these UI elements.
- The tutorials for the newbies are in the part [First Steps](#). They help perform the most important basic tasks with PyCharm - create and run an application.
- If you click Help button on a dialog, you will see the [reference page](#) that gives descriptions of controls and fields of each dialog.
- Besides the dialog descriptions, the part [Reference](#) also keeps miscellaneous important information, like [icons reference](#), [index of menu items](#), [version control reference](#), etc.
- Finally, the part [Getting Help](#) tells about using help topics in the online and built-in variants, applying to support service, reporting issues and sharing your feedback.

For advanced users

- For those who want to learn more about PyCharm, the part [General Guidelines](#) provides information about creating projects, configuring their structure, output etc.
- The advanced users will find it useful to read the various sections of the part [Language and Framework Specific Guidelines](#).
- The part [Migration Guides and Tutorials](#) contains the detailed step-by-step tutorials, which will lead you through the specific tasks.

Quick Start Guide

This Quick Start Guide is designed to introduce the key concepts and help you make a quick start with the IDE.

- [Step 0. Before you start](#)
 - [Which languages does PyCharm support?](#)
 - [What platforms can I run PyCharm on?](#)
 - [What do I need to start with PyCharm? What are the system requirements?](#)
 - [Windows](#)
 - [macOS](#)
 - [Linux](#)
 - [Why do I need a project?](#)
- [Step 1. Open/Create a project in PyCharm](#)
 - [Open an existing project](#)
 - [Check out an existing project from Version Control](#)
 - [Create a project from scratch](#)
- [Step 2. Look around](#)
- [Step 3. Customize your environment](#)
 - [Appearance](#)
 - [Editor](#)
 - [Code style](#)
 - [Keymap](#)
- [Step 4. Code with smart assistance](#)
 - [Code completion](#)
 - [Intention actions](#)
- [Step 5. Keep your code neat](#)
- [Step 6. Generate some code](#)
- [Step 7. Find your way through](#)
 - [Basic search](#)
 - [Project navigation](#)
 - [Navigate through the timeline](#)
 - [Find Action](#)
 - [Search Everywhere](#)
- [Step 8. Run and debug](#)
 - [Run configuration](#)
 - [Debug](#)
- [Step 9. Keep your source code under Version Control](#)
 - [VCS](#)
 - [Local history](#)
- [Step 10. That's it! Go ahead and develop with pleasure!](#)

Step 0. Before you start

Which languages does PyCharm support?

With PyCharm you can develop applications in Python. In addition, one can develop Django, Flask, Pyramid and WebToPy applications. Also, it fully supports HTML (including HTML5), CSS, JavaScript, and XML: these languages are bundled in the IDE via plugins and are switched on for you by default. Support for other languages can also be added via plugins (go to Settings | Plugins (or PyCharm | Preferences | Plugins for macOS users) to find out more or set them up during the first IDE launch).

What platforms can I run PyCharm on?

PyCharm is a cross-platform IDE that works on Windows, macOS, and Linux.

What do I need to start with PyCharm? What are the system requirements?

In general to start developing in Python with PyCharm you need the following (depending on your platform):

Windows

System Requirements	Installation
<ul style="list-style-type: none">- Microsoft Windows 10/8/7/Vista/2003/XP (incl.64-bit)- Python 2.4 or higher, Jython, PyPy or IronPython.	<ol style="list-style-type: none">1. Download PyCharm from the Download page.2. Run the <code>pycharm-professional</code> or <code>pycharm-community-*.exe</code> file that starts the Installation Wizard.3. Follow all steps suggested by the wizard. Pay special attention to the corresponding installation options.

macOS

System Installation Requirements

- macOS 10.5 or higher.
- Only 64-bit macOS is supported.
- Python 2.4 or higher, Jython, PyPy or IronPython

1. Download the `pycharm-professional` or `pycharm-community-*.dmg` macOS Disk Image file from the [Download](#) page.
2. Double-click the downloaded `pycharm-professional` or `pycharm-community-*.dmg` macOS Disk Image file to mount it.
3. Copy PyCharm to your Applications folder.

Linux

System Installation Requirements

- OS Linux 64 bit
- KDE, GNOME or Unity DE desktop
- Python 2.4 or higher, Jython, PyPy or IronPython

1. Download the `<pycharm-professional or pycharm-community>-.tar.gz` file from the [Download](#) page.
2. Unpack the `<pycharm-professional or pycharm-community>-.tar.gz` file to a different folder, if your current "Download" folder doesn't support file execution:

```
tar xfz <pycharm-professional or pycharm-community>-.tar.gz <new_archive_folder>
```

Warning! A new instance should not be extracted over the existing one! The target folder must be empty.

The recommended install location according to the filesystem hierarchy standard (FHS) is `/opt`. For example, it's possible to enter the following command:

```
sudo tar xf <pycharm-professional or pycharm-community>-.tar.gz -C /opt/
```

3. Switch to the `bin` directory:

```
cd <new archive folder>/<pycharm-professional or pycharm-community>-*/bin
```

For example,

```
cd opt/<pycharm-professional or pycharm-community>-*/bin
```

4. Run `pycharm.sh` from the `bin` subdirectory.

Why do I need a project?

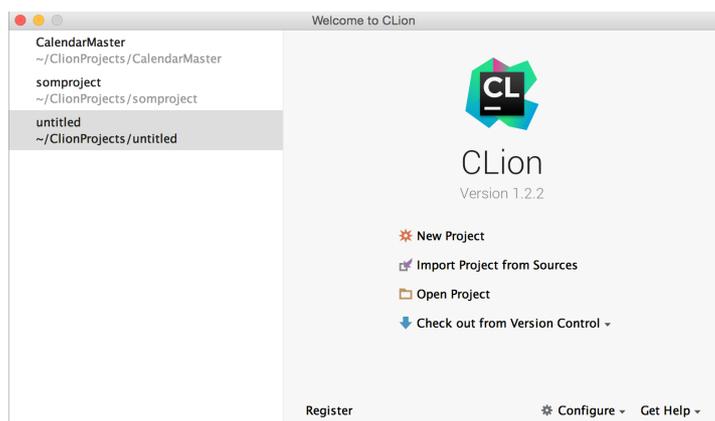
Everything you do in PyCharm is done within the context of a project. It serves as a basis for coding assistance, bulk refactoring, coding style consistency, etc.

Step 1. Open/Create a project in PyCharm

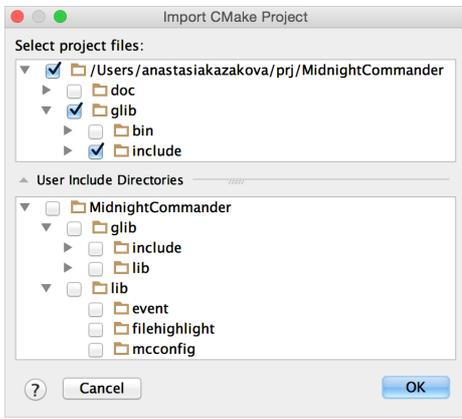
You have three options to start working on a project inside the IDE:

Open an existing project

Begin by [opening](#) one of your existing projects stored on your computer. You can do by clicking Open Project on the Welcome screen (or choosing File | Open in the IDE):



Otherwise select the command Import Project from Sources and specify the location of the sources, then select project files and directories:



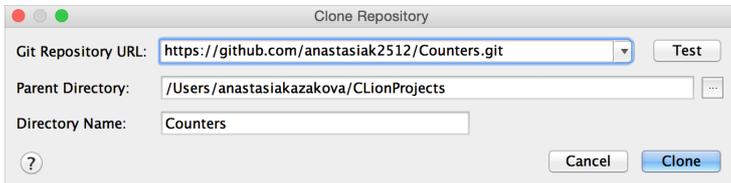
PyCharm will then create a project from your sources for you.

Refer to the section [Importing Project from Existing Source Code](#) for details.

Check out an existing project from Version Control

You can also download sources from a VCS storage or repository. Choose Git (GitHub), CVS, Mercurial, Subversion, Perforce (supported in Professional edition only), and then enter your credentials to access the storage.

Then, enter a path to the sources and clone the repository to the local host:



Refer to the section [Getting Local Working Copy of the Repository](#) or [Setting Up a Local Git Repository](#) for details.

Create a project from scratch

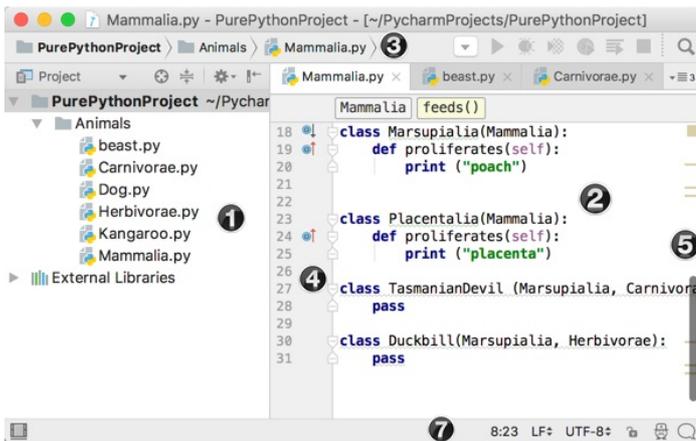
If you prefer to start from scratch, click New Project on the Welcome screen, enter your project's name in the dialog, and a Python project will be created.

Refer to the section [Creating and Managing Projects](#) for details.

Step 2. Look around

When you launch PyCharm for the very first time, or when there are no open projects, you see the [Welcome screen](#). It gives you the main entry points into the IDE: [creating or opening a project](#), checking out a project from version control, [viewing documentation](#), and [configuring the IDE](#).

When a project is opened, you see the main window divided into several logical areas. Let's take a moment to see the key UI elements here:



1. [Project view](#) on the left side displays your project files.
2. [Editor](#) on the right side, where you actually write your code. It has tabs for easy navigation between open files.
3. [Navigation bar](#) above the editor additionally allows you to quickly run and debug your application as well as do the basic [VCS actions](#).
4. Left gutter, the vertical stripe next to the editor, shows the breakpoints you have, and provides a convenient way to [navigate through the code](#) hierarchy like going to definition/declaration. It also shows line numbers and per-line VCS history.
5. Right gutter, on the right side of the editor. PyCharm constantly monitors the quality of your code and always shows the results of its [code analysis](#) in the right gutter: errors, warnings, etc. The indicator in the top right-hand corner shows the overall status of code analysis for the entire file.
6. [Tool windows](#) are specialized windows attached to the bottom and sides of the workspace and provide access to typical tasks such as project management, source code search and navigation, integration with version control systems, etc. Refer to the section [PyCharm Tool Windows](#) for details.
7. [The status bar](#) indicates the status of your project and the entire IDE, and shows various warnings and information messages like file encoding, line separator, inspection profile, etc. Refer to the section [Status Bar](#) for details.

Step 3. Customize your environment

Feel free to [tweak the IDE](#) so it suits your needs perfectly and is as helpful and comfortable as it can be. Go to [File | Settings](#) (PyCharm | Preferences for macOS users) to see the list of available customization options.

Appearance

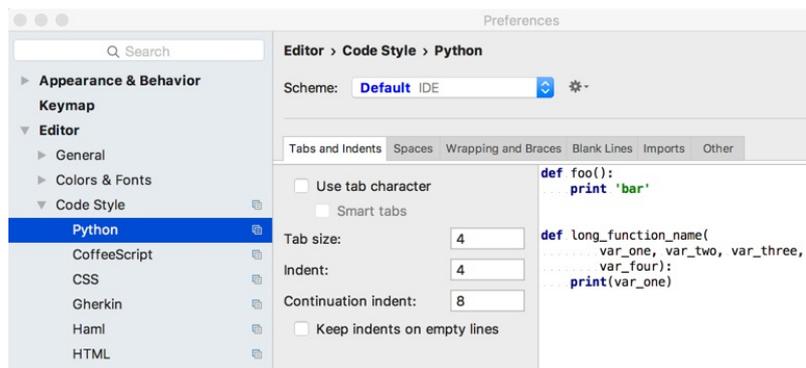
The first thing to fine-tune is the general "look and feel." Go to [File | Settings | Appearance and Behavior | Appearance](#) (PyCharm | Preferences | Appearance and Behavior | Appearance for macOS users) to select the **IDE theme**: the default light theme, or Darcula if you prefer a darker setting.

Editor

The many pages available under [File | Settings | Editor](#) (PyCharm | Preferences | Editor for macOS users) help you adjust every aspect of the editor's behavior. A lot of options are available here, from general settings (like Drag'n'Drop enabling, scrolling configuration, etc.), to color configuration for each available language and use case, to tabs and code folding settings, to code completion behavior and even postfix templates.

Code style

[Code style](#) can be defined for each language under [File | Settings | Editor | Code Style](#) (PyCharm | Preferences | Editor | Code Style for macOS users). You can also create and save your own coding style scheme.



Keymap

PyCharm uses the keyboard-centric approach, meaning that nearly all actions possible in the IDE are mapped to keyboard shortcuts. The [set of keyboard shortcuts](#) you work with is one of your most intimate habits — your fingers "remember" certain combinations of keys, and changing this habit is easier said than done. PyCharm supplies you with a default keymap (choose [Help | Keymap Reference](#) on the main menu) making your coding really productive and convenient. However, you can always [change it](#) by going to [File | Settings | Keymap](#) (PyCharm | Preferences | Keymap for macOS users).

There are also some pre-defined keymaps (like Emacs, Visual Studio, Eclipse, NetBeans etc.), and you can also create your own keymap based on an existing one.

If you feel most productive with [vi/Vim](#), an emulation mode will give you the best of both worlds. Simply enable the [IdeaVim](#) plugin in the IDE and select the vim keymap.

Step 4. Code with smart assistance

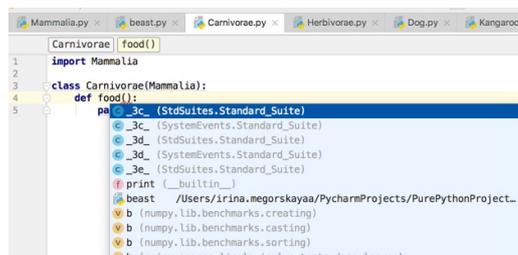
PyCharm takes care of the routine so that you can focus on the important. Use the following coding capabilities to create error-free applications without wasting precious time.

Code completion

[Code completion](#) is a great time-saver, regardless of the type of file you're working with.

[Basic](#) completion works as you type and completes any name instantly.

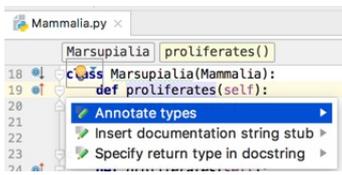
Second completion analyzes the context you're currently working in and offers more accurate suggestions based on that analysis.



Intention actions

PyCharm keeps an eye on what you are currently doing and makes smart suggestions, called [intention actions](#), to save more of your time. Indicated with a lightbulb, intention actions let you apply automatic changes to code that is correct (in contrast to code inspections that provides quick-fixes for code that may be incorrect). Did you forget to add some parameters and field initializers to the constructor? Not a problem with PyCharm. Click the lightbulb (or press

[Alt+Enter](#)) and select one of the suggested options:



The full list of available intention actions can be found in File Settings | Editor | Intentions or PyCharm | Preferences | Editor | Intentions for macOS users.

Step 5. Keep your code neat

PyCharm monitors your code and tries to keep it accurate and clean. It detects potential errors and problems and suggests [quick-fixes](#) for them.

Every time the IDE finds unused code, an endless loop, and many other things that likely require your attention, you'll see a lightbulb. Click it, or press

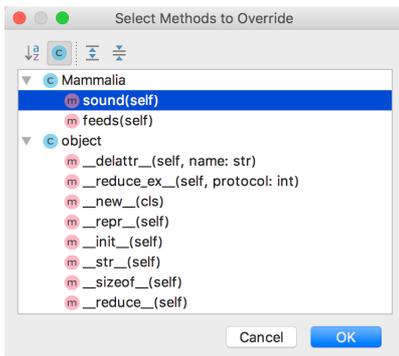
`Alt+Enter`), to apply a fix.

The complete list of available [inspections](#) can be found under Settings | Editor | Inspections (or PyCharm | Preferences | Editor | Inspections for macOS users).

Disable some of them, or enable others, plus adjust the severity of each inspection. You decide whether it should be considered an error or just a warning.

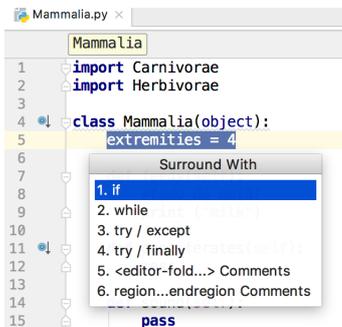
Step 6. Generate some code

Writing code can be a lot easier and quicker when you use the [code generation options](#) available in PyCharm. The Code | Generate menu (`Alt+Insert`) will help you with creating symbols from usage, as well as suggest overriding/implementing some functions:

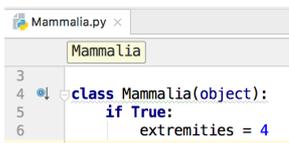


Use live templates (choose Code | Insert Live Template or press `Ctrl+J`) to produce the entire code constructs. You can explore the available ready-to-use live templates in the Settings/Preferences dialog ([Settings | Editor | Live templates](#) or PyCharm | Preferences | Editor | Live Templates if you are a macOS user).

If you see that you are lacking something especially important for your development, extend this set of templates with your own. Also, consider quickly [surrounding](#) your code with complete constructs (choose Code | Surround With or press `Ctrl+Alt+T`). For example, select an `if` statement:



and you will get:



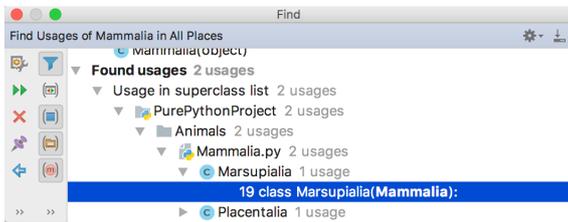
Step 7. Find your way through

When your project is big, or when you have to work with someone else's code, it's vital to be able to quickly find what you are looking for and dig into the code.

This is why PyCharm comes with a set of [navigation](#) and [search](#) features that help you find your way through any code no matter how tangled it is.

Basic search

To find where a particular symbol is used, PyCharm suggests full-scale search via [Find Usages](#) (`Alt+F7`):

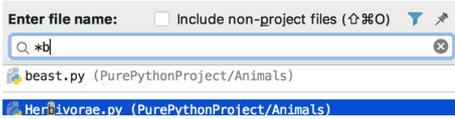


You also have the option to search only in the current file (`Ctrl+F`), or within a directory, any arbitrary scope, or the entire project (`Ctrl+Shift+F`).

Project navigation

You can tell a lot just looking at your File Structure, with its imports or call hierarchies, and possibly use it to navigate to:

- **Class by its name** (`Ctrl+N`).
- **File by its name** (`Ctrl+Shift+N`):



- **Symbol by its name** (`Ctrl+Shift+Alt+N`).
- **Declaration** (`Ctrl+B`).
- **Base class/base function** (`Ctrl+U`).

The icons in the left-hand gutter can also help you with navigation:



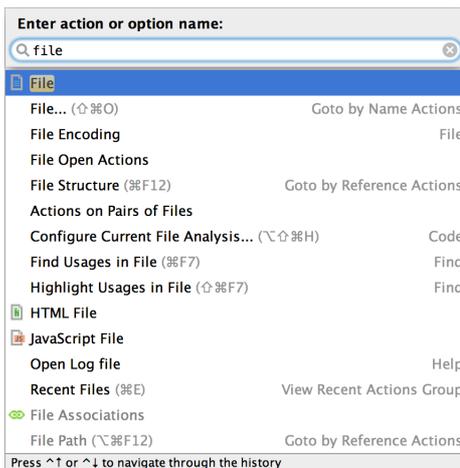
Navigate through the timeline

Remembering all your activity in the project, PyCharm can easily navigate you to the **Recent Files** (`Ctrl+E`) or **Recently Changed Files** (`Shift+Alt+C`).

To go through the history of changes, try using Back/Forward navigation (`Ctrl+Alt+Left` / `Ctrl+Alt+Right`) and/or go to last edit location (`Ctrl+Shift+Backspace`).

Find Action

Take advantage of many smart actions possible with PyCharm. For example, use the **Find Action** search (`Ctrl+Shift+A`): just type a part of the action name, and the IDE will show you the list of all available options. Then, select the action you need:



Search Everywhere

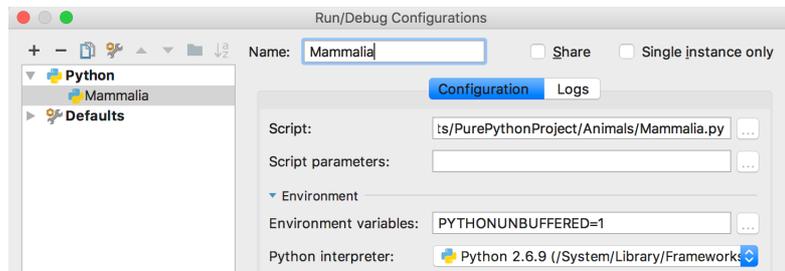
If you have a general idea of what you're looking for, you can always locate the corresponding element using one of the existing navigation features. But what if you really want to look for something in every nook and cranny? The answer is to use **Search Everywhere!** To try it, click the magnifying glass in the upper right-hand corner of the window, or invoke it with Double Shift (press Shift twice).

Step 8. Run and debug

Run configuration

Now when you've played with the code and discovered what you can do with it, it's time to run your app. In PyCharm you do this via the **Run/Debug Configurations**. Open Run | Edit Configurations to see all the available options. For example, if you want to run some script before/after the build phase, you can

do this easily by creating an external tool:



To run a configuration, press `Shift+F10`.

Debug

Does your application stumble on a run-time error? To find out what's causing it, you will have to do some debugging. PyCharm supports the debugger on all platforms.

[Debugging](#) starts with placing breakpoints at which program execution will be suspended, so you can explore program data. Just click the left gutter of the line where you want the breakpoint to appear. To start debugging your application, press `Shift+F9`. Then go through the program execution step by step (see the available options in the Run menu or in the [Debug tool window](#)), evaluate any arbitrary expression, add watches and manually set values for the variables.

Step 9. Keep your source code under Version Control

VCS

If you are keeping your source code under version control, you will be glad to know that PyCharm integrates with many popular [version control systems](#): Git (or GitHub), Mercurial, Perforce (supported in Professional edition only), Subversion and CVS. To specify credentials and any settings specific to a particular VCS, go to [Settings | Current Project | Version Control](#) (or PyCharm | Preferences | Current Project | Version Control if you are macOS user).

The VCS menu gives you a clue about what commands are available. For example, you can see the changes you've made, commit them, create changelists and much more from the Local Changes view: VCS | Show Changes (or just press `Alt+9`). Also find some VCS basic commands in the Navigation bar above the editor:



Refer to the section [Version Control with PyCharm](#) for details.

Local history

In addition to traditional version control, you can use the [local history](#). With Local History, PyCharm automatically tracks changes you make to the source code, the results of refactoring, etc. Local history is always enabled. To view it for a file or a folder, bring up Local History by selecting VCS | Local History | Show History. Here you can review the changes, revert them or create a patch.

Refer to the section [Using Local History](#) for details.

Step 10. That's it! Go ahead and develop with pleasure!

We hope this brief overview of essential PyCharm features will give you a quick start. There are many important features that make a developer's life easier and more fun, and their source code neater and cleaner. Take these first few steps now, and then dig deeper when you feel the time is right. Enjoy PyCharm!

With any questions please visit our [Discussion Forum](#), [twitter](#) and [blog](#), where you can find news, updates, and useful tips and tricks. Also, don't hesitate to report any problems to our [support team](#) or the PyCharm [issue tracker](#).

Keyboard Shortcuts You Cannot Miss

PyCharm, as a keyboard-centric IDE, suggests keyboard shortcuts for most of its actions. In this topic, you can find a short list of the most indispensable of them, to make your first steps with PyCharm easy.

Tip PyCharm also provides the default keymap reference in the [pdf](#) format. To view the keymap reference, choose Help | Keymap Reference from the main menu.

See the detailed list of default keyboard shortcuts in the [Keyboard Shortcuts Reference](#) and learn how to customize your preferred keymap in the [Configuring Keyboard Shortcuts](#) section.

Action	Shortcut
Find action by name	Ctrl+Shift+A
Show the list of available intention actions .	Alt+Enter
Switch between views (Project , Structure , etc.).	Alt+F1
Switch between the tool windows and files opened in the editor.	Ctrl+Tab
Show the Navigation bar .	Alt+Home
Insert a live template .	Ctrl+J
Surround with a live template .	Ctrl+Alt+J
Edit an item from the Project or another tree view.	F4
Comment or uncomment a line or fragment of code with the line or block comment .	Ctrl+Slash Ctrl+Shift+Slash
Find class or file by name .	Ctrl+N Ctrl+Shift+N
Duplicate the current line or selection.	Ctrl+D
Incremental expression selection.	Ctrl+W and Ctrl+Shift+W
Find/replace text string in the current file .	Ctrl+F Ctrl+R
Find/replace text in the project or in the specified directory	Ctrl+Shift+F Ctrl+Shift+R
Search everywhere.	Double-press Shift
Quick view the usages of the selected symbol.	Ctrl+Shift+F7
Expand or collapse a code fragment in the editor.	Ctrl+NumPad Plus Ctrl+NumPad -
Invoke code completion.	Ctrl+Space
Show the list of available refactorings (Refactor This).	Ctrl+Shift+Alt+T

The complete keymap reference is available from the main menu (Help | Keymap Reference).

First Steps

This part contains the basic tutorials:

- [Creating and Running Your First Python Project](#)
- [Debugging Your First Python Application](#)
- [Testing Your First Python Application](#)
- [Creating and Running Your First Django Project](#)

Creating and Running Your First Python Project

In this section:

- [Before you start](#)
- [Creating a simple Python project in PyCharm](#)
 - [Sine note](#)
- [Creating a Python file](#)
- [Editing source code](#)
- [Running application](#)
- [Run/debug configuration](#)

Before you start

Make sure that the following prerequisites are met:

- You are working with [PyCharm](#) CE or Professional.
- You have installed Python itself. If you're using macOS or Linux, your computer already has Python installed. You can get Python from python.org.

Creating a simple Python project in PyCharm

To get started with PyCharm, let's write a Python script.

Do you remember the [quadratic formula from math class](#)? This formula is also known as the A, B, C formula, it's used for solving a simple quadratic equation:

$ax^2 + bx + c = 0$. As manually solving quadratic formulas gets boring quickly, let's replace it with a script.

Note that PyCharm suggests several project templates for the development of the various types of applications (Django, Google AppEngine, etc.). When PyCharm creates a new project from a project template, it produces the corresponding directory structure and specific files.

Let's start our project: if you're on the [Welcome screen](#), click Create New Project. If you've already got a project open, choose File | New Project.

PyCharm suggests several project templates for the creation of the various types of applications (Django, Google AppEngine, etc.). When PyCharm creates a new project from a project template, it produces the corresponding directory structure and specific files, and any needed run configurations or settings.

In this tutorial we'll create a simple Python script, so we'll choose Pure Python. This template will create an empty project for us.

Choose the project location. To do that, click the browse button  next to the Location field, and specify the directory for your project.

Sine note

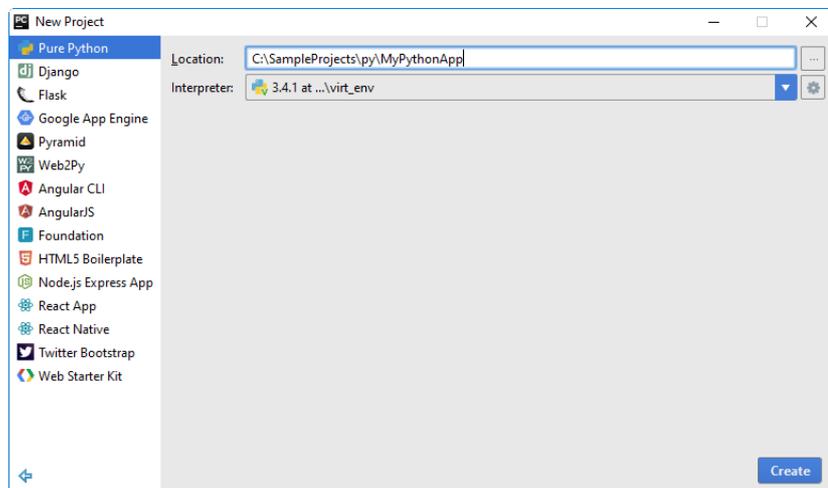
Choosing which interpreter to use for a project is an important decision. Python is a script language, which means that your code is converted to machine code by a Python interpreter.

You can have multiple versions of Python installed on your computer, and you need to choose the one you'd like for this project. Refer to the section [Configuring Python SDK](#) for details. Note that you can always change your mind later and specify another interpreter for your project.

When you're using external libraries (from PyPI or elsewhere) you need to manage the versions of these as well. The Pythonic solution for this are virtualenvs (sometimes abbreviated to venv). Virtualenvs help you keep the dependencies for your different projects separate. Although in this project we're not using any dependencies, it's considered best practice to create a virtualenv for every new project you make, in case you'd like to add dependencies in the future.

Python best practice is to create a virtualenv for each project. To do that, click , choose Create VirtualEnv, and then specify the virtual environment's name and location, and the base interpreter. For more information, see [Creating Virtual Environment](#).

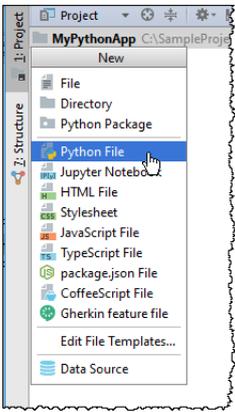
Then click Create.



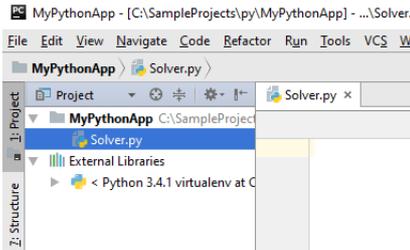
If you've already got a project open, after clicking Create PyCharm will ask you whether to open a new project in the current window or in a new one. Choose Open in current window - this will close the current project, but you'll be able to reopen it later. Refer to the page [Opening Multiple Projects](#) for details.

Creating a Python file

Select project root in the Project tool window, and press [Alt+Insert](#).



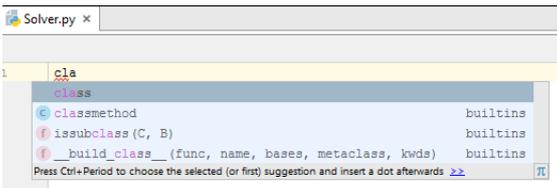
Choose the option Python file from the pop-up window, and then type the new file name `Solver`. PyCharm creates a new Python file and opens it for editing.



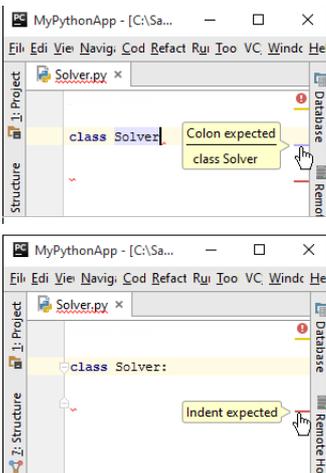
Editing source code

Let us first have a look at the Python file we've just generated.

Immediately as you start typing, you understand that PyCharm, like a pair-programmer, looks over your shoulder and suggests how to complete your line. For example, you want to create a Python class. As you just start typing the keyword, a suggestion list appears:



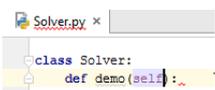
Choose the keyword `class` and type the class name (`Solver`). PyCharm immediately informs you about the missing colon, then expected indentation:



Note the error stripes in the right gutter. Hover your mouse pointer over an error stripe, and PyCharm shows a balloon with the detailed explanation. Since PyCharm analyses your code on-the-fly, the results are immediately shown in the inspection indicator on top of the right gutter.

This inspection indication works like a traffic light: when it is green, everything is OK, and you can go on with your code; a yellow light means some minor problems that however will not affect compilation; but when the light is red, it means that you have some serious errors.

Let's continue creating the function `demo`: when you just type the opening brace, PyCharm creates the entire code construct (mandatory parameter `self`, closing brace and colon), and provides proper indentation:



Note as you type, that unused symbols are greyed out:

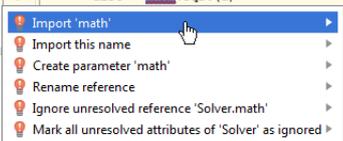
```
Solver.py x
class Solver:
    def demo(self):
        a = int(input("a "))
        b = int(input("b "))
        c = int(input("c "))
```

As soon as you calculate a discriminant, they are rendered as usual. Next, pay attention to the unresolved reference `math`. PyCharm underlines it with the red curly line, and shows the red bulb.

Let's make a brief excursion into PyCharm's notion of [intention actions and quick fixes](#). When you write your code, it is sometimes advisable to modify code constructs - in this case PyCharm shows a yellow light bulb. However, if PyCharm encounters an error, it shows the red light bulb.

In either case, to see what does PyCharm suggest you to do, press `Alt+Enter` - this will display the suggestion list, which in our case contains several possible solutions:

```
Solver.py x
class Solver:
    def demo(self):
        a = int(input("a "))
        b = int(input("b "))
        c = int(input("c "))
        d = b ** 2 - 4 * a * c
        disc = math.sqrt(d)
```



- Import 'math'
- Import this name
- Create parameter 'math'
- Rename reference
- Ignore unresolved reference 'Solver.math'
- Mark all unresolved attributes of 'Solver' as ignored

Let's choose importing the `math` library. Import statement is added to the `Solver.py` file. Next, calculate roots of the quadratic equation, and print them out, and finally, let's call the function `demo` of the class `Solver`:

```
import math

class Solver:
    def demo(self):

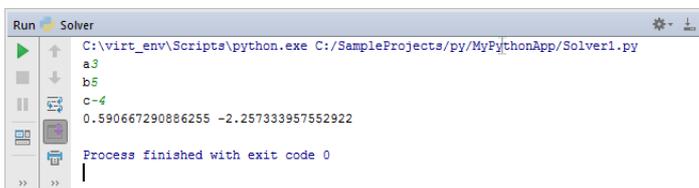
        a = int(input("a "))
        b = int(input("b "))
        c = int(input("c "))
        d = b ** 2 - 4 * a * c
        disc = math.sqrt(d)
        root1 = (-b + disc) / (2 * a)
        root2 = (-b - disc) / (2 * a)
        print(root1, root2)

Solver().demo()
```

Running application

Next, right-click the editor, and on the context menu choose to run the script (`Ctrl+Shift+F10`). A console appears in the [Run tool window](#).

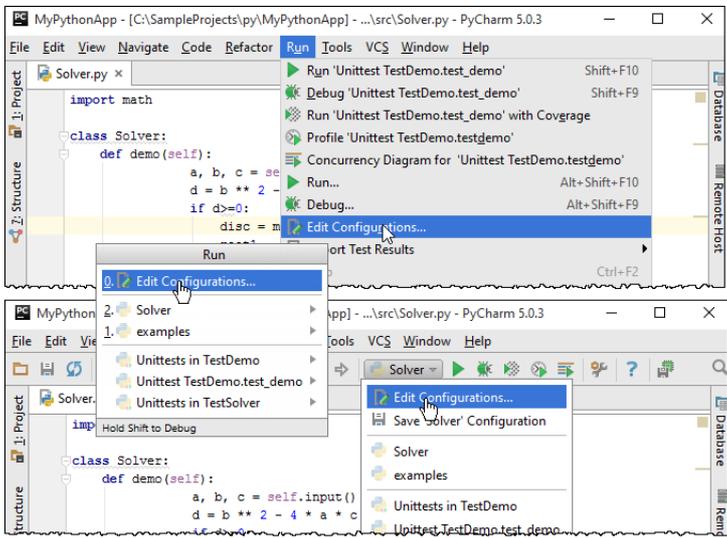
Let's try out our code by calculating the roots of $3x^2 + 5x - 4$. So enter `a = 3`, `b = 5`, and `c = -4`, and you should see a result:



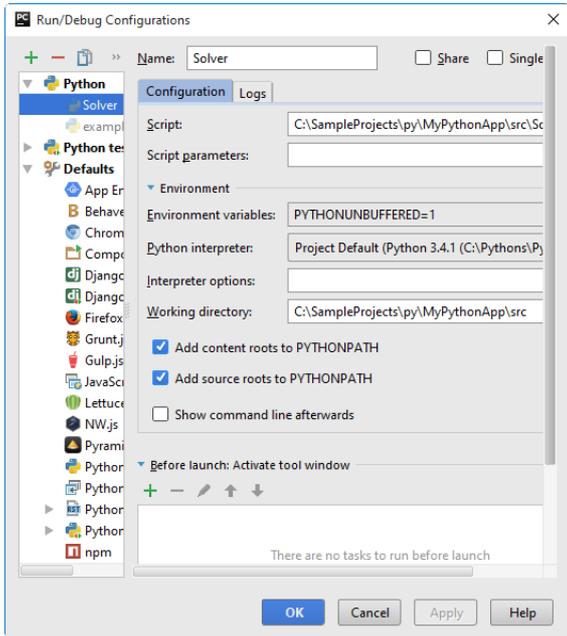
```
Run Solver
C:\virt_env\Scripts\python.exe C:/SampleProjects/py/MyPythonApp/Solver1.py
a3
b5
c-4
0.590667290886255 -2.257333957552922
Process finished with exit code 0
```

Run/debug configuration

When we ran the script just now, PyCharm created a temporary [run/debug configuration](#) for us. Let's first save this configuration: go to the run configuration dropdown on the top-right of the editor, and choose Save configuration.



Afterwards, choose Edit Configurations to have a look at what is happening here.



Note that there are different configuration types for unit testing, and the various frameworks (like Django in PyCharm Professional).

If you'd like to change how your program is executed by PyCharm, this is where you can configure various settings like: command-line parameters, work directory, and more. See [Run/Debug configurations](#) for more details.

If you'd like to start the script using this Run configuration, use the button  next to the dropdown.

Congratulations on completing your first script in PyCharm! Learn how to [debug](#) your programs in PyCharm.

Debugging Your First Python Application

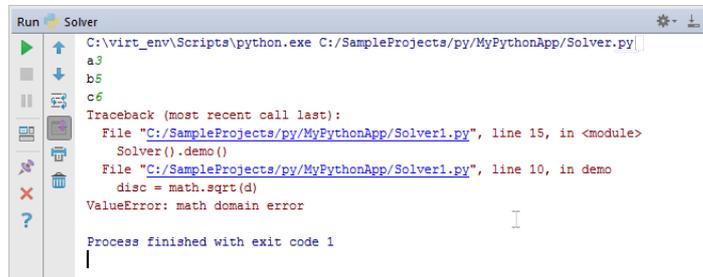
In this section:

- [Finding out the origin of the problem](#)
 - [Surrounding code](#)
- [Debugging](#)

Finding out the origin of the problem

Remember, in the [previous tutorial](#) you've solved the quadratic equation? Let's play a little more with it, and calculate the roots of $3x^2 + 5x + 6$.

Oops... PyCharm reports a run-time error:

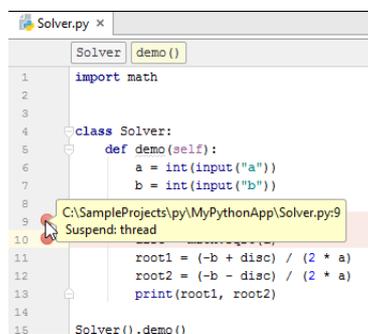


```
Run Solver
C:\virt_env\Scripts\python.exe C:/SampleProjects/py/MyPythonApp/Solver.py
a 3
b 5
c 6
Traceback (most recent call last):
  File "C:/SampleProjects/py/MyPythonApp/Solver1.py", line 15, in <module>
    Solver().demo()
  File "C:/SampleProjects/py/MyPythonApp/Solver1.py", line 10, in demo
    disc = math.sqrt(d)
ValueError: math domain error

Process finished with exit code 1
```

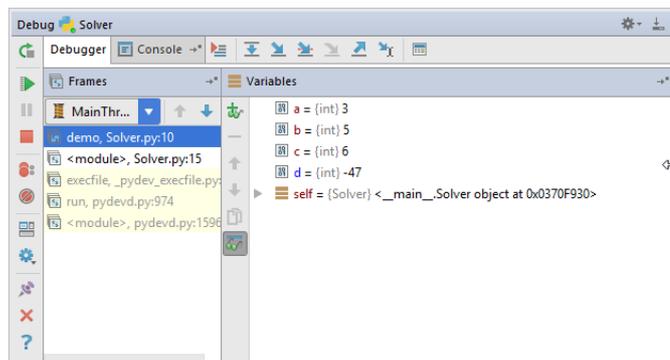
We're getting a 'ValueError: math domain error'. This means that we're doing something which is mathematically impossible.

Let's dig a little deeper into our code to find out what's going wrong. We can use the PyCharm debugger to see exactly what's happening in our code. To start debugging, you have to set the [breakpoints](#) first. To create breakpoints, just click the left gutter:



```
Solver.py x
Solver demo()
1 import math
2
3
4 class Solver:
5     def demo(self):
6         a = int(input("a"))
7         b = int(input("b"))
8
9         C:\SampleProjects\py\MyPythonApp\Solver.py:9
10        Suspend: thread
11        root1 = (-b + disc) / (2 * a)
12        root2 = (-b - disc) / (2 * a)
13        print(root1, root2)
14
15 Solver().demo()
```

Next, right-click the editor background, and choose [Debug 'Solver'](#) on the context menu. PyCharm starts the debugging session and shows the [Debug tool window](#).



```
Debug Solver
Debugger Console
Frames Variables
MainThr...
demo, Solver.py:10
<module>, Solver.py:15
execfile, _pydev_execfile.py:
run, pydevd.py:974
<module>, pydevd.py:1596
a = (int) 3
b = (int) 5
c = (int) 6
d = (int) -47
self = {Solver} <__main__.Solver object at 0x0370F930>
```

The [Variables](#) tab shows the entered values of the variables `a`, `b`, `c` and the calculated value of the variable `d`. Now remember, we got an exception in the line where we calculated `math.sqrt(d)`. We can see that the value of the discriminant `d` is `-47`, calculating the square root of `-47` would result in an imaginary number. So we've found our problem.

Surrounding code

To prevent our users from getting this exception again, let's add an `if` statement to check for a negative discriminant. If the discriminant is negative, that means that the equation the user has entered has no roots (it doesn't cross the X-axis). So let's just tell the user that in case we get a negative discriminant. To do that, select the discriminant calculation statements, and then press `Ctrl+Alt+T` (Code | Surround with):

```

1 import math
2
3
4 class Solver:
5     def demo(self):
6         a = int(input("a"))
7         b = int(input("b"))
8         c = int(input("c"))
9         d = b ** 2 - 4 * a * c
10        disc = math.sqrt(d)
11        root1 = (-b + disc) / (2 * a)
12        root2 = (-b - disc) / (2 * a)
13        print(root1, root2)
14
15 Solver().demo()
16

```

PyCharm creates a stub `if` construct, leaving you with the task of filling it with the proper contents. Finally, it would be nice to have the whole calculation repeated more than once, so let's use the `Surround with` action again: select the entire body of the function `demo` and surround it with `while`. You'll end up with this code:

```

import math

class Solver:
    def demo(self):
        while True:
            a = int(input("a "))
            b = int(input("b "))
            c = int(input("c "))
            d = b ** 2 - 4 * a * c
            if d >= 0:
                disc = math.sqrt(d)
                root1 = (-b + disc) / (2 * a)
                root2 = (-b - disc) / (2 * a)
                print(root1, root2)
            else:
                print('This equation has no roots')

Solver().demo()

```

Next, let's debug this script, and then run it again.

Debugging

As it was said previously, the debugging session begins with putting breakpoints, and displays the debugging output in the Debug tool window.

The Debug tool window shows dedicated panes for `frames`, `variables`, and `watches`, and the `console`, where all the input and output information is displayed. If you want the console to be always visible, just drag it to the desired place.

Use the `stepping toolbar buttons` to step through your application:



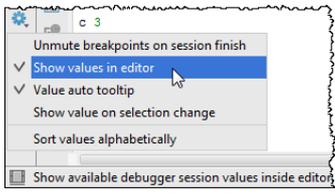
As you step through the application, each reached breakpoint becomes blue:

```

Solver.py x
Solver demo() while True
1 import math
2
3
4 class Solver:
5     def demo(self): self: <_main_.Solver object at 0x03A8F250>
6         while True:
7             a = int(input("a")) a: 1
8             b = int(input("b")) b: 2
9             c = int(input("c")) c: 3
10            d = b ** 2 - 4 * a * c
11            if d > 0:
12                disc = math.sqrt(d)
13                root1 = (-b + disc) / (2 * a)
14                root2 = (-b - disc) / (2 * a)
15                print(root1, root2)
16            else:
17                print_ ("This equation has no roots")
18
19
20 Solver().demo()

```

Pay attention to the values that appear next to the variables in the editor: this is the `inline debugging`. This feature is enabled in the Debug tool window:



Testing Your First Python Application

In this section:

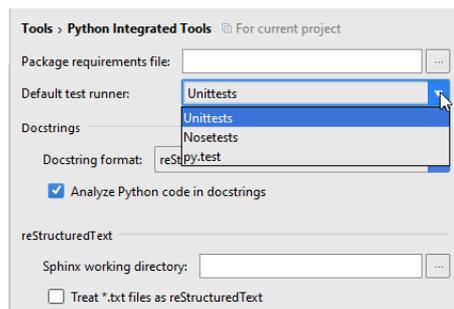
- [Introduction](#)
- [Before you start](#)
- [Creating test](#)
- [Transforming the source code](#)
 - [Transforming the source code of the class Solver](#)
 - [Transforming the source code of the test class](#)

Introduction

Remember, in the first tutorial you've [created your first Python application](#), and in the second tutorial you've [debugged](#) it. Now it's time to do some testing.

Before you start

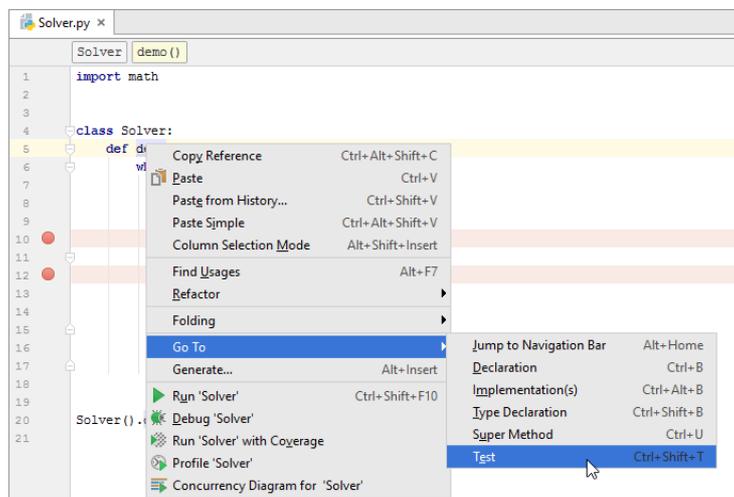
Open Settings/Preferences dialog (`Ctrl+Alt+S`), under the Tools node click Python Integrated Tools and select the default test runner - let it be `unittest` (refer to [this page](#) for details).



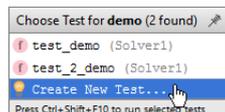
Note also that we are using `unittest` to test our application, but if you prefer to use another framework, PyCharm also supports `nosetest` and `py.test`.

Creating test

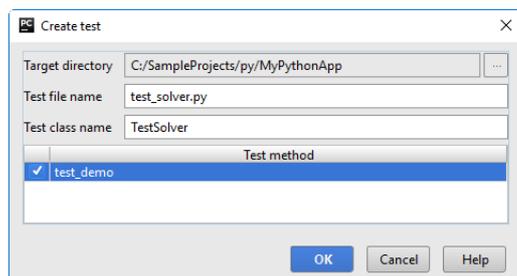
Open the Python file `Solver.py` for editing (`F4`). In the class `Solver` right-click the editor background, point to Go To, and then choose Test (or just press `Ctrl+Shift+T`):



The pop-up appears, suggesting you to create a new test:



OK, let's do it:



The new Python test class is created:

```
Solver.py x test_solver.py x
TestSolver | test_demo()
1 from unittest import TestCase
2
3
4 class TestSolver(TestCase):
5     def test_demo(self):
6         self.fail()
7
```

You see that there is the `import` statement, the test class, as required by the [documentation](#), and the test method for the method `demo`, that just fails.

Run this test (Run 'Solver_unittest' on the context menu) and see it failing.

However, this failing behavior is by default - let's now provide some meaningful contents, but before that some transformation is required.

Transforming the source code

Transforming the source code of the class Solver

First, let's transform our code a little. Let's add a check for the zero value of `d`, and place all input statements out of the class `Solver`:

```
import math

class Solver:

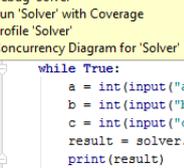
    def demo(self, a, b, c):
        d = b ** 2 - 4 * a * c
        if d > 0:
            disc = math.sqrt(d)
            root1 = (-b + disc) / (2 * a)
            root2 = (-b - disc) / (2 * a)
            return root1, root2
        elif d == 0:
            return -b / (2 * a)
        else:
            return "This equation has no roots"

if __name__ == '__main__':
    solver = Solver()

    while True:
        a = int(input("a: "))
        b = int(input("b: "))
        c = int(input("c: "))
        result = solver.demo(a, b, c)
        print(result)
```

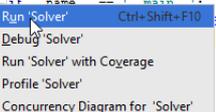
Now you see the Run icon  in the left gutter next to the block `if __name__ == '__main__':`. If you hover your mouse pointer over this icon, you will see a pop-up window describing the available commands:

```
Solver.py x
Solver
1 # ...
21 import math
22
23
24 class Solver: ...
37
38
39
40
41
42 while True:
43     a = int(input("a: "))
44     b = int(input("b: "))
45     c = int(input("c: "))
46     result = solver.demo(a, b, c)
47     print(result)
48
```

Clicking this icon reveals the pop-up menu:

```
Solver.py x
if __name__ == ...
23
24
25
26
```



Choose Run 'Solver' and see your script running.

Transforming the source code of the test class

We'll add few methods that check the equation roots, and also the method [setUp](#):

```
import unittest

from Solver import Solver

class SolverTest(unittest.TestCase):

    def setUp(self):
        self.solver = Solver()

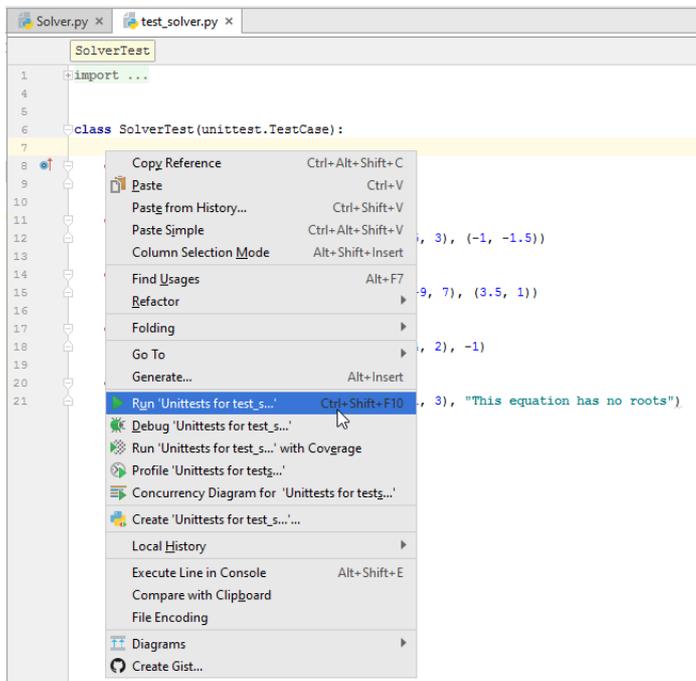
    def test_two_roots_1(self):
        self.assertEqual(self.solver.demo(2, 5, 3), (-1, -1.5))

    def test_two_roots_2(self):
        self.assertEqual(self.solver.demo(2, -9, 7), (3.5, 1))

    def test_one_root(self):
        self.assertEqual(self.solver.demo(2, 4, 2), -1)

    def test_no_roots(self):
        self.assertEqual(self.solver.demo(2, 1, 3), "This equation has no roots")
```

Now run the test by right-clicking the editor background above the declaration of the class `SolverTest`. This time all the tests pass successfully:



Creating and Running Your First Django Project

This feature is supported in the Professional edition only.

In this section:

- [Before you start](#)
- [Creating a new project](#)
- [Exploring project structure](#)
 - [Project view of the Project tool window](#)
 - [Project Files view of the Project tool window](#)
 - [What do we see in the Project view?](#)
- [Configuring the database](#)
- [Launching Django server](#)
- [Creating models](#)
- [Creating database](#)
- [Performing administrative functions](#)
 - [Preparing run/debug configuration](#)
 - [Launching the admin site](#)
 - [Editing admin.py](#)
- [Writing views](#)
- [Creating Django templates](#)
- [Using a stylesheet](#)
- [Here we are!](#)
- [Test it...](#)
- [Summary](#)

Before you start

Make sure that the following prerequisites are met:

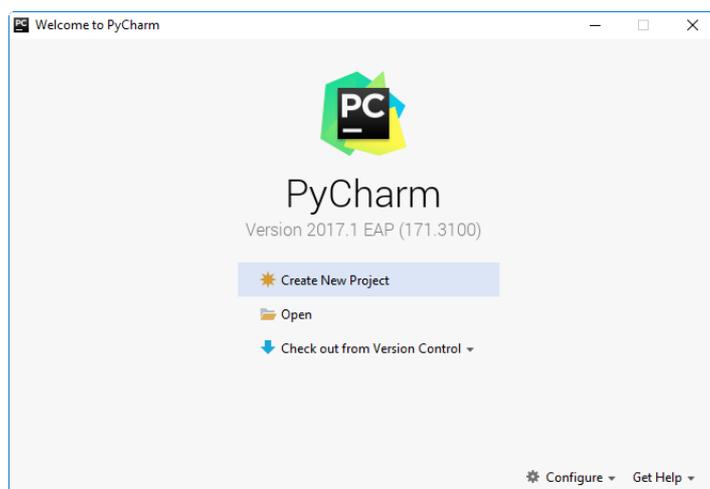
- You are working with PyCharm version 2016.1 or higher. If you still do not have PyCharm, download it from [this page](#). To install PyCharm, follow the instructions, depending on your platform.
- You have at least one Python interpreter properly installed on your computer. You can download an interpreter from [this page](#).
- You have Django package installed. To learn how to install packages using the PyCharm UI, read the section [Installing, Uninstalling and Upgrading Packages](#). You can also install Django as described in the page [How to install Django](#).

This tutorial has been created with the following assumptions:

- Python 3.4.1.
- Django 1.10.0 or higher.
- The example used in this tutorial is similar to the one used in [Django documentation](#). Sample project can be downloaded from [here](#).

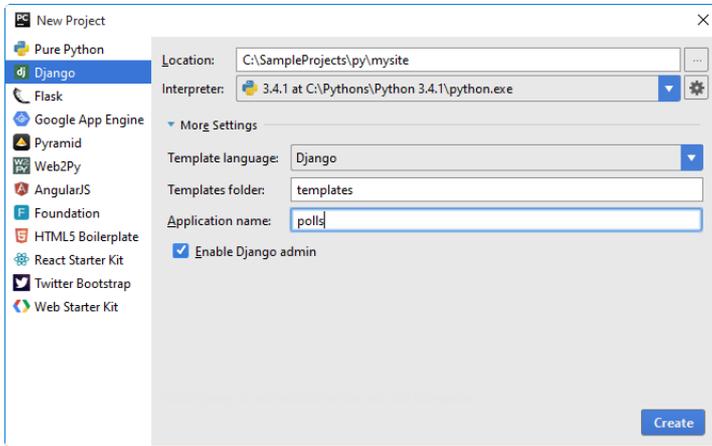
Creating a new project

Actually, all new projects are created same way: by clicking the Create New Project button in the Quick Start area of the [Welcome screen](#):



If you have an already opened project, create a new one by choosing File | New Project... on the main menu.

Then, select the desired project type (here it is Django). Specify the project name and location, the Python interpreter to be used for this project (remember, having at least one Python interpreter is one of the prerequisites of this tutorial!), and the application name (here it is polls).



Click Create - the Django project is ready.

Exploring project structure

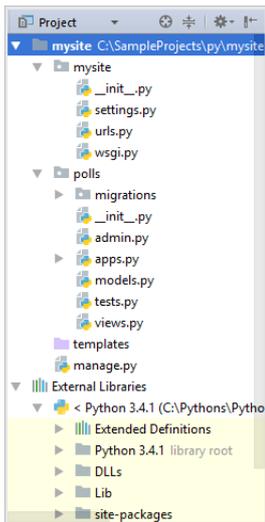
As mentioned above, basically, the stub project is ready. It contains framework-specific files and directories. Same happens when you create a project of any supported type, be it Pyramid, or Google App Engine.

Let's see how the structure of the new project is visible in the [Project tool window](#).

Project view of the Project tool window

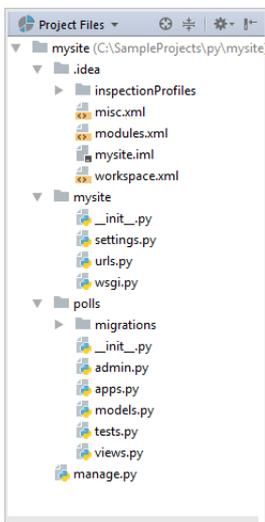
This view is displayed by default. It shows the Django-specific project structure: `polls` and `mysite` directories; also, you see the `manage.py` and `settings.py` files.

Note that you cannot see the `.idea` directory in this view:



Project Files view of the Project tool window

If for some reasons you would like to see contents of the `.idea` directory, choose the view Project Files: as you see, this view shows same directories and files, plus `.idea` directory, since is located under the project root:



Let's return to the Project view.

What do we see in the Project view?

- `mysite` directory is a container for your project. In the Project view it is denoted with bold font.
- `manage.py` : This is a command-line utility that lets you interact with your Django project. Refer to the [Django documentation](#) for details.
- The nested directory `mysite` is the actual Python package for your project.
- `mysite/___init__.py` : This empty file tells Python that this directory should be considered a Python package.
- `mysite/settings.py` : This file contains [configuration for your Django project](#).
- `mysite/urls.py` : This file contains the [URL declarations for your Django project](#).
- `mysite/wsgi.py` : This file defines an entry-point for WSGI-compatible web servers to serve your project. See [How to deploy with WSGI](#) for more details.
- The nested directory `polls` contains all the files required for developing a Django application (at this moment, these files are empty):
 - Again, `polls/___init__.py` tells Python that this directory should be considered a Python package.
 - `polls/models.py` : In this file, we'll [create models](#) for our application.
 - `polls/views.py` : In this file, we'll [create views](#).
- `templates` directory is by now empty. It should contain the template files.
- The nested directory `migrations` contains by now only the package file `___init__.py`, but will be used in the future to propagate the changes you make to your models (adding a field, deleting a model, etc.) into your database schema. Read the migrations description [here](#).

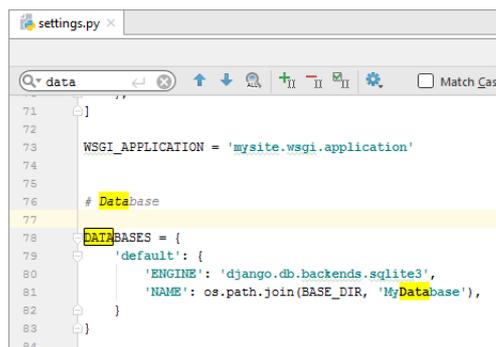
Note that you can create as many Django applications as needed. To add an application to a project, run the `startapp` task of the `manage.py` utility (Tools | Run manage.py task, then type `startapp` in the console).

Configuring the database

Now, when the project stub is ready, let's do some fine tuning. Open for editing `settings.py`. To do it, select the file in the Project tool window, and press `F4`. The file opens in its own tab in the editor.

Specify which database you are going to use in your application. For this purpose, find the `DATABASES` variable: click `Ctrl+F`, and in the search field start typing the string you are looking for. Then, in the `'ENGINE'` line, add the name of your database management system after dot (you can use any one specified after comment, but for the beginning we'll start with `sqlite3`.)

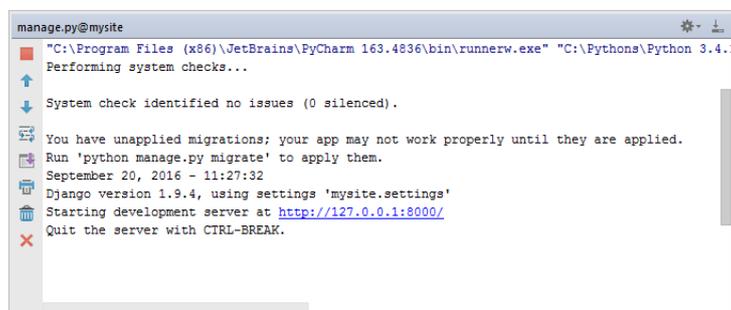
In the `'NAME'` line, enter the name of the desired database, even though it doesn't yet exist.



```
71 }
72
73 WSGI_APPLICATION = 'mysite.wsgi.application'
74
75 # Database
76
77
78 DATABASES = {
79     'default': {
80         'ENGINE': 'django.db.backends.sqlite3',
81         'NAME': os.path.join(BASE_DIR, 'MyDatabase'),
82     }
83 }
84
```

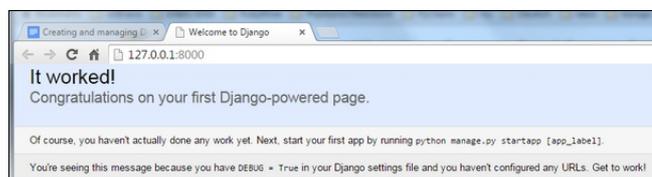
Launching Django server

Since we've prudently chosen `sqlite3`, we don't need to define the other values (user credentials, port and host). Let's now check whether our settings are correct. This can be done most easily: just launch the `runserver` task of the `manage.py` utility: press `Ctrl+Alt+R`, and enter task name in the `manage.py` console:



```
manage.py@mysite
"C:\Program Files (x86)\JetBrains\PyCharm 163.4836\bin\runserver.exe" "C:\Python3\Python 3.4.
Performing system checks...
System check identified no issues (0 silenced).
You have unapplied migrations; your app may not work properly until they are applied.
Run 'python manage.py migrate' to apply them.
September 20, 2016 - 11:27:32
Django version 1.9.4, using settings 'mysite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Follow the suggested link and see the following page:



Creating models

Next, open for editing the file `models.py`, and note that import statement is already there. Then type the following code:

```

from django.db import models

# the following lines added:
import datetime
from django.utils import timezone

class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')

    def __str__(self):
        return self.question_text

    def was_published_recently(self):
        now = timezone.now()
        return now - datetime.timedelta(days=1) <= self.pub_date <= now

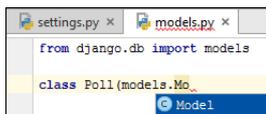
was_published_recently.admin_order_field = 'pub_date'
was_published_recently.boolean = True
was_published_recently.short_description = 'Published recently?'

class Choice(models.Model):
    question = models.ForeignKey(Question)
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField(default=0)

    def __str__(self):
        return self.choice_text

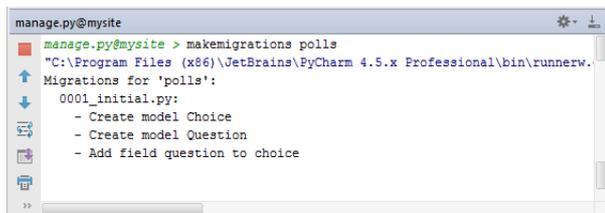
```

Actually, you can just copy-paste, but typing is advisable - it helps you see the powerful PyCharm's code completion in action:

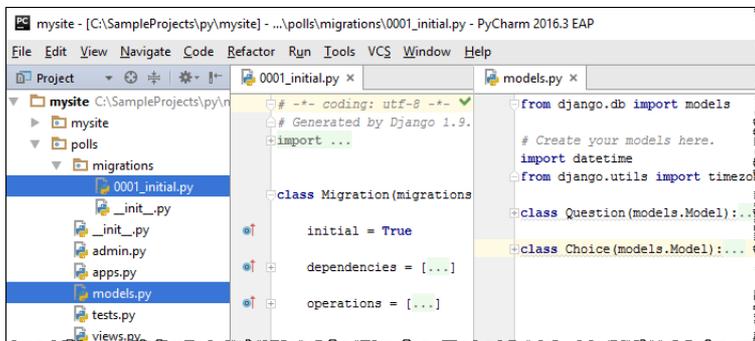


Creating database

We have to create tables for the new model. For this purpose, we'll use the magic `Ctrl+Alt+R` shortcut to invoke the `manage.py` console. First command to perform is `makemigrations polls`:



Thus you've told Django that two new models have been created, namely, `Choice` and `Question`, and created a migration:



Next, after the prompt, type the following command:

```
sqlmigrate polls 0001
```

```

manage.py@mysite
manage.py@mysite > sqlmigrate polls 0001
"C:\Program Files (x86)\JetBrains\PyCharm 163.4836\bin\runnerw.exe" "C:\Python3\Python 3.4.4\python.exe"
BEGIN;
--
-- Create model Choice
--
CREATE TABLE "polls_choice" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "choice_test"
--
-- Create model Question
--
CREATE TABLE "polls_question" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "question_t
--
-- Add field question to choice
--
ALTER TABLE "polls_choice" RENAME TO "polls_choice_old";
CREATE TABLE "polls_choice" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "choice_test"
INSERT INTO "polls_choice" ("choice_test", "votes", "id", "question_id") SELECT "choice_test"
DROP TABLE "polls_choice_old";
CREATE INDEX "polls_choice_7aa0f6ee" ON "polls_choice" ("question_id");

manage.py@mysite >

```

Finally, run `migrate` command to actually create these tables in your database:

```

manage.py@mysite
manage.py@mysite > migrate
"C:\Program Files (x86)\JetBrains\PyCharm 163.4836\bin\runnerw.exe" "C:\Python3\Python 3.4.4\python.exe"
Operations to perform:
  Apply all migrations: polls, admin, auth, sessions, contenttypes
Running migrations:
  Rendering model states... DONE
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying polls.0001_initial... OK
  Applying sessions.0001_initial... OK

Process finished with exit code 0
manage.py@mysite >

```

Performing administrative functions

First thing, create a superuser. To do that, type the `createsuperuser` command in the `manage.py` console, specify your email address, and password:

```

manage.py@mysite
manage.py@mysite > createsuperuser
"C:\Program Files (x86)\JetBrains\PyCharm 163.4836\bin\runnerw.exe" "C:\Python3\Python 3.4.4\python.exe"
Username : wombat
Email address: wombat@gmail.com
Warning: Password input may be echoed.
Password: Password123
Warning: Password input may be echoed.
Password (again): Password123
Superuser created successfully.

Process finished with exit code 0
manage.py@mysite >

```

Since we've decided to enable site administration, PyCharm has already uncommented the corresponding lines in the file `urls.py`.

Open the `admin.py` file in the `polls` directory for editing, and see the following code that already exists:

```

from django.contrib import admin

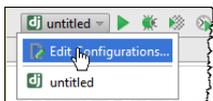
```

However, we need to enable editing functionality for the admin site.

Preparing run/debug configuration

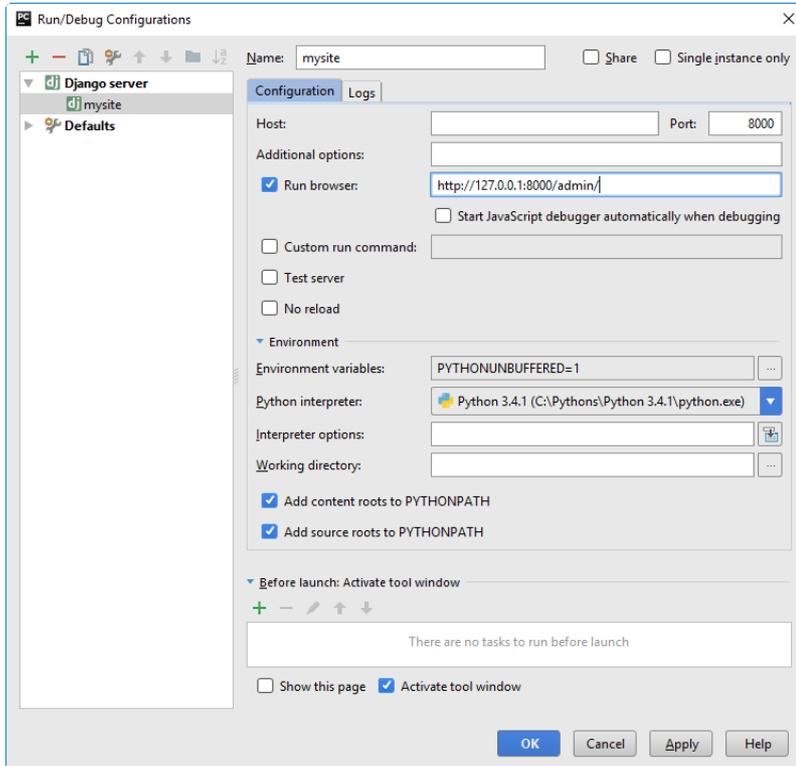
We are now ready to go to the admin page. Sure, it is quite possible to run the Django server, then go to your browser, and type the entire URL in the address bar, but with PyCharm there is an easier way: use the pre-configured Django server run configuration with some slight modifications.

To open this run/debug configuration for editing, on the main toolbar, click the run/debug configurations selector, and then choose Edit Configurations (or choose Run | Edit Configurations on the main menu):



In the `Run/Debug Configuration` dialog box, give this run/debug configuration a name (here it is `mysite`), enable running the application in the default browser

(select the checkbox Run browser) and specify the page of the site to be opened by default (here this page is <http://127.0.0.1:8000/admin/>):



Launching the admin site

Now, to launch the application, press `Shift+F10`, or click  on the main toolbar to open the standard Django site login page:

Username:

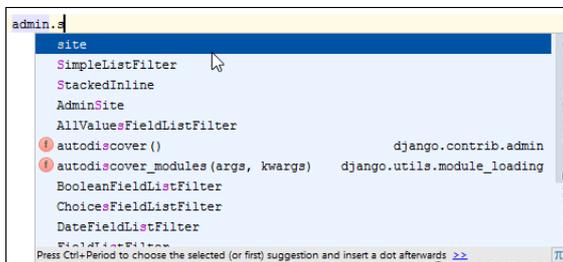
Password:

After you log in, the administration page is displayed. It has a section Authentication and Authorization (Groups and Users), but Polls is not available. Why so?

We must tell admin that Question objects have an admin interface; to do that, let's open the file `polls/admin.py` for editing (select it in Project view and press `F4`), and type the following code:

```
from django.contrib import admin
from .models import Question #this line added
admin.site.register(Question)#this line added
```

Again pay attention to the code completion:



Refresh the page and see that Polls section with Questions appeared:

Django administration

Site administration

AUTHENTICATION AND AUTHORIZATION	
Groups	+ Add Change
Users	+ Add Change
POLLS	
Questions	+ Add Change

Click Add to create some questions.

Editing admin.py

However, each question has a number of choices, but choices are still not available. Again, open for editing the file `polls/admin.py` and change it as follows:

```
from django.contrib import admin
from .models import Choice, Question

class ChoiceInline(admin.TabularInline):
    model = Choice
    extra = 3

class QuestionAdmin(admin.ModelAdmin):
    fieldsets = [
        (None, {'fields': ['question_text']}),
        ('Date information', {'fields': ['pub_date'], 'classes': ['collapse']}),
    ]
    inlines = [ChoiceInline]

admin.site.register(Question, QuestionAdmin)
```

Now look at the Change question page:

Change question | Django

127.0.0.1:8000/admin/polls/question/1/change/

Django administration WELCOME, WOMBAT. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home > Polls > Questions > Do you like ice cream?

Change question HISTORY

Question text:

Date information (Show)

CHOICE TEST	VOTES	DELETE?
<input type="text" value="Yes"/>	<input type="text" value="1"/>	
<input type="text" value="No"/>	<input type="text" value="1"/>	
<input type="text" value="What is it?"/>	<input type="text" value="1"/>	

[+ Add another Choice](#)

Writing views

Open the file `polls/views.py` for editing and type the following Python code:

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("Hello, world. You're at the polls index.")
```

Next, add a new file to the `polls` directory with the name `urls.py` and type the following code in it:

```
from django.conf.urls import url
from . import views
urlpatterns = [
    url(r'^$', views.index, name='index'),
]
```

Next, open for editing the file `mysite/urls.py` (which PyCharm has already created for you) and add a URL for the index page. You should end up with the following code:

```
from django.conf.urls import include, url
from django.contrib import admin
urlpatterns = [
    url(r'^polls/', include('polls.urls')), #this line added
    url(r'^admin/', admin.site.urls),
]
```

Now, open the page `127.0.0.1:8000/polls/` and enjoy:



Next, let's add more views. Again, add the following code to the file `polls/views.py`:

```
def detail(request, question_id):
    return HttpResponse("You're looking at question %s." % question_id)

def results(request, question_id):
    response = "You're looking at the results of question %s."
    return HttpResponse(response % question_id)

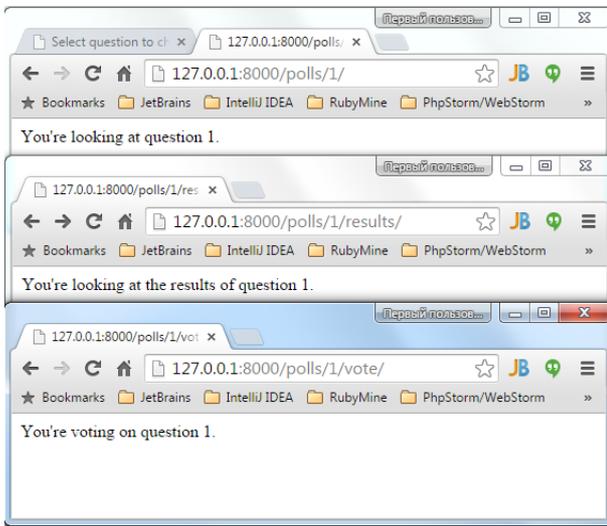
def vote(request, question_id):
    return HttpResponse("You're voting on question %s." % question_id)
```

Wire these new views into the `polls.urls` module by adding the following `url()` calls:

```
from django.conf.urls import url
from . import views

urlpatterns = [
    url(r'^$', views.index, name='index'),
    # ex: /polls/5/
    url(r'^(?P<question_id>[0-9]+)/$', views.detail, name='detail'),
    # ex: /polls/5/results/
    url(r'^(?P<question_id>[0-9]+)/results/$', views.results, name='results'),
    # ex: /polls/5/vote/
    url(r'^(?P<question_id>[0-9]+)/vote/$', views.vote, name='vote'),
]
```

If you now open the corresponding pages in your browser, you will see, for example:



Creating Django templates

As you see, the design of these pages is hard-coded in views. So, to make it more readable, you have to edit the corresponding Python code. Let's then separate the visual representation of the output from Python - to do that, let's create templates.

Open for editing the file `polls/views.py` and replace its contents with the following code:

```
from django.http import HttpResponseRedirect, HttpResponseRedirect
from django.shortcuts import get_object_or_404, render
from django.urls import reverse

from .models import Question, Choice

def index(request):
    latest_question_list = Question.objects.order_by('-pub_date')[:5]
    context = {'latest_question_list': latest_question_list}
    return render(request, 'polls/index.html', context)

def detail(request, question_id):
    question = get_object_or_404(Question, pk=question_id)
    return render(request, 'polls/detail.html', {'question': question})

def results(request, question_id):
    question = get_object_or_404(Question, pk=question_id)
    return render(request, 'polls/results.html', {'question': question})

def vote(request, question_id):
    question = get_object_or_404(Question, pk=question_id)
    try:
        selected_choice = question.choice_set.get(pk=request.POST['choice'])
    except (KeyError, Choice.DoesNotExist):
        return render(request, 'polls/detail.html', {
            'question': question,
            'error_message': "You didn't select a choice.",
        })
    else:
        selected_choice.votes += 1
        selected_choice.save()
        return HttpResponseRedirect(reverse('polls:results', args=(question.id,)))
```

By the way, pay attention to the import assistant that helps you create import statements.

The first thing you notice is an unresolved reference to the page `index.html`:

```

views.py x
3 from django.urls import reverse
4
5 from .models import Question, Choice
6
7
8 def index(request):
9     latest_question_list = Question.objects.order_by('-pub_date')[:5]
10    context = {'latest_question_list': latest_question_list}
11    return render(request, 'polls/index.html', context)
12

```

PyCharm suggests a **quick fix**: if you click the light bulb, or press `Alt+Enter`, the corresponding template file is created in the templates folder (note that PyCharm also creates the directory `polls` where this template should reside):

```

views.py x
index()
3 from django.urls import reverse
4
5 from .models import Question, Choice
6
7
8 def index(request):
9     latest_question_list = Question.objects.order_by('-pub_date')[:5]
10    context = {'latest_question_list': latest_question_list}
11    return render(request, 'polls/index.html', context)
12
13
14 def detail(request, question_id):
15    question = get_object_or_404(Ques
16    return render(request, 'polls/det
17

```

By now, the file `index.html` is empty. Add the following code to it:

```

{% load staticfiles %}
<link rel="stylesheet" type="text/css" href="{% static 'polls/style.css' %}" />
{% if latest_question_list %}
    <ul>
        {% for question in latest_question_list %}
            <li><a href="{% url 'detail' question.id %}">{{ question.question_text }}</a></li>
        {% endfor %}
    </ul>
{% else %}
    <p>No polls are available.</p>
{% endif %}

```

Note **code completion** in the template files! For example, when you type the opening `{%`, PyCharm adds the matching closing one `%}` automatically, placing the caret at the location of the future input. In `HTML` tags, code completion is also available.

Pay attention to the icons  and  that appear in the left gutter of the files `views.py` and `index.html` respectively.

```

views.py x
index()
7
8 def index(request):
9     latest_question_list = Question.objects.order_by('-pub_date')
10    context = {'latest_question_list': latest_question_list}
11    return render(request, 'polls/index.html', context)
12

index.html x
1 {% load staticfiles %}
2 <link rel="stylesheet" type="text/css" href="{% static 'polls/sty
3 {% if latest_question_list %}
4 <ul>
5     {% for question in latest_question_list %}
6     <li><a href="{% url 'polls:detail' question.id %}">{{ questio
7     {% endfor %}
8 </ul>

```

These icons enable you to jump between a view method and its template straight away. Read more about this kind of navigation in the articles [Navigating Between Templates and Views](#) and [Part 6. Django-Specific Navigation](#).

Using a stylesheet

As you can see in the view file `index.html`, there is a reference to a stylesheet, and it's unresolved:

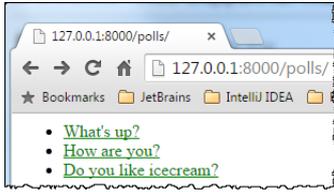
Resolve this reference in the following way:

1. Create directory. To do that, in the Project view, select the Python package `polls`, and then press `Alt+Insert`. On the pop-up menu that appears, choose `Directory`, and specify the name of the directory structure `static/polls`.
2. Next, create a style sheet in this directory. To do that, choose the innermost directory `polls`, press `Alt+Insert`, choose the option `Stylesheet`, and enter style in the dialog box that opens.
3. Provide some contents to the created stylesheet, depending on your preferences. For example, we'd like to see a bulleted list of questions colored green:

```
li a {
  color: green;
}
```

Here we are!

Now let's check the list of available polls. Our admin site is already running, and the easiest way to visit the page that contains the list of polls (the index page), is to specify its URL: in the address bar of the browser, instead of `/admin/`, type `/polls/`:



Test it...

Now let's see how PyCharm helps simplify testing your application.

There is already the file `tests.py` in the `polls` directory. By now, this file is empty. Naturally, it is advisable to place the new tests in this particular file. For example, we'd like to make sure our poll is not empty:

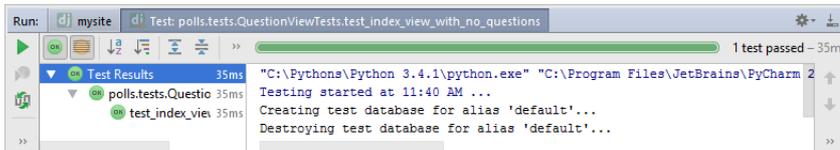
```
import datetime
from django.core.urlresolvers import reverse
from django.test import TestCase
from django.utils import timezone
from .models import Question

def create_question(question_text, days):
    time = timezone.now() + datetime.timedelta(days=days)
    return Question.objects.create(question_text=question_text, pub_date=time)

class QuestionViewTests(TestCase):
    def test_index_view_with_no_questions(self):
        """
        If no questions exist, an appropriate message should be displayed.
        """
        response = self.client.get(reverse('index'))
        self.assertEqual(response.status_code, 200)
        self.assertContains(response, "No polls are available.")
        self.assertQuerysetEqual(response.context['latest_question_list'], [])
```

To run this test, right-click the background of the file `tests.py` in the editor, choose the option Run, or just press `Ctrl+Shift+F10`. PyCharm suggests two options: run unittest (which is defined as the [default test runner](#)), or a Django test.

The test results show in the [Test Runner](#) tab of the Run tool window:



Summary

This brief tutorial is over. You have successfully created and launched a simple Django application. Let's repeat what has been done with the help of PyCharm:

- a Django project and application has been created
- the Django server launched
- a database configured
- models, views and templates created
- the application launched
- tests created and executed

Getting Help

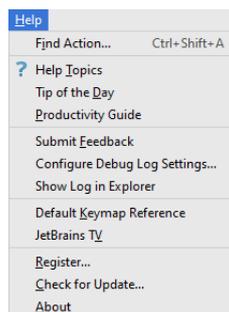
There are a lot of ways to obtain information about PyCharm and its features, report issues and get support.

In this section:

- Getting Help
 - [Overview](#)
 - [Menu commands](#)
- [Using Help Topics](#)
- [Using Tips of the Day](#)
- [Using Online Resources](#)
- [Using Productivity Guide](#)
- [Reporting Issues and Sharing Your Feedback](#)
- [Keymap Reference](#)

Overview

Use the Help menu commands:



Menu commands

Menu
Keyboard
Description
item **shortcut**

Find Action		Choose this command to invoke an action by its name .
Keymap Reference		Choose this command to see the PyCharm shortcuts map in PDF format.
Demos and Screencasts		Choose this command to see the PyCharm demo videos and screencasts on YouTube .
Help		Choose this command to visit PyCharm online Help topics.
Tip of the Day		Choose this command to show an arbitrary tip. Refer to the section Using Tips of the Day .
Productivity Guide		Choose this command to show productivity guide .
Submit Feedback		Choose this command to report your overall impression of PyCharm to the support service. Refer to the section Reporting Issues and Sharing Your Feedback .
Show Log in Explorer/Finder		Choose this command to find PyCharm's log. Refer to the section Reporting Issues and Sharing Your Feedback for details.
Edit Custom Properties		Choose this command to open the custom file <code>idea.properties</code> , located under the user home. If this file does not exist, PyCharm suggests to create it. Refer to the section Tuning PyCharm for details.
Edit Custom VM Options		Choose this command to open the custom file <code>*.vmoptions</code> , located under the user home. If this file does not exist, PyCharm suggests to create it. Refer to the section Tuning PyCharm for details.
Debug Log Settings		Choose this command to change logging level for a category. Choosing this command leads to opening the Custom Debug Log Configuration dialog box, where you have to type the log categories names, separated with new lines. Refer to the section Reporting Issues and Sharing Your Feedback .
Developer Community		Choose this command to open the PyCharm community page .
Register...		Choose this command to register PyCharm.
Check for Updates...		Choose this command to obtain information about the current version, and the availability of newer versions of PyCharm. Refer to Updates page. This command is available on Windows/Linux. On Mac OS it appears on the PyCharm menu.
About		Choose this command to obtain information about the current version of PyCharm, current build, etc. Press  to close the popup window. This command is available on Windows/Linux. On Mac OS it appears on the PyCharm menu.

Using Help Topics

In this section:

– [Documentation structure](#)

– [Built-in documentation](#)

This type of documentation is available for Professional edition only.

– [Online documentation](#)

This type of documentation is available for both Professional and Community editions.

Documentation structure

PyCharm documentation has the following structure:

Meet PyCharm

This part contains system requirements and installation information, quick start guide that helps you get a grip of PyCharm, the section [First Steps](#) with the tutorials that help you perform the most basic actions - create and run your first application.

Migration Guides

[This part](#) is intended for the users of Vim, Emacs, or Sublime Text, who are thinking about switching to PyCharm.

How to

This contains information related to the platform features (such as, for example, [using the PyCharm editor](#), [tool windows](#), or [version control](#)), and [language- and framework-specific guidelines](#).

Reference

This part contains the miscellaneous information, which includes the [basic concepts, or essentials](#), [dialogs reference](#), [icons reference](#), and more.

Using built-in documentation

Built-in documentation enables you to browse through the topics using the table of contents, find occurrences in the Search tab, or use the detailed Index that contains all keywords from all topics.

To bring up help contents, do one of the following

- On the main menu, choose Help | Help Topics.
- Press **F1**.
- Click Help button, if it is available.

To find a particular piece of information

1. Click the Search tab of the help viewer.
2. In the search field, type the search string and press **Enter**.
3. In the list of topics that contain occurrences of the search string, select the desired one. Occurrences are highlighted in the right pane of the help viewer.

To find a keyword in the Index tab

1. Click the Index tab of the help viewer.
2. In the search field, type the desired keyword and press **Enter**. The caret rests at the first occurrence of the keyword. Every time you press **Enter**, the caret moves to the next occurrence of the keyword. To see the information about a keyword, select one of its sub-entries.

Online documentation

On the PyCharm site, find online documentation:

[PyCharm web help](#)

The online version makes it possible to find entries in the table of contents, browse documentation with the table of contents, rate topics and express your opinion. The layout of online documentation consists of:

Table of contents pane

This pane shows the table of contents.

- Use this pane to browse through the topics.
- Click  button to show or hide this pane.
- If for some reason your browser fails to show actual table of contents, refresh the page.

Topics pane

This pane shows the topic that is currently selected in the table of contents.

_ In the Keymap drop-down list, choose the platform you want to view keyboard shortcuts in (Windows/Linux, macOS etc.)

Finding a piece of information in the table of contents

1. Switch to the Table of contents pane.
2. In the Search PyCharm help field, type your query.
3. Press **Enter**.

The table contents shrinks to show the search results. Click the desired entry to show the corresponding page in the Topic pane.

Using Tips of the Day

Tips of the Day provide a collection of useful and interesting hints. They show up every time you start PyCharm.

To show Tips of the Day

- Choose Help | Tip of the Day on the main menu.

To navigate through the collection of tips

- Use the Previous and Next buttons.

To suppress Tips of the Day

- In the Tips of the Day window, clear the check box Show Tips on Startup.

Using Online Resources

If built-in documentation fails to answer your questions, you can find more information on the web. The following resources are available:

- [Official PyCharm home page](#).
- [PyCharm and Python plugin community](#)
- [PyCharm Resources](#) page contains the links to keymaps, [online version of documentation and tutorials](#), [blog](#), and more.

Finally, do not miss the JetBrains TV (Help | JetBrains TV). On the [JetBrains TV](#) page, choose the PyCharm channel, and watch screencasts.

Using Productivity Guide

PyCharm smartly analyzes the features you use most often during your development sessions, and reminds you of the features you might have missed.

The [Productivity Guide](#) dialog, available in Help | Productivity Guide, displays the list of features with usage statistics and tips.

Besides analysing your personal usage of features, you can discover similar features that you've never used. One of the ways to do so is to sort features by Group, and look for unused features that are next to the frequently used ones. You can quickly check how to use the feature by selecting it and studying the corresponding tip that opens.

Reporting Issues and Sharing Your Feedback

PyCharm provides various means to report problems and seek for assistance. In this section:

- [Locating PyCharm log](#)
- [Configuring PyCharm log settings](#)
- [Reporting issues](#)
- [Sharing feedback](#)
- [Seeking assistance](#)

Locating PyCharm log

On certain occasions, you will be required to attach the PyCharm log to an email to the support service. You can easily locate the log file as described below.

- On the main menu, choose Help | Show Log in Explorer (Windows and Linux), or Help | Show Log in Finder (macOS). The Explorer/Finder opens, with the log file selected.

Configuring PyCharm log settings

To avoid editing the `log.xml` file itself, PyCharm suggests a handy dialog box to change logging level for a category. This file resides under the `bin` directory of PyCharm installation.

1. On the main menu, choose Help | Configure Debug Log Settings.
2. In the dialog box that opens, type the log categories names, separated with new lines.

While editing `log.xml`, keep in mind the following:

- It is not recommended to change `log.xml`, because from time to time it causes problems with patches.
- Editing `log.xml` should be done in tight contact with the support service. The reason is that the users might be unaware of the modules names to be specified, while the support service can suggest modules for the better diagnostics.

Reporting issues

1. Open the PyCharm tracking system at <https://youtrack.jetbrains.com/>.
If you are not yet registered, please do so.
2. Click Create issue.
3. On the page that opens, choose PyCharm from the Project drop-down list.
4. Describe your problem and provide a brief summary of it in the Description and Summary fields respectively.
5. If necessary, attach a screenshot that illustrates your problem.
6. Click Create issue when ready.

Sharing feedback

To share your feedback, do one of the following:

- Choose Help | Submit Feedback, which redirects you to the online feedback form.
This form enables you to create a PyCharm-specific [YouTrack](#) issue.
- Click the Comments link on each page of online version of the PyCharm documentation. In the Comments area, vote for the page, and leave your comments.

Seeking assistance

To find assistance, do one of the following:

- Apply to the [JetBrains Support](#)
- Write to the support service. Use the following address:
 - intellij-support@jetbrains.com
 - pycharm-support@jetbrains.com

If necessary, attach the source code and the [PyCharm log](#).

- Ask the [PyCharm Community](#).

Keymap Reference

PyCharm provides the default platform-specific keymap reference in `pdf` format.

You can view the keymap either from the built-in documentation, or on the web.

– [Keymap for Windows/Linux](#) , [Keymap for macOS](#)

How to

This part provides descriptions of the platform and language-specific procedures.

- [General Guidelines](#)
- [Language and Framework Specific Guidelines](#)

General Guidelines

This part provides descriptions of procedures required to fulfil the platform tasks with PyCharm.

- [Guided Tour Around PyCharm User Interface](#)
- [PyCharm Tool Windows](#)
- [PyCharm Editor](#)
- [Configuring Project and IDE Settings](#)
- [Creating and Managing Projects](#)
- [Configuring Project Structure](#)
- [File and Code Templates](#)
- [Live Templates](#)
- [Populating Projects](#)
- [Generating Code](#)
- [Auto-Completing Code](#)
- [Analyzing Applications](#)
- [Code Inspection](#)
- [Intention Actions](#)
- [Creating and Optimizing Imports](#)
- [Viewing Reference Information](#)
- [Viewing Pages with Web Contents](#)
- [Navigating Through the Source Code](#)
- [Searching Through the Source Code](#)
- [Refactoring Source Code](#)
- [Working with Run/Debug Configurations](#)
- [Running](#)
- [Debugging](#)
- [Testing](#)
- [Deployment. Working with Web Servers](#)
- [Code Coverage](#)
- [Using Language Injections](#)
- [Using Local History](#)
- [Version Control with PyCharm](#)
- [Managing Tasks and Context](#)
- [Managing Plugins](#)
- [Comparing Files and Folders](#)
- [Working with Background Tasks](#)
- [Text Direction](#)
- [Working with Consoles](#)
- [Working with Diagrams](#)

Guided Tour Around PyCharm User Interface

This chapter gives you insight into how PyCharm user interface is organized to help you find your way through your working environment.

Tip This chapter outlines the default (out-of-the-box) IDE interface layout. Note that plugins and other add-ons you are installing may change the way your IDE looks and behaves, for example there can be extra command buttons or menu items appearing in uncommon locations.

When you first run PyCharm, or have no open project, PyCharm displays the [Welcome screen](#), which enables quick access to the major entry points. When a project is opened, PyCharm displays the main window. This window is made up of the logical areas, which are shown on the picture below, marked with number labels.



1. **Menu and toolbars** - the main menu and toolbars let you carry out various commands.
2. **Navigation bar** that helps navigate through the project and open files for editing.
3. **The status bar** - indicates the status of your project, the entire IDE, and shows various warning and information messages.
4. **The editor** - here you create and modify the code.
5. **PyCharm tool windows** - secondary windows that provide access to various specific tasks (project management, search, running and debugging, integration with version control systems, etc.).

Welcome Screen

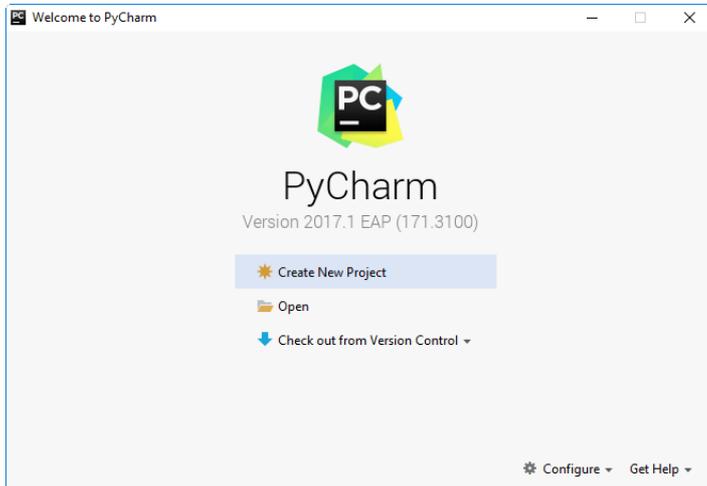
On this page:

- [Overview](#)
- [Recent projects](#)
- [Quick start](#)

Overview

PyCharm displays the Welcome screen when no project is open. From this screen, you can quickly access the major starting points of PyCharm. The Welcome screen appears when you close the current project in the only instance of PyCharm. If you are working with multiple projects, usually closing a project results in closing the PyCharm window in which it was running, except for the last project, closing this will show the Welcome screen.

The Welcome screen is divided into the following sections: Quick Start and Recent Projects (if any).

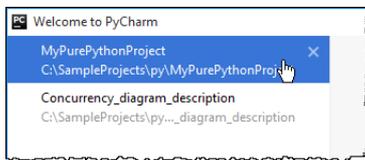


TIP Use the `Tab` key to navigate through the Welcome screen.

Recent projects

If appropriate, the left-hand pane shows a list of projects you've recently been working with. Click a project to reopen it.

To find a project of interest, start typing its name.



To delete a recent project from the list, follow these steps:

1. Use the `Tab` key to move focus into the list of Recent projects.
2. Use arrows keys to select the project you'd like to remove, or find it, as shows above.
3. Do one of the following:
 - Press Delete on your keyboard and confirm deletion in the Remove Recent Project dialog box that opens.
 - To remove the selected recent project silently, click `X` or choose Remove Selected from Welcome Screen on the context menu of the selection.

Quick start

Use the links of this section to [create a new project](#), [open](#) or [import](#) an existing project, or [check out](#) a project from version control.

Also, use the drop-down arrows (▼) Configure to [configure your working environment](#) and [default project](#), and Get Help to open help topics, tips of the day, and default keymap document.

Menus and Toolbars

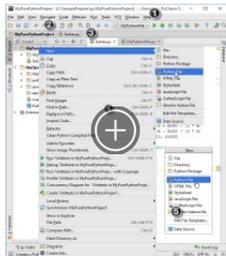
On this page:

- [Overview](#)
- [Main elements of PyCharm window](#)
- [Tips and tricks](#)

Overview

PyCharm menus and toolbars let you carry out various commands. The main menu and toolbar contain commands that affect the entire project or large portions of it. Additionally, context-sensitive pop-up menus let you perform the commands, which are specific to a part of a project, like source file, class, etc. Almost each of the commands has an associated keyboard shortcut to enable quicker access to it.

Use the check commands of the View menu to show or hide the main elements of the PyCharm window. For example, if you want to show the main toolbar, make sure that the check command `View | Toolbar` is selected.



Main elements of PyCharm window

1. Main menu

The main menu contains commands for opening, creating projects, refactoring the code, running and debugging applications, keeping files under version control and more.

2. Main toolbar

The main toolbar contains buttons that duplicate the essential commands for quicker access. You can hide the main toolbar, using the checked command on the toolbar context menu.

By default, the main toolbar is hidden. To show it, select the check command `View | Toolbar` on the main menu.

3. Navigation bar

[Navigation bar](#) is a quick alternative to the Project tool window.

4. Context menus

These menus, available with right-click, contain commands applicable to the current context.

5. Pop-up menus

These menus, available with `Alt+Insert`, contain commands applicable to the current context.

Tips and tricks

- Show or hide the main elements of PyCharm UI using the View menu.
- Descriptions of the actions from all the menus and toolbar buttons are displayed in the left side of the Status bar.
- If you know which action you want to perform, but do not know where to find the appropriate command, just press `Ctrl+Shift+A`, type some part of the name of action you want to perform, and select the desired action from the suggestion list.

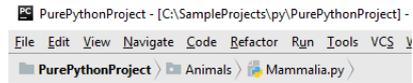
Navigation Bar

On this page:

- [Introduction](#)
- [Toggling the Navigation Bar](#)
- [Tips and tricks](#)

Introduction

Navigation Bar is a quick alternative to the [Project view](#). Use this tool to [navigate](#) through the project and [open files for editing](#).



Toggling the Navigation Bar

To show the Navigation Bar, do one of the following

- On the View menu, select the check command Navigation Bar.
- Press `Alt+Home`.

To hide the Navigation Bar

- On the View menu, clear the check command Navigation Bar.

Tips and tricks

Please note the following:

- When the [main toolbar](#) is hidden, the Navigation Bar shows the run/debug configuration selector, run  and debug , version control buttons (if version control integration is enabled) and [search everywhere](#) magnifying glass .
- When the Navigation bar is hidden, press `Alt+Home` to open it floating.
- Pressing `Escape` returns focus to the editor.

Status Bar

On this page:

- [Introduction](#)
- [Status Bar icons](#)

Introduction

PyCharm Status Bar indicates the current IDE state and lets you carry out certain environment maintenance tasks.



Status Bar icons

IconDescription

	Click to toggle showing or hiding tool window bars. Also double press and hold (for macOS) or (for Windows or *NIX) to show hidden tool window bars. Refer to the procedure description .
<small>Load offline inspection results</small>	This section of the Status bar shows description of a command, currently selected on the main menu, context menu, or a toolbar.
	Click this icon to invoke the Background Tasks manager . Visibility of this icon in the Status bar depends on a launched background task.
	The first two numbers denote the current caret position in the editor(line and column). If a selection is made in the editor, PyCharm shows the length of the current selection after slash.
	This Lock icon indicates the read-only or writable attribute of the current file in the editor. To toggle the file attribute, click the Lock icon or use the File Make File Writable/Read-only command on the main menu.
	View and change line endings of the current file in the editor.
	View and change encoding of the current file in the editor.
	Click this icon to navigate to the pending source control changelists in the Incoming tab of the Version Control tool window.
	Hovering your mouse pointer over the icon shows the current code inspection profile at the tooltip. Clicking the Hector icon results in showing a dialog box with the following functions: <ul style="list-style-type: none">– Highlighting level. Use the slider to change highlighting level for the current file, or configure inspection profile. Depending on the highlighting level selected by the slider, Hector keeps an eye on the code (Inspection level), turns half face (Syntax), or averts his face from the code (None).– Power Save Mode. Select this check box to minimize power consumption of your computer on account of eliminating the background operations. To indicate that the mode is on, Hector fades . When Power Save Mode is on, PyCharm reduces its functionality to the one of a text editor, by not executing expensive background activities that drain laptop battery. These activities include error highlighting and on-the-fly inspections, autopopup code completion . In the files with Django or any other template language support, it is possible to separately configure highlighting for the entire file (for example, HTML), and for a template language (for example, when the user want to see all the warnings in a template, but doesn't want to see them in JS). You can also toggle Power Save Mode through the File Power Save Mode command on the main menu.
	Indicates that there are unattended notifications. Click this icon to see the notification descriptions in the Event Log tool window. Alternatively, when the icon is empty, there are no new notifications.
	This blinking icon indicates that internal IDE errors have occurred. Click to view the error descriptions and submit reports.
	Shows the current heap level and memory usage. Visibility of this section in the Status bar is defined by the Show memory indicator check box in the Appearance page of the Settings/Preferences dialog. It is not shown by default. Click the memory indicator to run the garbage collector.
	Shows logged in to or logged out from IntelliJ Configuration Server status. Click to show the IntelliJ Configuration Server Settings dialog box. This is only available upon installing the IntelliJ Configuration Server plugin.

Note More icons appear in the Status Bar as you download and install plugins.

PyCharm Viewing Modes

On this page:

- [Basics](#)
- [Toggling the full screen mode](#)
- [Toggling the presentation mode](#)
- [Toggling the distraction-free mode](#)
- [Toggling the viewing modes in the Switch pop-up list](#)

Basics

PyCharm provides special view modes:

- **Full Screen mode** allows you to use the entire screen for coding. This removes all menus from view, as well as the operating system controls. However, you can use context menus and keyboard shortcuts. The main menu is also available when you hover the mouse pointer over the top of the screen.
- **Presentation mode** is similar to the **Full Screen mode**, but it is designed for making presentations that involve coding with PyCharm. In this mode, PyCharm increases the font size and hides everything except the editor. If necessary, tool windows can be also displayed in this view using the corresponding items in the View | Tool Windows menu.
- **Distraction-free mode** shows no toolbars, no tool windows, no editor tabs; the code is center-aligned, etc.

These actions are available only through the View menu. By default they are not mapped to any shortcuts but you can create your own shortcuts as described in [Configuring Keyboard Shortcuts](#).

Toggling the full screen mode

Besides [manipulating the tool windows](#) (show/hide or resize them), PyCharm makes it possible to maximize the entire product window, hiding the main menu.

- To switch to the full screen mode, choose View | Enter Full Screen on the main menu.
- To exit the full screen mode, choose View | Exit Full Screen on the main menu.

Toggling the presentation mode

In the presentation mode, the editor occupies the entire screen, while all the other PyCharm components are hidden.

Besides that, the font size in this mode is larger than usual. You can define the font size for the presentation mode in the [Appearance page](#) of the Settings dialog.

- To switch to the presentation mode, choose View | Enter Presentation Mode on the main menu.
- To exit the presentation mode, choose View | Exit Presentation Mode on the main menu.

Toggling the distraction-free mode

In the distraction-free mode, the editor occupies the entire PyCharm frame, without any editor tabs and tool-window buttons. The code is center-aligned.

- To switch to the distraction-free mode, choose View | Enter Distraction-Free Mode on the main menu.
- To exit the distraction-free mode, choose View | Exit Distraction-Free Mode.

Toggling the viewing modes in the Switch pop-up list

1. Press `Ctrl+Back Quote` or choose View | Quick Switch Scheme on the main menu.
2. In the Switch pop-up list that opens, choose View mode.
3. On the context menu, choose the required mode. The contents of the menu depend on your current mode:
 - Enter Presentation Mode/Exit Presentation Mode
 - Enter Distraction Free Mode/Exit Distraction Free Mode
 - Enter Full Screen/Exit Full Screen

Setting Background Image

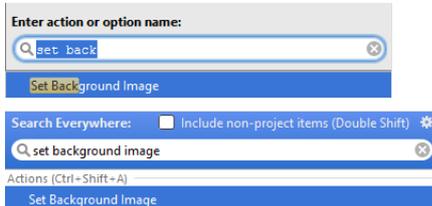
PyCharm allows you to define any image as a background. So doing, you can set a background image for the current project only, or for any project you open or create anew.

This feature has no keyboard shortcut (you can easily create a shortcut as described in the section [Configuring Keyboard Shortcuts](#)).

To set a background image

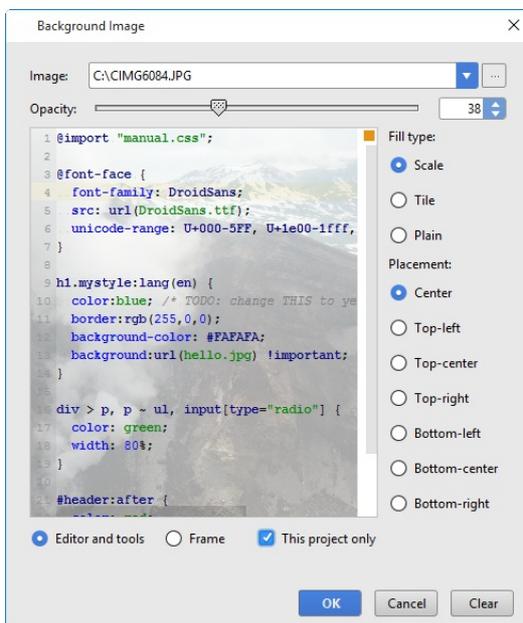
1. Do one of the following:

- Press **(Shift)** twice (see [Searching Everywhere](#)).
- Press **(Ctrl+Shift+A)** (see [Navigating to Action](#)).

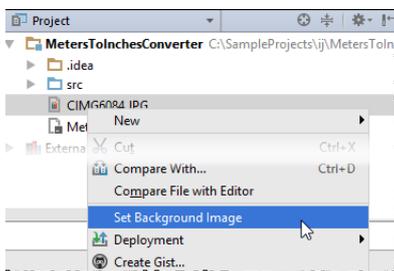


2. In the dialog box that opens, specify the image you want to use as the background, its opacity, filling and placement options. Besides that, you can choose to show background in the editor and tool windows, or in the PyCharm frame.

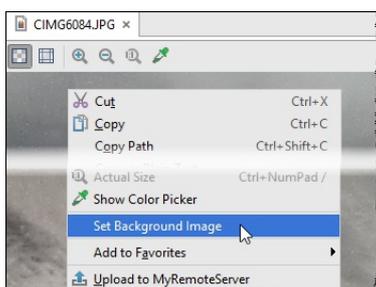
Also, select the check box **This project only** to show background in the current project, and ignore this background in the other projects.



If an image is already selected in a PyCharm project, this action is also available from the context menu of the Project tool window:



and in the image editor



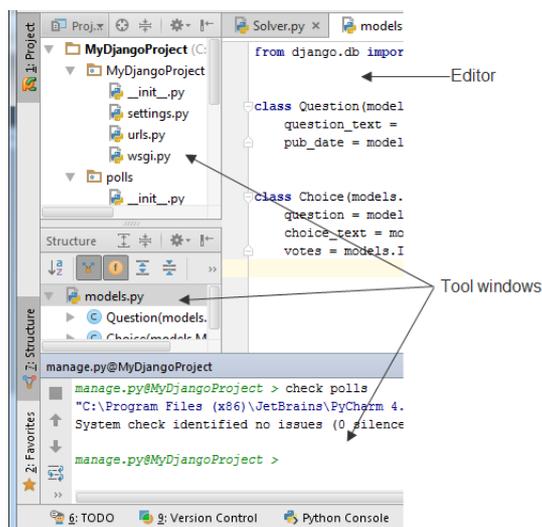
PyCharm Tool Windows

In this section:

- PyCharm Tool Windows
 - [Overview](#)
 - [Tool window quick access](#)
 - [Tool window bars and buttons](#)
 - [General tool window layout](#)
 - [Accessing tool window menus](#)
- [Manipulating the Tool Windows](#)
- [Specifying the Appearance Settings for Tool Windows](#)
- [Viewing Modes](#)
- [Speed Search in the Tool Windows](#)
- [Managing Your Project Favorites](#)

Overview

Attached to the bottom and sides of the workspace are PyCharm tool windows. These secondary windows let you look at your project from different perspectives and provide access to typical development tasks. These include project management, source code search and navigation, running and debugging, integration with version control systems, and many other specific tasks.

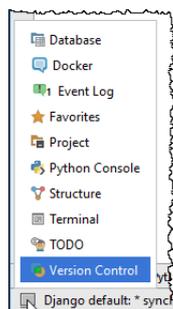


Certain tool windows are available always, that is, in any [project](#) irrespective of the project nature, contents, and configuration. The Other tool windows are available only if the corresponding [plugins](#) are enabled. There are also tool windows that only appear when certain actions are performed. For example, to invoke [Find tool window](#) you need to initialize a search.

Tool window quick access

In the lower left corner of the workspace, there is a button which initially looks like this .

If you hover the mouse cursor over this button, a menu opens that provides quick access to tool windows:



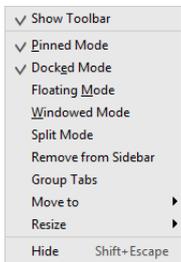
If you click this button, tool window bars and buttons are shown. At the same time the button, appearance toggles to . If you click the button again, the tool window bars and buttons are hidden again.

Tool window bars and buttons

When visible, the tool window button bars (or just tool window bars) are around the tool windows (or the [editor](#) area if the tool windows are hidden). These bars contain the buttons for showing or hiding the tool windows (tool window buttons).



The tool window buttons also provide access to tool window context menus displayed if you right-click a tool window button:



The context menus let you control the tool window [viewing modes](#) and other aspects of the tool window appearance.

Initially, there are three button bars, two at the sides of the main window and one at the bottom. You can show or hide all button bars at once by clicking  in the bottom-left corner of the workspace.

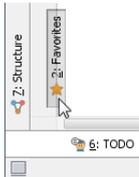
Each tool window button has the name of the corresponding tool window on it. On certain buttons, the window name may be preceded by a number, for example, 1: Project. This means that the keyboard shortcut `Alt+<number>` is available for showing or hiding the window. You can, for example, show or hide the Project tool window by pressing `Alt+1`.

You can turn showing the window access numbers on the buttons on and off in the [Appearance settings](#).

The buttons for visible tool windows and for hidden ones are shown differently.



You can rearrange the tool windows by dragging-and-dropping the tool window buttons onto a different tool window bar (or to a different corner of the same bar). As a result, the tool window becomes attached to the bar you've moved the window button to.

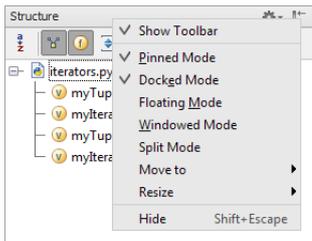


General tool window layout

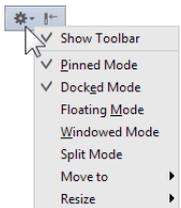
Generally, all tool windows are organized in a similar way.



At the top of the window is a title bar. When you right-click the title bar, a menu for managing the window appearance and contents is shown.



The title bar contains two buttons in its right-hand part. The first of these buttons () opens a menu for managing the tool window [viewing modes](#). Note that the menu options are a subset of the title bar context menu.



The second of the buttons () is for hiding the tool window. When used in combination with the `Alt` key, clicking this button hides all the windows attached to the same tool window bar.

Underneath the title bar are the toolbar and content pane. Depending on the window, the toolbar may be above or to the left of the content pane.

The toolbar buttons, generally, are window-specific. However, the windows with similar purposes may contain similar controls on their toolbars.

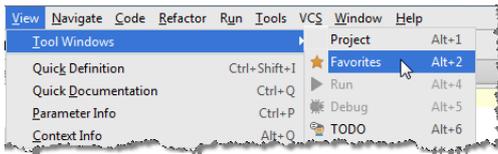
In most cases, a function associated with a toolbar button may also be accessed from the main or context menu, or may have a keyboard equivalent.

The content panes may be plain or contain two or more "layers" (views) represented, for example, by tabs. There are also tool windows with the content pane

part shown on a separate tab in the editor area.

Accessing tool window menus

– Use the View | Tool Windows menu to show or hide the tool windows.



Note Some of the tool windows (for example, [Find](#) tool window) appear in the View menu ONLY when an action has been performed for the first time. So, unless you do something, say, find, these tool windows are not visible in the View menu.

Manipulating the Tool Windows

You can manipulate the [tool windows](#) in various ways to adjust PyCharm workspace to your needs.

- [Showing a tool window](#)
- [Hiding an individual tool window](#)
- [Hiding all tool windows attached to the same tool window bar](#)
- [Hiding all tool windows](#)
- [Switching to the last active tool window](#)
- [Hiding or showing the tool window bars](#)
- [Hiding tool window buttons](#)
- [Attaching a tool window to a different tool window bar](#)
- [Resizing a tool window](#)
- [Increasing the number of tool windows shown at a time](#)
- [Saving and restoring the arrangement of the tool windows](#)

Showing a tool window

Do one of the following:

- In the lower left corner of the workspace, point to  and select the tool window in the menu that is shown.
- Click the corresponding tool window button on the [tool window bar](#).
- Choose View | Tool Windows | <tool window> in the main menu.
- If the tool window has an associated quick access number, press `Alt+<number>` (for example, `Alt+1` for the Project tool window).

Hiding an individual tool window

Do one of the following:

- Click the corresponding tool window button on the [tool window bar](#).
- Click **I*** on the tool window title bar.
- Right-click the corresponding tool window button and select Hide from the context menu.
- Right-click the tool window title bar and select Hide from the context menu.
- Choose View | Tool Windows | <tool window> in the main menu.
- If the tool window has an associated quick access number, press `Alt+<number>` (for example, `Alt+1` for the Project tool window).
- If the tool window you are going to hide is currently active, press `Shift+Escape`.

Hiding all tool windows attached to the same tool window bar

Do one of the following:

- Press and hold the `Alt` key, and click **I*** on the title bar of any of the tool windows attached to the corresponding tool window bar.
- Choose Window | Active Tool Window | Hide Side Tool Windows in the main menu. This command hides all the tool windows attached to same tool window bar as the active tool window or the last of the active tool windows.

Hiding all tool windows

Do one of the following:

- Choose Window | Active Tool Window | Hide All Windows in the main menu.
- Press `Ctrl+Shift+F12`.

Switching to the last active tool window

Do one of the following:

- Choose Window | Active Tool Window | Jump to Last Tool Window in the main menu.
- Press `F12`.

If all the tool windows are currently hidden, the last active tool window will be shown and made active.

Hiding or showing the tool window bars

You can hide all the tool window bars if you need more space in the PyCharm window:

- In the lower left corner of the workspace, click .

If the tool window bars are hidden, you can bring them back onto the screen either permanently or for a short period of time:

- To restore the tool window bars, click  in the lower left corner of the workspace.
- To see the tool window bars for a short period of time, double-press and hold the `Alt` key. The tool window bars appear on the screen making the tool window buttons accessible. The tool window bars are hidden again when you release the key.

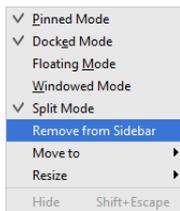
Hiding tool window buttons

PyCharm makes it possible to hide individual tool window buttons, without actually uninstalling the corresponding plugins.

To remove a tool window button from view:

Make sure that the [tool window bars are visible](#).

1. make sure that the **tool window bars are visible**.
2. Right-click the tool window button you want to hide.
3. Choose Remove from Sidebar



To restore the hidden tool window button, choose View | Tool Windows on the main menu, and then click the window with the hidden toolbar button.

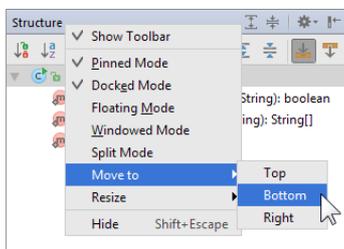
Attaching a tool window to a different tool window bar

Do one of the following:

- Drag the corresponding tool window button onto the desired tool window bar (top, left, bottom or right).

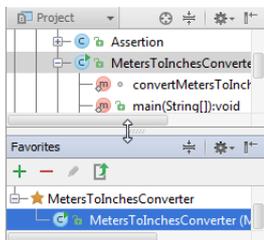


- Right-click the corresponding tool window button or the tool window title bar to open the context menu. Choose Move to and then select the destination tool window bar (Top, Left, Bottom or Right).



Resizing a tool window

- Hover the mouse pointer over the tool window border. When the pointer becomes a double-headed arrow, drag the border in the required direction.



- You can also resize a tool window by moving its border to left or right, or up or down in steps. The following alternatives are available for doing that:
 - Right-click the corresponding tool window button or title bar and select Resize. Then select one of the available Stretch to options.
 - Make the tool window of interest active and do one of the following:
 - Choose Window | Active Tool Window | Resize, and then select the necessary Stretch to option.
 - Press `Ctrl+Shift` in combination with the corresponding arrow key.

Increasing the number of tool windows shown at a time

To increase the number of tool windows to be shown at a time, you should appropriately set the **viewing modes** for different tool windows. Consider the following:

- Generally, for a tool window to be visible always (i.e. even when inactive), the tool window should be **pinned**.
- There are no limiting factors for the number of visible **floating** windows and ones in the windowed mode. Note that the **windowed** mode is not available if you are using macOS.
- To be able to see two windows **docked** to the same tool window bar at a time (rather than one), one of the windows should have the **split mode** off and the other one on.
- Initially, three (out of four) tool window bars are used. You can "activate" the forth tool window bar (the top one) by **moving** certain tool windows to it.

Saving and restoring the arrangement of the tool windows

You can save the way the tool window are currently arranged by choosing Window | Store Current Layout as Default in the main menu.

At a later time, you can return to the saved workspace layout by choosing Window | Restore Default Layout (`Shift+F12`).

Specifying the Appearance Settings for Tool Windows

You can change certain tool window appearance properties by specifying the corresponding Appearance settings.

To change the appearance properties for tool windows

1. In the Settings dialog, expand the Appearance&Behaviour node, and click [Appearance](#).
2. If necessary, change the settings related to tool window appearance. These are mainly in the Transparency and the Window Options sections.
For more information, see descriptions of the pages under [Appearance and Behavior](#).

Viewing Modes

PyCharm provides various viewing modes that let you control the way the tool windows are shown and behave. These modes help you keep a proper balance between quick, easy access to tool windows and maximum screen space for editing your code.

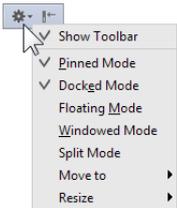
The viewing modes are set separately for each of the tool windows.

- [Ways to control the viewing modes](#)
- [Fixed / floating / windowed mode](#)
- [Docked / undocked mode](#)
- [Pinned / unpinned mode](#)
- [Split mode](#)
- [Group Tabs option](#)
- [Wide screen support](#)

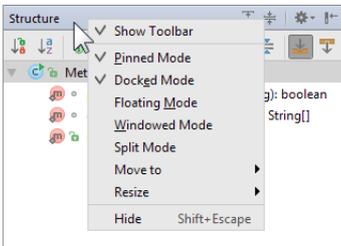
Ways to control the viewing modes

The viewing modes are set by turning the corresponding viewing options on or off. To access these options, you can use:

-  on the title bar of a tool window.



- Context menus. The context menus are accessed by right-clicking the tool window buttons or the tool window title bars.

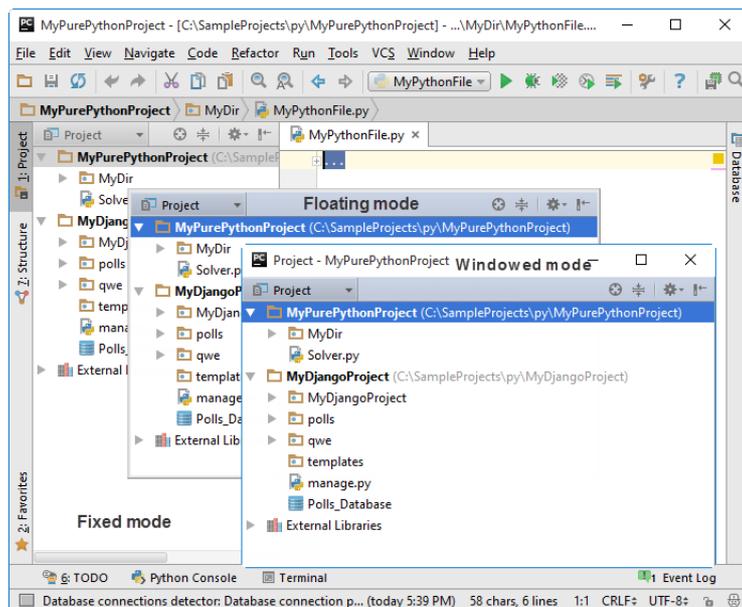


- For an active tool window: the Window | Active Tool Window menu.

Fixed / floating / windowed mode

A tool window may be **fixed**. In that case, it stays within the main window.

Alternatively, a tool window may be in the **floating** or in the **windowed** mode. Note that the **windowed** mode is not available if you are using macOS.



When in the fixed mode, one side of the tool window is attached to one of the tool window bars. The behavior of the opposite side depends on whether the window is **docked** or **undocked**.

Initially, all the tool windows are in the fixed mode (i.e. the floating and windowed modes are off).

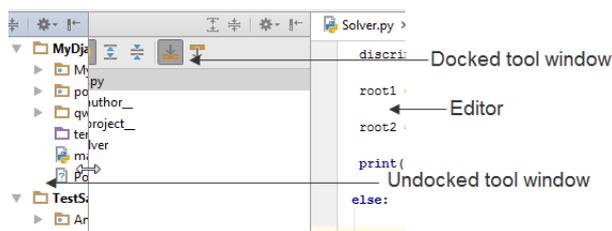
In the floating and in the windowed modes, a tool window may be moved around the screen to any place.

To switch to the floating or the windowed mode, turn on the Floating Mode or the Windowed Mode option. To bring a tool window back to the fixed mode, turn the Floating Mode and the Windowed Mode options off. See [Ways to control the viewing modes](#).

Note that for the tool windows in the windowed mode, the Window menu command Hide Active Tool Window is disabled.

Docked / undocked mode

A tool window in the **fixed mode** may be docked or undocked.



In the docked mode, all the sides of a tool window are attached to surrounding elements (the editor, other tool windows, etc.) Thus, the tool window and the adjacent elements share the space available in the main window.

When a docked tool window becomes inactive, it stays visible or is hidden depending on whether the window is **pinned or unpinned**.

Initially, all the tool windows are in the docked mode (i.e. the docked mode is on).

When undocked, all the sides of a tool window (except the one at the tool window bar) are detached from surrounding elements. The window moves to the "upper layer" covering the elements it used to share the space with. In one of the directions (along the tool window bar), the window stretches and takes all the available space. In the other direction, one of the window borders becomes loose and can be moved without affecting the sizes of other, underlying elements.

When an undocked tool window becomes inactive, it is automatically hidden.

To switch between the docked and undocked mode, turn the Docked Mode option on or off. See [Ways to control the viewing modes](#).

Pinned / unpinned mode

Pinned tool windows, generally, stay visible when becoming inactive. Unpinned tool windows in such cases are automatically hidden.

Initially, all the tool windows are pinned (i.e. the pinned mode is on).

There may be slight differences in behavior depending on the other viewing modes:

- **Undocked** tool windows are always hidden when inactive. (In the undocked mode, the tool windows are effectively unpinned.)
- **Floating** pinned tool windows, when inactive, may become semi-transparent.

To switch between the pinned and unpinned mode, turn the Pinned Mode option on or off. See [Ways to control the viewing modes](#).

Split mode

This mode has to do with how many windows **docked** to the same tool window bar may be shown at a time (one or two).

Generally, the space along a tool window bar is shared between two groups of docked tool windows.

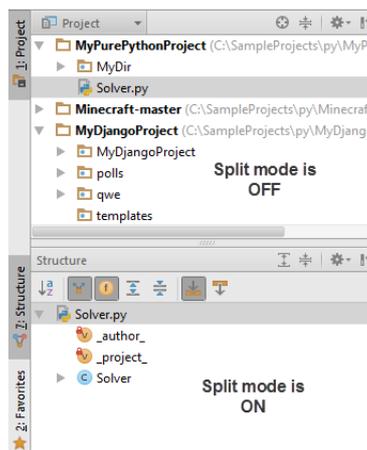
In one of the groups are the tool windows for which the split mode is off; in the other group are the ones with this mode on.

At each moment of time, only one window from each of the groups may be visible.

Thus, if all the tool windows docked to a tool window bar have the split mode off, only one tool window may be shown at a time. In this case, the tool window which is visible takes all the space available along the tool window bar. So when you make a certain window visible, the previous visible window is automatically hidden.

The same behavior is observed if the split mode is on for all the windows docked to the same tool window bar.

To be able to see two windows simultaneously, the corresponding windows should belong to different groups, that is, one of the windows should have the split mode off and the other one on.



The tool window buttons for the tool windows with different settings of the split mode are grouped and shown at different corners of the corresponding tool

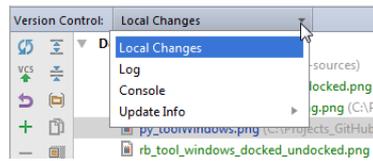
window bar. For vertical window bars, the windows with the split mode off have the buttons at the top corner; for the horizontal bars, the buttons for such windows are at the left corner.

To turn the split mode on or off see [Ways to control the viewing modes](#).

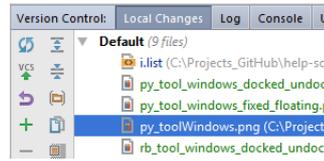
Group Tabs option

If more than one view is available in a tool window, the corresponding views may be shown on separate tabs if the Group Tabs option is off. If this option is on, the views are selected from a list.

Group Tabs: ON



Group Tabs: OFF



Wide screen support

PyCharm makes it possible for the tool windows to use the full width and height of the screen. In the Settings dialog, expand the node Appearance and Behaviour, and in the [Appearance](#) page, use the check boxes Wide screen tool window layout and Side by side layout on the left/right to optimize placement of the tool windows.

Note also that you can turn side-by-side layout on or off by `Ctrl+MouseClicked` on splitter between the tool windows.

Refer to [Appearance](#) for details.

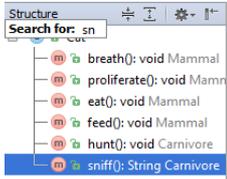
Speed Search in the Tool Windows

Speed search in the tool windows helps you find and navigate to a file or folder in the Project tool window, a member in the Structure tool window, a changelist in the Version Control tool window, an item in the TODO list, and more.

Note that speed search is performed only on expanded nodes, if a node is folded the matching items under it are not detected.

To search through a tool window

1. Select the desired tool window.
2. Start typing the item name (for instance, file, class, field, etc.). As you type, the Search for field appears over the tool window toolbar showing the entered characters, and the element selection moves to the first item that matches the specified string. The matching part of the string is highlighted.



3. If several neighboring items match the pattern, use the Up and Down keys on the keyboard to navigate among them.
4. Press `Enter` when ready. As a result, the matching item is selected in the tool window. Pressing `Escape` hides the Search for field.

Managing Your Project Favorites

On this page:

- [Basics](#)
- [Using the Project tool window to add items to favorites](#)
- [Using the editor to add items to favorites](#)
- [Creating a new favorites list](#)
- [Renaming a favorites list](#)
- [Moving an item to a different list](#)
- [Removing items from favorites](#)

Basics

You can arrange the most frequently used project items (files, folders, packages, instance and class members, etc.), [bookmarks](#), and [breakpoints](#) in the lists of favorite items (favorites). In PyCharm, there is a dedicated [tool window](#) for managing your favorites (the [Favorites tool window](#)).

Initially, there is one (empty) favorites list which has the same name as the project.

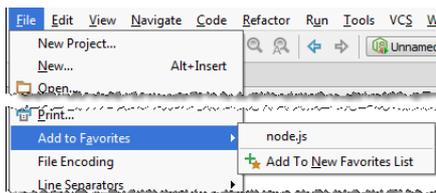
You can create more favorites lists and manage their contents as necessary.

Using the Project tool window to add items to favorites

1. In the [Project](#), select the item or items you want to add to favorites.
2. Do one of the following:
 - On the main menu, point to File | Add To Favorites.
 - On the context menu of the selection, point to Add To Favorites

3. To add the selected item or items to an existing favorites list, select the name of the list.

To create a new favorites list and add the selected item or items to it, select Add To New Favorites List. In the Add New Favorites List dialog, specify the name of the new list and click OK.



Tip Add items to favorites using drag-and-drop: drag the item of interest from the Project tool window, or an external file from Explorer or Finder, and drop it onto the desired favorites list in the Favorites tool window.

Using the editor to add items to favorites

In the editor, you can add to favorites one file, or all the currently opened files.

1. Right-click the editor tab of interest and select one of the following options:
 - If you want to add the current file to favorites, select Add To Favorites. (For the current file, File | Add To Favorites is also available as an alternative.)
 - If you want to add all the files open in the editor to favorites, select Add All To Favorites.
2. To add the item or items to an existing favorites list, select the name of the list.

To create a new favorites list and add the item or items to it, select Add To New Favorites List. In the Add New Favorites List dialog, specify the name of the new list and click OK.

Creating a new favorites list

You have an option of creating a new favorites list when [adding items to favorites](#) or when [moving](#) a favorites list item to a different list.

You can also create a new (empty) favorites list just on its own, as a separate task:

1. If the [Favorites tool window](#) is not currently shown, [open it](#).
2. Do one of the following:
 - Click **+** on the toolbar.
 - Press **Alt+Insert**.
3. In the Add New Favorites List dialog, specify the name of the new list and click OK.

Renaming a favorites list

1. If the [Favorites tool window](#) is not currently shown, [open it](#).
2. Do one of the following:
 - Right-click the list whose name you want to change and select Rename Favorites List.
 - In the toolbar of the Favorites tool window, click .
3. In the New Name for Favorites List dialog box, change the name of the list as required, and click OK.

Moving an item to a different list

1. If the [Favorites tool window](#) is not currently shown, [open it](#).

2. Right-click the list item that you are going to move and select Send To Favorites.

3. To move the selected item to an existing favorites list, select the name of the destination list.

To create a new favorites list and move the item there, select Send To New Favorites List. In the Add New Favorites List dialog, specify the name of the new list and click OK.

Tip To move an item from one favorites list to another, use drag-and-drop.

Removing items from favorites

To remove items from favorites, you can delete the corresponding favorites list items and/or the whole favorites lists.

1. If the [Favorites tool window](#) is not currently shown, [open it](#).

2. Select the item or items that you want to remove from favorites. Note that you can select separate list items and the whole lists at the same time.

3. Do one of the following:

– Click  on the toolbar.

– Select Remove From Favorites in the context menu. (If a single favorites list is currently selected, note that there are also the following options: Delete Favorites List <list_name> and Delete All Favorites Lists Except <list_name>.)

– Press .

PyCharm Editor

In this section:

- PyCharm Editor
 - [Basics](#)
 - [Active editor](#)
 - [Editor's areas](#)
- [Basic Editing Procedures](#)
- [Advanced Editing Procedures](#)
- [Managing Editor Tabs](#)
- [Using TODO](#)

Basics

PyCharm editor is a powerful tool for creating and modifying source code. As any other IDE editor, it supports basic features like [bookmarks](#), [breakpoints](#), [syntax highlighting](#), [code completion](#), [zooming](#), [folding code blocks](#), etc. There are, however, plenty of advanced features like [macros](#), [highlighted TODO items](#), [code analysis](#), [intention actions](#), [intelligent and fast navigation](#), and a lot more.

To configure your editing environment, use the [Editor settings](#) page and its child pages. There is also a [Quick Switch Scheme](#) command that lets you change color schemes, themes, keymaps, etc. with a couple of keystrokes.

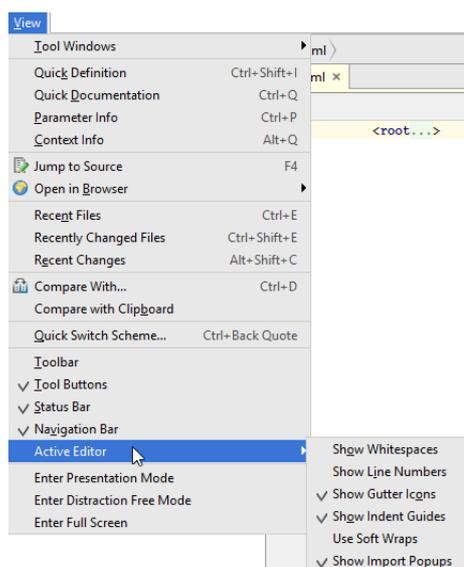
The editor is tab-based. All operations with the editor tabs are available from the context menu of a tab, or from Window | Editor tabs node of the main menu.

Active editor

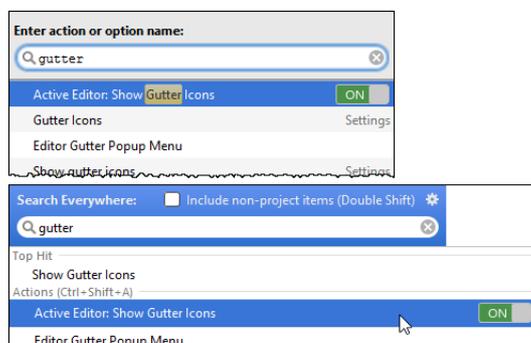
TIP You always return the focus to the active editor from any tool window by pressing the `Escape` key.

When you [open a file for editing](#), it opens in its own tab. The editor you are currently working in, is the active editor.

You can change behavior of the active editor using the commands under View | Active Editor node of the main menu:



Alternatively, you can invoke the commands related to the active editor, from [Find Action](#) or [Search Everywhere](#):



Editor's areas

```

1  _author_ = 'wombat'
2  _project_ = 'MySimplePythonApplication'
3
4  import math
5
6  class Solver:
7      # the first line
8      # the second line
9      def demo(self):
10         while True:
11             a = int(input("a "))
12             b = input(prompt)
13             c = raw_input(prompt)
14             d = b
15             if d >= 0:
16                 # REVIEW[wombat] please make sure that

```

1. Editor area

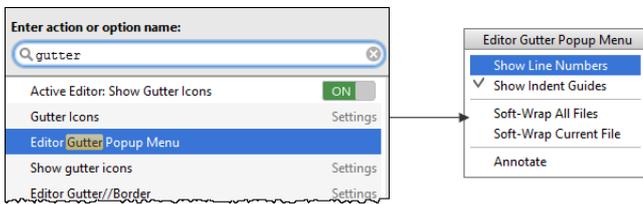
Use this area to type and edit your source code. The editor suggests numerous coding assistance facilities. Refer to the sections under this node, and to [Basic Editing Procedures](#) and [Advanced Editing Procedures](#) for details.

2. Gutter area

The left gutter provides additional information about your code and displays the various icons that identify the code structure, bookmarks, breakpoints, scope indicators, change markers and the code folding lines that let you hide arbitrary code blocks. You can change the behavior of the left gutter.

For example, it's possible to make the left gutter thinner by hiding the gutter icons. This is done either for the [active editor](#), or for all the [newly created editors](#).

To change the behavior of the left gutters, use either the [Appearance](#) page of the editor settings, or the Editor Gutter Popup Menu:



By default, this command is not mapped to any keyboard shortcut. You can create your own shortcut as described in the section [Configuring Keyboard Shortcuts](#).

3. Smart completion pop-up

This is one of the key editing assistance features that suggests method names, functions, tags and other keywords you are typing.

4. Document tabs

Enable quick navigation across the multiple documents you are working on. Clicking a tab brings its contents to front and makes it available for editing in the active editor.

To navigate between the tabs, use the keyboard shortcuts `Alt+Right` or `Alt+Left`.

Clicking a tab while the `Ctrl`/`⌘` key is pressed, allows navigating to any part of the file path, through opening it in an external browser.

Context menu of a tab provides all commands applicable to a file opened in the editor, for example:

- Close one or more tabs.
- Pin active tab.
- Split and unsplit tabs.
- Manage groups of tabs.
- Navigate between tabs.
- Add to Favorites.
- Move to a changelist.
- Run, or debug in the active editor.
- Perform local history and version control commands.
- Perform commands of your own tools.

By default, the tabs appear on top of the editor, but you can change their location as described in the section [Changing Placement of the Editor Tab Headers](#).

5. Validation side bar / marker bar

This is the bar to the right from the editing area, showing the green, red or yellow box on its top depending on whether your code is okay, or contains errors or warnings. This bar also displays active red, yellow, white, green and blue navigation stripes that let you jump exactly to the erroneous code, changed lines, search results, or TODO items.

Basic Editing Procedures

This section describes how to perform the most common editing tasks:

- [Opening and Reopening Files in the Editor](#)
- [Closing Files in the Editor](#)
- [Selecting Text in the Editor](#)
- [Cutting, Copying and Pasting](#)
- [Commenting and Uncommenting Blocks of Code](#)
- [Undoing and Redoing Changes](#)
- [Adding, Deleting and Moving Code Elements](#)
- [Joining Lines and Literals](#)
- [Splitting Lines With String Literals](#)
- [Using Drag-and-Drop in the Editor](#)
- ['Lens' Mode](#)
- [Multicursor](#)
- [Scratches](#)
- [Saving and Reverting Changes](#)
- [Zooming in the Editor](#)
- [Viewing Current Caret Location](#)

On this page:

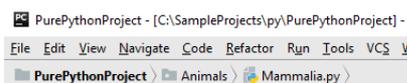
- [Opening files for editing](#)
- [Opening external files](#)
- [Reopening files](#)
- [Opening files in a separate window](#)

Opening files for editing

To open a file for editing

1. Do one of the following:

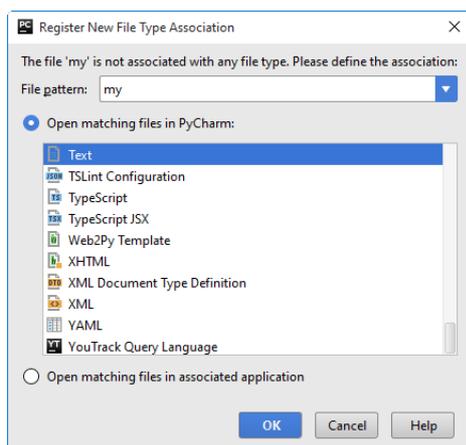
- Double-click the desired file in one of the [Tool Windows](#).
- Select the desired file in one of the [Tool Windows](#) and press `F4`.
- Select the desired file in one of the [Tool Windows](#) and choose Jump to Source on the context menu.
- Use the [Navigate](#) command for a Class, File, or Symbol.
- Click the desired directory in the [Navigation bar](#), and select file from the drop-down list:



2. If the file type is registered, the file opens silently in PyCharm's editor.

If the file type is registered under the category Files opened in associated applications, it will be opened in its associated application, rather than in the PyCharm editor. By default, PyCharm suggests a number of such file types, for example `.doc`, `.chm`, or `.pdf`.

If the file type is unknown, PyCharm suggests you to choose whether you want to register a new file type, or open such file in its associated application. Specify your choice in the [Register New File Type Association](#) dialog box:



You can register the required file types on the [File Types](#) page of the Settings/Preferences dialog.

The maximum size of files parsed by PyCharm is controlled by the `idea.max.intellisense.filesize` setting in `idea.properties` file.

The file `idea.properties`, located in the `bin` directory of the PyCharm installation folder, should not be edited. Instead of editing the original `idea.properties`, create file `idea.properties` in the location specified below, open it for editing and add the required properties.

So, depending on your platform:

- For **Windows**: in `%USERPROFILE%\PyCharmXX`
- For ***NIX**: in `~/PyCharmXX`
- For **macOS**: in `~/Library/Preferences/PyCharmXX`

Note that the larger the file is, the slower its editor works and the higher overall system memory requirements are.

Opening external files

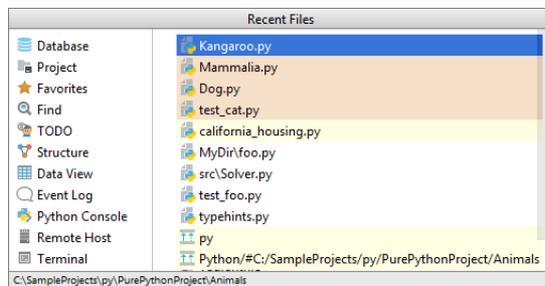
Do one of the following:

- Choose File | Open on the main menu and select the desired file in the dialog box that opens.
- [Drag](#) the required file from the Explorer (Windows), File Browser (Linux), or Finder and [drop](#) it to the editor. The file opens for editing in a new tab.

Reopening files

To reopen a file

- To open a recently opened file, choose View | Recent Files on the main menu or press `Ctrl+E`. Then select the desired file from the Recent Files pop-up window, that opens.



- To open a recently updated file, on the main menu, choose View | Recently Changed Files or press `Ctrl+Shift+E`. Then select the desired file from the Recently Edited Files pop-up window, that opens.

Tip Use Recent files limit text box in the [Editor](#) settings page to define the maximum number of recent files.

Opening files in a separate window

To open a file in a separate PyCharm window

Do one of the following:

- Drag and drop an editor tab outside of the current PyCharm window.
- Press `Shift+F4` for a file selected in the Project tool window.
- `Shift+mouse double click` on a file name in the Project tool window.

PyCharm suggests several ways to close editor tabs.

To close a file in the editor, do one of the following

- On the main menu point to Window | Editor Tabs, choose one of the appropriate closing commands.

Close

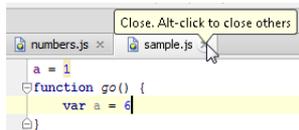
Closes the file in the active tab.

Close All

Closes all editor tabs.

Close Others

Closes all tabs except the current one. The alternative way to close all other tabs lays with clicking the **x** button, while holding the **Alt** key pressed:



Close Unmodified

Closes all files that were not changed. This command is only available, when version control integration is enabled in project.

Close All But Pinned

Closes all files that were not pinned. This command appears, if there are pinned editor tabs.

- Right-click any editor tab, and choose same commands on the context menu.
- Point with your mouse cursor to a tab and click the middle mouse button.
- Point with your mouse cursor to a tab and click **x**.
- Press **Ctrl+F4**.

Tip When you close modified files, PyCharm preserves all changes in the current editing session. After reopening such files, the results of editing are restored.

Selecting Text in the Editor

In this section:

- [Basics](#)
- [Selecting all text in the active editor tab](#)
- [Selecting with navigation keys](#)
- [Alternative ways to select code](#)
- [Extending selection](#)
- [Shrinking selection](#)
- [Multiselection](#)
- [Toggling between line and column selection modes](#)
- [Sticky selection](#)
- [Tips and tricks](#)

Basics

The basic way to select a piece of text is to extend the selection with the mouse cursor. PyCharm, as a keyboard-centric IDE, suggests to use navigation keys to make selections.

You can opt to select pieces of text, or select rectangular fragments in the column mode, extend and shrink the selection, use multiselection, and sticky selection.

Selecting all text in the active editor tab

To select the entire text in the current editor tab, do one of the following:

- On the main menu, choose Edit | Select All.
- Press `Ctrl+A`.

Selecting with navigation keys

To select text from the current caret position to the beginning/end of the current word:

- `Ctrl+Shift+Left`, `Ctrl+Shift+Right`.

To select text from the caret position to the beginning/end of the current line:

- Double-click `Ctrl` and press `Home` / `End`.

To select text from the current caret position to the top/bottom of the screen:

- `Ctrl+Shift+Page Up`, `Ctrl+Shift+Page Down`.

Alternative ways to select code

To make selection of a column of text, do one of the following:

- Keeping the `Alt` key pressed, drag your mouse pointer to select the desired area.
- Keeping the middle mouse button pressed, drag your mouse pointer to select the desired area.
- Press `Shift+Alt` and the middle mouse button. This is specially helpful, if you want to avoid dragging.

Extending selection

To extend selection from the word at caret to the piece of code the caret is contained in, do one of the following:

- On the main menu, choose Edit | Extend Selection
- Press `Ctrl+W` to select the word where the caret is currently located.
- Press `Ctrl+W` successively to extend selection to the next containing node (for example, an expression, a paired tag, an entire conditional block, a method body, a class, a group of vararg arguments, etc.)

While extending selection, keep in mind that:

- Pressing `Ctrl+W` successively in plain text or comments extends the selection first to the current sentence, then to the current paragraph.
- Pressing `Ctrl+W` successively inside a list, dictionary, or a list of arguments or parameters, selects an element of the list, then the right or left comma and a neighbouring space (if any), then the contents of the list without parentheses, and finally the enclosing parentheses.

Shrinking selection

To shrink selection in the reverse order (from the outermost container to the word where the caret currently resides), do one of the following:

- On the main menu, choose Edit | Shrink Selection
- Press `Ctrl+Shift+W`.

Tip The selection extends or shrinks according to capitalization, if the Use "Camel-Humps" words check box is selected on the [Smart Keys](#) page of the editor settings.

If you want to make selection according to capitalization, using double-click, make sure that the check box Honor Camel-Humps words... is selected on the [General](#) page of the editor settings.

Multiselection

PyCharm supports selecting multiple text fragments. So doing, one can select multiple words, lines or rectangles.

To select multiple words, follow these steps

1. Do one of the following:

- Press `(Alt)` and double-click the left mouse button.

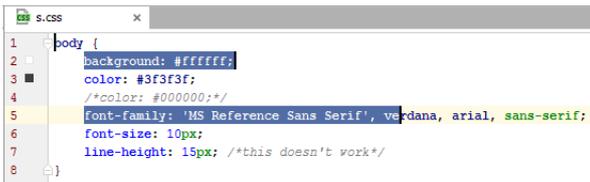
```
<procedure title="To Select multiple words" alternative-title="Using a  
<step...>  
</procedure>  
<anchor name="column selection"/>  
<procedure title="To toggle between the line and the column selection  
<anchor name="altdrag"/>  
<procedure title="To make selection in the Column Selection Mode"
```

- Press `(Alt+J)`, or select some text fragment. Then press `(Alt+J)` again, to find the matching piece of text.

2. After selection is complete, you can start editing all the fragments as if they were one.

To select multiple fragments of text, follow these steps

1. Press `(Alt)`
2. Drag the mouse pointer



```
s.css x  
1 body {  
2   background: #ffffff;  
3   color: #3f3f3f;  
4   /*color: #000000;*/  
5   font-family: 'MS Reference Sans Serif', verdana, arial, sans-serif;  
6   font-size: 10px;  
7   line-height: 15px; /*this doesn't work*/  
8 }
```

To select multiple rectangular fragments of text, follow these steps

1. Press `(Alt)` (Windows or UNIX)/ (macOS)
2. Drag the mouse pointer



```
s.css x  
1 body {  
2   background: #ffffff;  
3   color: #3f3f3f;  
4   /*color: #000000;*/  
5   font-family: 'MS Reference Sans Serif', verdana, arial, sans-serif;  
6   font-size: 10px;  
7   line-height: 15px; /*this doesn't work*/  
8 }
```

Refer to the section [Multicursor](#) for additional information.

toggling between line and column selection modes

To toggle between the line and the column selection modes, do one of the following:

- On the main menu, choose Edit | Column Selection Mode.
- On the context menu of the editor, choose Column Selection Mode.
- Press `(Shift+Alt+Insert)`.

Sticky selection

To toggle sticky selection, press `(Ctrl+Shift+A)`, in the pop-up frame type sticky, and choose Toggle Sticky Selection from the suggestion list:



Tip In the Emacs keymap, use keyboard shortcut `(N/A)`.

Tips and tricks

- When sticky selection is on, you can turn it off by invoking copy or cut, or by toggling it again.
- To create a large rectangular selection, create a normal selection first, with the given starting and ending points, and then press `(Shift+Alt+Insert)` to toggle to the column selection mode.

Cutting, Copying and Pasting

On this page:

- [Basics](#)
- [Copying a selected fragment of text](#)
- [Copying the path to a file](#)
- [Copying the reference to a line or a symbol](#)
- [Cutting a selected fragment of text](#)
- [Pasting the last entry from the clipboard](#)
- [Pasting the last entry from the clipboard as plain text](#)
- [Pasting a specific entry from the clipboard](#)

Basics

PyCharm provides a number of handy Clipboard operations. You can copy, cut, and paste selected text, a path to a file, or a reference to a symbol or a line of code.

Because PyCharm uses the system Clipboard, you can copy and paste between applications. So doing, when pasting Clipboard entries, PyCharm removes any formatting from the text and any special symbols from the `String` values.

The Paste command smartly understands what is being inserted. If you paste a reference to a symbol, it is analyzed for possible imports, references, etc. So doing, PyCharm provides the necessary brackets and places the caret at the appropriate insertion point. The Paste Simple command helps paste any Clipboard entry as a plain text, without any analysis.

PyCharm enables Clipboard stacking, which means that you can store multiple Clipboard entries and access them with a single shortcut. The number of entries that can be kept in the Clipboard stack is customizable on the [Editor](#) page of the Settings/Preferences dialog.

Copying a selected fragment of text

Do one of the following:

- On the main menu, choose Edit | Copy.
- Press `Ctrl+C`.
- Click the Copy button  on the toolbar.

Note that the `Ctrl+D` keyboard shortcut clones a line at the caret or a selected arbitrary fragment of text.

Copying the path to a file

Do one of the following:

- Open the desired file in the editor, then choose Edit | Copy Path on the main menu or press `Ctrl+Shift+C`.
- Select the desired file in the [Project](#) tool window and choose Copy Path on the context menu of the selection.

Copying the reference to a line or a symbol

1. Open the desired file in the editor.
2. Place the caret at a certain line of code.
3. Do one of the following:
 - On the main menu, choose Edit | Copy Reference.
 - On the context menu of the line at caret, choose Copy Reference.
 - Press `Ctrl+Shift+Alt+C`.

PyCharm creates a string in the format that depends on a symbol at caret. For example:

```
Solver.Solver#discr
```

 for a Python method

```
Solver\Solver.py:14
```

 for a Python file

Cutting a selected fragment of text

1. [Select](#) the desired fragment in the editor.
2. Do one of the following:
 - On the main menu, choose Edit | Cut.
 - Press `Ctrl+X`.
 - Click the Cut button  on the toolbar.

Pasting the last entry from the clipboard

1. Place the caret in the location where you want to paste content.
2. Do one of the following:
 - On the main menu, choose Edit | Paste.
 - Press `Ctrl+V`.
 - Click the Paste button  on the toolbar.

If you perform paste in a Python file, the further behavior depends on the settings in the [Auto Import](#) page of the Editor options. If the Ask option has been

selected, select the necessary imports from the list of missing imports. In all other cases, the last clipboard entry is pasted silently.

Pasting the last entry from the clipboard as plain text

Do one of the following:

- On the main menu, choose Edit | Paste Simple.
- Press `Ctrl+Shift+Alt+V`.

Pasting a specific entry from the clipboard

1. On the main menu, choose Edit | Paste from History or press `Ctrl+Shift+V`.
2. In the Choose Content to Paste dialog box select the desired entry from the list of recent Clipboard entries, and click OK.

The depth of the Clipboard stack is configured in the Limits section on the [Editor](#) page of the [Settings](#) dialog box. When the specified number is exceeded, the oldest entry is removed from the list.

Commenting and Uncommenting Blocks of Code

On this page:

- [Basics](#)
- [Commenting and uncommenting lines of code](#)
- [Commenting and uncommenting blocks of code](#)

Basics

You can comment or uncomment the current line or selected block of source code.

Commenting feature extends to all supported file types. For the custom file types, you can define line and block comments characters, as described in the section [Creating and Registering File Types](#).

Commenting and uncommenting lines of code

Do one of the following:

- On the main menu, choose Code | Comment with Line Comment.

- Press `Ctrl+Slash`.

```
def demo(self, a, b, c):
    d = math.sqrt(b ** 2 - 4 * a * c)
    root1 = (-b + d) / (2 * a)
    root2 = (-b - d) / (2 * a)
    # print(root1, root2)
    # print("SE")
```

Commenting and uncommenting blocks of code

Warning! Block comments do not apply to Python scripts!

To add or remove a block comment, do one of the following:

- On the main menu, choose Code | Comment with Block Comment.

- Press `Ctrl+Shift+Slash`.

Undoing and Redoing Changes

On this page:

- [Basics](#)
- [How it works?](#)
- [Undoing and redoing changes](#)

Basics

The Undo command discards the last changes to the file in the editor. The Redo command discards the results of the last Undo command.

You can undo or redo your changes as many times as required. However, when you exit PyCharm, the undo history is lost.

PyCharm smartly defines the logical steps that can be undone and redone. The following events signal about the end of a logical step:

- Pressing `Enter`.
- Repositioning the mouse cursor.
- Using navigation keyboard shortcuts.
- Cutting or pasting.
- Pressing `Tab`.

PyCharm expands the undo and redo mechanism to complex operations, such as reformatting or refactoring source code, creating or deleting files. When you undo or redo a complex operation, PyCharm requests for your confirmation.

How it works?

PyCharm moves the caret before each step of undo/redo, and then performs the **Undo/Redo** actions.

Undoing and redoing changes

To undo an action, do one of the following

- On the main menu, choose Edit | Undo.
- Press `Ctrl+Z`.

To redo an action, do one of the following

- On the main menu, choose Edit | Redo.
- Press `Ctrl+Shift+Z`.

On this page:

- [Adding lines](#)
- [Duplicating lines](#)
- [Deleting lines](#)
- [Moving lines](#)
- [Moving statements](#)
- [Moving code element left or right](#)

To add a line

- Press `Shift+Enter` to add a new line after the one where the caret is currently located and move the caret to the beginning of this new line. For instance, you have typed some text:

```
{
function Animal (type) {
    this.getInfo = getAnimalInfo;
}
```

- Press `Shift+Enter` to start the next line immediately:

```
{
function Animal (type) {
    this.getInfo = getAnimalInfo;
}
```

- To add a line before the current one, press `Ctrl+Alt+Enter`.

Tip Make sure that keyboard shortcuts are not in conflict. You can do that in the [Keymap](#) page of the [Settings/Preferences dialog](#).

To duplicate a line or fragment

1. Place the caret at the line to be duplicated, or [select](#) the desired fragment of text.
2. Press `Ctrl+D`.

To remove a line

- Press `Ctrl+Y` to delete the line at caret.

To move a line

1. Place the caret at the line to be moved.
2. Do one of the following:
 - On the main menu, choose Code | Move Line Up or Code | Move Line Down.
 - Press `Shift+Alt+Up` or `Shift+Alt+Down`.

PyCharm moves the selected line one line up or down, performing the syntax check. For example:

```
class ReverseOrderDescendant(ReverseOrderBaseClass):
    def reverse(self, data):
        for index in range(len(data)-1, -1, -1):
            yield data[index]
```

After moving line at caret:

```
class ReverseOrderDescendant(ReverseOrderBaseClass):
    def reverse(self, data):
        yield data[index]...
        for index in range(len(data)-1, -1, -1):...
```

To move a statement up or down

1. [Select](#) a statement to be moved, or just place the caret at the first or last lines of a multi-line statement. Note that if moving a statement is not allowed in the current context, the commands will be disabled.
2. Do one of the following:
 - On the main menu, choose Code | Move Statement Up/Move Statement Down.
 - Press `Ctrl+Shift+Up` or `Ctrl+Shift+Down`.

If you apply the same commands to a line at caret, or a to a selection, it will be moved one line up or down.

PyCharm moves the selected statement above the previous one, or directly underneath the next one, performing the syntax check. For example, place the caret at the method declaration:

```

class Solver:
    def sol(self):
        pass
    def demo(self, a, b, c):
        d = b**2 - 4*a*c
        root1 = (-b**2 + math.sqrt(d)) / (2*a)
        root2 = (-b**2 - math.sqrt(d)) / (2*a)
        return root1, root2
    def demo_cycle(self, a, b, c):...

```

After moving the statement:

```

class Solver:
    def sol(self):
        pass
    def demo_cycle(self, a, b, c):...
    def demo(self, a, b, c):
        d = b**2 - 4*a*c
        root1 = (-b**2 + math.sqrt(d)) / (2*a)
        root2 = (-b**2 - math.sqrt(d)) / (2*a)
        return root1, root2

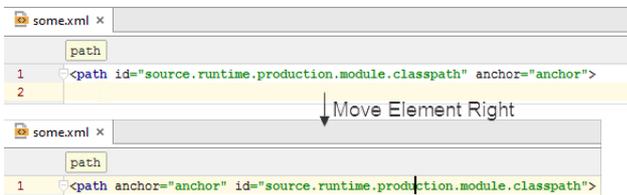
```

To move code element to the left or to the right

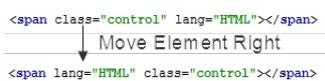
1. Place the caret at the desired code element, or select the elements to be moved.
2. Do one of the following:
 - On the main menu, choose the commands Code | Move Element Left or Code | Move Element Right
 - Press `Ctrl+Shift+Alt+Left` or `Ctrl+Shift+Alt+Right`

Examples of code elements for which this functionality is currently implemented:

- XML: tag attributes.



- HTML: tag attributes.



Joining Lines and Literals

PyCharm makes it possible to concatenate two unselected or several selected lines into one, removing the extra spaces, and providing the proper syntax. This operation smartly analyzes the lines being joined and treats them accordingly. For example, you can join lines of code, lines of comments, field declaration and initialization.

On this page:

- [Joining lines](#)
- [Joining string literals](#)
- [Examples](#)

Joining lines

1. Place the caret on the line, to which the other lines should be added.

```
5 |  
6 | a = 3  
7 | b = 25  
  | c = 46
```

2. Sequentially press `Ctrl+Shift+J` keyboard shortcut, until all fragments are joined in a single line.

```
5 | a = 3; b = 25  
6 | c = 46  
5 | a = 3; b = 25; c = 46
```

Tip You can select the lines and press `Ctrl+Shift+J` to obtain the same result.

Joining string literals

1. Select the lines with string literals that should be joined.

2. Press `Ctrl+Shift+J` keyboard shortcut. All redundant characters (spaces, quotes, and plus signs) are gone.

Examples

Joining a list of parameters:

```
20 | def joining_lines(a,  
21 | b,  
22 | c,  
23 | d):  
  |  
  | → 20 | def joining_lines(a, b, c, d):
```

Joining commented lines:

```
15 | #def bar():  
16 | # q = a  
17 | # tasks = {'completed': u'0',  
18 | # tasks[]  
  |  
  | → 15 | #def bar():  
  | 16 | # q = a tasks = {'completed': u'0',  
  | 17 | # tasks[]
```

Splitting Lines With String Literals

This feature is designed to correctly split string literals, providing the correct syntax.

To split a string literal into two parts:

1. Set the caret in the string literal to be split.

```
19 | print ("Hello,|World!")
```

2. Press .

```
19 | print ("Hello, "  
20 | |World!")
```

PyCharm allows copying and moving code fragments within the active editor tab, by means of drag-and-drop.

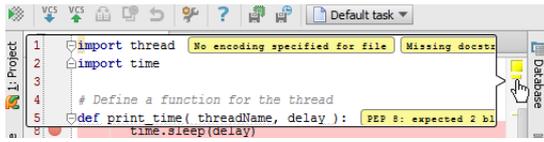
To move or copy code fragment

1. Make sure that using drag-and-drop is enabled in the [Editor](#) page of the IDE Settings.
2. [Select](#) the desired fragment in the editor.
3. Move or copy selection:
 - Move: Drag the selected fragment to the target location.
 - Copy: Keeping the `Ctrl` key pressed, drag selection to the target location.

'Lens' Mode

Hover the mouse pointer over a warning, error stripe or just some section on the scroll bar outside of the scroll box. PyCharm shows the source code fragment annotated with the warning/error messages.

This helps viewing the context a marker applies to, and the source code outside of the editor visible area.



Must-read This topic describes the usage of lens.

This behavior is enabled by default.

To toggle the lens mode

1. Do one of the following:
 - Open the [Appearance](#) page of the Settings dialog.
 - Right-click the code analysis marker on top of the current editor.
2. Select or clear the check box Show code lens on scrollbar hover.

Multicursor

On this page:

- [Basics](#)
- [Adding, deleting, and cloning carets](#)
- [Copying and pasting](#)

Basics

PyCharm supports multiple carets. The majority of the editor actions, such as keyboard navigation, text insertion and deletion, etc., apply to each caret. Live templates and autocompletion are supported as well.

It is possible to add or delete carets; at least one caret always exists in an editor tab.

The most recently added caret is considered primary. Highlighting of an editor line at caret applies to the primary caret only.

Placement and behavior of the carets depend on:

- Enabled or disabled [column selection mode](#).
- Enabled or disabled [placement of caret after the end of line](#).

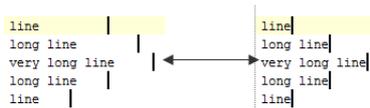
Refer to the section [Selecting Text in the Editor](#) for additional information.

Adding, deleting, and cloning carets

To add carets, do one of the following

- Press **Alt** (Windows or UNIX)/**Cmd** (macOS) and click the left mouse button at the location of the caret.
- Press **Ctrl** (Windows or UNIX)/**Alt** (macOS) twice, and then without releasing it, press up or down arrow keys.

The new carets are added to the specified locations, according to setting of the Allow placement of caret after end of line check box:



To delete carets, do one of the following

- Press **Esc** to delete all the existing carets, except the primary one.
- Press **Shift+Alt** and click the left mouse button on a caret to be deleted.

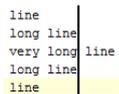
To clone an existing caret upward or downward, do one of the following:

- Press **Ctrl+Shift+A**, type **Clone caret**, and choose the desired action from the suggestion list:



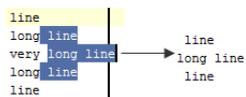
Note that by default these actions are not bound to the keyboard shortcuts. You can do it yourself, as described in the section [Configuring keyboard shortcuts](#).

The primary caret is propagated upwards or downwards:



Copying and pasting

When a text with multiple cursors is copied (**Ctrl+C**) or cut (**Ctrl+X**), selections for each caret are placed to the clipboard. On paste (**Ctrl+V**), text from the clipboard is split into lines.



Scratches

On this page:

- [Basics](#)
- [Creating scratch files](#)
- [Creating scratch buffers](#)
- [Observing the available scratches](#)
- [Closing scratches](#)
- [Deleting scratches](#)
- [Changing the language of a scratch](#)
- [Renaming, copying and moving scratches](#)
- [Important notes about scratches](#)

Basics

PyCharm provides a temporary editor. You can create a text or a piece of code for search or exploration purposes. PyCharm suggests two types of temporary files:

Scratch files

The scratch files are fully functional, runnable, debuggable, etc. They require a language to be specified and have an extension. The scratch files are created via `Ctrl+Shift+Alt+Insert`.

Scratch buffers

The scratch buffers are only intended for pure editing, and as such they do not require specifying a language and do not have an extension. The scratch buffers belong to `.txt` type by default.

This action has no dedicated shortcut, but you can configure one as described in the section [Configuring Keyboard Shortcuts](#).

Buffer files are reused after creating 5 files. So doing, after reuse, the content and language are reset.

Creating scratch files

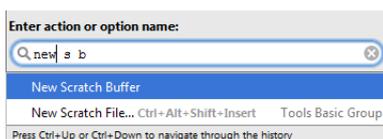
To create a scratch file

1. Do one of the following:
 - On the main menu, choose File | New Scratch File .
 - Press `Ctrl+Shift+Alt+Insert`
2. Select the language of the future scratch from the list that PyCharm suggests. PyCharm creates a temporary editor tab with the name `scratch.<extension>`. In the future, the default names will be `scratch_<number>.<extension>`.
3. Type the desired code.

Creating scratch buffers

To create a scratch buffer, follow these steps:

1. Press `Ctrl+Shift+A` or [search everywhere](#).
2. Start typing the command name `New Scratch Buffer` :



PyCharm creates a temporary editor tab with the name `buffer1`. In the future, the default names will be `buffer<number>`.

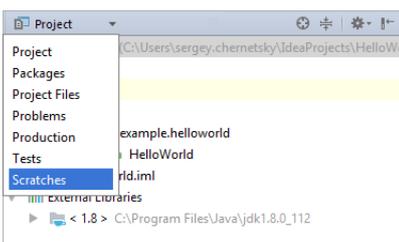
3. Type the desired code.

Note that though this action has no keyboard shortcut, you can still configure one as described in the section [Configuring Keyboard Shortcuts](#). You can also switch from scratch files to scratch buffers by reassigning the shortcut to avoid garbage buildup.

Observing the available scratches

To observe the available scratch files and buffers, do one of the following:

- Choose Scratches view in the [Project tool window](#):



- Press `Alt+F1` and choose Scratches ([Navigating Between IDE Components](#)).

Closing scratches

To close a scratch file or buffer, just click `x` on the editor tab. Refer to the section [Closing Files in the Editor](#) for details.

Deleting scratches

To delete a scratch file or buffer, follow these steps:

1. Switch to the Scratches view of the [Project tool window](#).
2. Under the `Scratches` pseudo-folder, right-click the scratch to be deleted, and choose Delete on the context menu.
3. Confirm deletion.

Changing the language of a scratch

If you want to change the scratch's language when a scratch is already created, you can do so with the aid of the editor's context menu. This is how it's done:

1. Switch to the Scratches view of the [Project tool window](#), and open for editing the scratch file or buffer you want to change language for.
2. Right-click the editor background, and choose Change Language (<current language>) on the context menu.
3. Select the desired language.

Note the following:

- Four latest items appear on top of the list before a separator.
- You can narrow down the list by typing the language name.
- You can assign a shortcut to this action as described in the section [Configuring keyboard shortcuts](#).
- Change Language action keeps extension in sync, if it exists.

Renaming, copying and moving scratches

PyCharm makes it possible to perform [rename refactoring](#) of the scratches. To rename a scratch, follow these steps:

1. In the [Project tool window](#), switch to the Scratches view and select the scratch to be renamed.
2. Press `Shift+F6`.

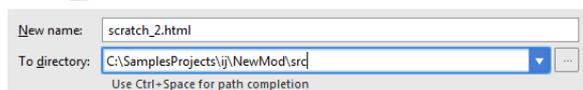
You can also perform renaming in the other ways:

- In [NavBar](#): Jump to NavBar (`Alt+Home`) -> Rename (`Shift+F6`).
- In the [Project tool window | Scratches view](#): Select In | Project | Scratches (`Alt+F1`) -> Rename (`Shift+F6`).
- Right from the editor: Refactor | Rename File.

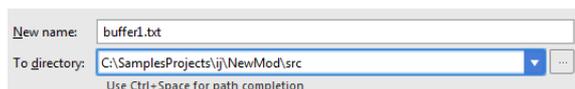
Copy and Move file actions are available the same way. Refer to the sections [Copy](#) and [Move Refactorings](#) for details.

Note that when copying a scratch, PyCharm includes the respective extension that corresponds to the file type. This is how it's done:

1. In the [Project tool window](#), switch to the Scratches view and select the scratch to be copied.
2. Press `F5`. PyCharm shows the following dialog box:



This dialog box shows the scratch name with the corresponding extension. Note that when you copy a [scratch buffer](#), the extension is `.txt`:



Important notes about scratches

Note the following:

- The scratch code in scripting languages is executable: you can [run](#) and [debug](#) it.
- [Local history](#) for scratches is supported.
- It is possible to perform [clipboard operations](#) with scratches.
- The scratches are stored, depending on your operating system,
 - Under PyCharm home, in the directory `config/scratches` (on Windows/*NIX)
 - `~ Library -> Preferences -> <PyCharm>XX -> scratches` (on macOS)
- You can [undo or redo changes](#) in scratches.

Saving and Reverting Changes

When working with PyCharm, you don't need to worry about saving changed files: all changes are auto saved.

Unwanted changes can be undone at any stage of your development workflow. Any file or directory can be reverted to any of the previous states.

- [When does PyCharm auto save changed files?](#)
- [Tuning the autosave behavior](#)
- [Using the Save All command](#)
- [Marking files with unsaved changes in the editor](#)
- [Saving a file under a different name](#)
- [Reverting changes](#)

When does PyCharm auto save changed files?

Autosave is initiated by:

- Starting a run/debug configuration
- Performing a version control operation such as pull, commit, push, etc.
- Closing a file in the editor
- Closing a project
- Quitting the IDE

In fact, there is a lot more autosave triggers, and only the most important ones are mentioned above.

Tuning the autosave behavior

The following options are available for tuning the autosave behavior (File | Settings | Appearance and Behavior | System Settings):

- Save files on frame deactivation (i.e. on switching from PyCharm to a different application)
- Save files automatically if application is idle for N seconds

Note that those are optional autosave triggers, and you cannot turn off autosave completely.

Using the Save All command

If necessary, you can initiate saving all changed files yourself. There is the Save All command for that:

- File | Save All
- `Ctrl+S`

Marking files with unsaved changes in the editor

Changed but yet unsaved files can be marked. For this purpose, there is the Mark modified tabs with asterisk option (File | Settings | Editor | General | Editor Tabs).

When this option is on, the files with unsaved changes have an asterisk * on their editor tabs.

Saving a file under a different name

There is no File | Save As command in PyCharm. To save a file under a different name or in a different directory, use Refactor | Copy or `F5`.

Reverting changes

You can undo changes by using Edit | Undo or `Ctrl+Z`. To revert files to their previous states, use [Local History](#) and corresponding version control functionality.

Zooming in the Editor

PyCharm makes it possible to change font size (zoom) in the active editor, and reset font size to the default value. These operations apply to the active editor only. In the other editor tabs, font size is not affected.

On this page:

- [Enabling zooming](#)
- [Changing font size with the mouse wheel](#)
- [Changing font size with the keyboard](#)
- [Restoring default font size](#)

To enable changing font size in the editor

1. Open the Settings/Preferences dialog, expand the Editor node, and click [General](#).
2. Make sure that the setting Change font size (Zoom) with Ctrl+MouseWheel is enabled.

To change font size using the mouse wheel

1. Place the caret in the editor.
2. While keeping the `Ctrl`/`⌘` key pressed, rotate the mouse wheel. As you rotate the mouse wheel forward, font size grows larger; as you rotate the mouse wheel backwards, font size decreases.
The macOS users can use trackpad "Pinch-to-Zoom" gesture to change size of the font and the whole editing area.

To change font size using the keyboard

1. Press `Ctrl+Shift+A`.
2. In the popup frame, type Increase font size or Decrease font size, and then click `Enter`.
Font grows larger or smaller.

Tip There are no default keyboard shortcuts associated with the actions Increase font size and Decrease font size action. However, you can create your own shortcuts as described in the section [Configuring Keyboard Shortcuts](#).

To reset font size

1. Press `Ctrl+Shift+A`.
2. In the popup frame, type Reset font size, and click `Enter`.
The default font size is restored.

Tip There is no default keyboard shortcut associated with Reset font size action. However, you can create your own shortcut as described in the section [Configuring Keyboard Shortcuts](#).

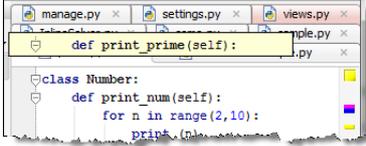
Viewing Current Caret Location

If in course of editing, searching, or navigating through a file, the cursor position runs out of the visible editor area, above the upper editor edge, you don't need to scroll through the file to obtain instant information about the current caret location.

To view the current caret position, do one of the following

- On the main menu, choose View | Context Info
- Press `Alt+Q`.

The pop-up frame appears on top of the editor, showing the name of the class or method where the caret currently resides:



Advanced Editing Procedures

This part describes more sophisticated editing techniques provided by PyCharm:

- [Code Folding](#)
- [Improving Visibility of the Source Code](#)
- [Using Macros in the Editor](#)
- [Spellchecking](#)
- [Editing CSV and Similar Files in Table Format](#)
- [Adding Editors to Favorites](#)

Code Folding

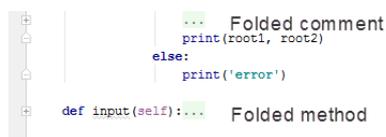
On this page:

- [Basics](#)
- [Code folding means](#)
- [Folding predefined and custom regions](#)
- [Commands of the Folding menu and associated shortcuts](#)
- [Specifying code folding preferences](#)
- [Viewing folded code fragments](#)
 - [Viewing the beginning of a folding region](#)
- [Using code folding comments](#)
 - [Supported folding comments](#)
 - [Surrounding a fragment with folding comments](#)
 - [Navigating to folding regions](#)

Basics

You can collapse (fold) code fragments reducing them to a single visible line. In this way, you can hide the details that, at the moment, seem unimportant. If and when necessary, the folded code fragments can be expanded (unfolded).

Folded code fragments, normally, are shown as shaded ellipses (...).



Code folding means

You can collapse and expand code fragments by using:

- Code folding toggles (⌵, ⌴ or ⌶). These toggles are shown in the editor to the left of the corresponding folding regions. If a region is unfolded, ⌵ indicates the beginning of the region while ⌴ is located at its end. For folded regions, the toggle is shown as ⌶.
If you hold the **Alt** key and click ⌵, ⌴ or ⌶, the region is collapsed or expanded recursively, i.e. along with all its subordinate regions.
- Commands of the Folding menu and associated keyboard shortcuts. The Folding menu can be accessed from the main menu bar (Code | Folding), or as a context menu in the editor. The shortcuts are shown right in the menu. See [Commands of the Folding menu and associated shortcuts](#).
- Folded fragments themselves: click ... to expand the corresponding fragment. See also, [Viewing folded code fragments](#).

Folding predefined and custom regions

You can fold and unfold:

- Lists and dictionaries.
- Code blocks, i.e. code under the keywords `class`, `def`, `while`, `if`, etc.
- Consecutive commented lines.
- Predefined regions that correspond to such elements as import declarations, method bodies, documentation comments, etc. The predefined regions, roughly, correspond to the ones listed under Collapse by default on the Editor | General | Code Folding page in the Settings/Preferences dialog.
For the predefined regions, the folding toggles are available right away, without the need to perform any additional actions.
- Any selected code fragment. A custom folding region for a selection is created and removed by means of the Fold Selection/ Remove Region command (**Ctrl+Period**).
- Regions surrounded by corresponding commented folding markers (e.g. `//<editor-fold desc="Description">...</editor-fold>`). See [Using code folding comments](#).

Note Code folding works for the keywords `if` / `while` / `else` / `for` / `try` / `except` / `finally` / `with` / `Long string literals` / `Long collection literals` / `Sequential comments` in case of at least two statements.

Commands of the Folding menu and associated shortcuts

The Folding menu can be accessed from the main menu bar (Code | Folding), or as a context menu in the editor.

CommandShortcutDescription

Command	Shortcut	Description
Expand	Ctrl+NumPad Plus	Expand the current collapsed fragment
Collapse	Ctrl+NumPad -	Collapse the current folding region
Expand Recursively	Ctrl+Alt+NumPad Plus	Expand the current folded fragment and all the subordinate collapsed folding regions within that fragment
Collapse Recursively	Ctrl+Alt+NumPad -	Collapse the current folding region and all the subordinate folding regions within it
Expand All	Ctrl+Shift+NumPad Plus	Expand all collapsed fragments within the selection, or, if nothing is selected, expand all the collapsed fragments in the current file
Collapse All	Ctrl+Shift+NumPad -	Collapse all folding regions within the selection, or, if nothing is selected, collapse all the folding regions in the current file
Expand to level 1, 2, 3, 4 or 5	Ctrl+NumPad *, 1	Expand the current fragment and all the nested fragments up to the specified level

Ctrl+NumPad *, 2

Ctrl+NumPad *, 3

Ctrl+NumPad *, 4

Ctrl+NumPad *, 5

Expand all to level | 1, 2, 3, 4 or 5

Ctrl+Shift+NumPad *, 1

Expand all the collapsed fragments in the file up to the specified nesting level

Ctrl+Shift+NumPad *, 2

Ctrl+Shift+NumPad *, 3

Ctrl+Shift+NumPad *, 4

Ctrl+Shift+NumPad *, 5

Expand doc comments

Expand all documentation comments in the current file

Collapse doc comments

Collapse all documentation comments in the current file

Fold Selection / Remove region

Ctrl+Period

Collapse the selected fragment and create a custom folding region for it to make it "foldable" / Expand the current fragment and remove the corresponding custom folding region to make the fragment "unfoldable"

Specifying code folding preferences

You can specify:

- Whether the code folding toggles should be shown.
- Which folding regions should be collapsed by default.

The corresponding settings are in the Settings dialog ([Ctrl+Alt+S](#)) on the Editor | General | Code Folding page.

For more information, see [Code Folding page](#).

Viewing folded code fragments

To see the contents of a folded fragment, point to the ellipsis  that indicates that fragment.

```
def view2(request):
    food = {'beautiful soup', 'steak'}
    c = django.template.context.Context({
        'fav_food': food
    })
    return HttpResponse(t.render(c))
```

Viewing the beginning of a folding region

To see the beginning of a folding region - if it's not currently visible - point to the folding toggle at the end of that region.

```
6 def demo(self, a, b, c):
7     d = b ** 2 - 4 * a * c
8     if d >= 0:
9         return root1, root2
13    else:
14        raise Exception
15
16 Solver().demo(2, 3, 1)
```

Using code folding comments

- [Supported folding comments](#)
- [Surrounding a fragment with folding comments](#)
- [Navigating to folding regions](#)

Supported folding comments

You can create custom folding regions by surrounding code fragments with the commented lines. So doing, the comments can be either NetBeans style, or Visual Studio style.

NetBeans style

For non-Python files:

```
//<editor-fold desc="Description">
...
//</editor-fold>
```

For Python files:

```
#<editor-fold desc="Description">
...
#</editor-fold>
```

Visual Studio style

For non-Python files:

```
//region Description
...
//endregion
```

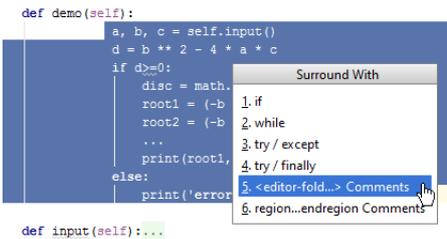
For Python files:

```
#region Description
...
#endregion
```

Once you have chosen the style for a file, don't use the other style in that file.

Surrounding a fragment with folding comments

1. Select the code fragment of interest.
2. Select Code | Surround With or press `Ctrl+Alt+T`.
3. Select the folding comments to be used.



4. Specify the fragment description.

```
def demo(self):
    # <editor-fold desc="MyRegion">
    a, b, c = self.input()
    d = b ** 2 - 4 * a * c
    if d >= 0:
        disc = math.sqrt(d)
        root1 = (-b + disc) / (2 * a)
        root2 = (-b - disc) / (2 * a)
        ...
        print(root1, root2)
    else:
        print('error')
    # </editor-fold>
```

Now if you collapse the fragment, the description you have specified is shown in place of the code.

Navigating to folding regions

You can navigate to custom folding regions that were formed by surrounding code fragments with the corresponding commented folding markers:

1. Select Navigate | Custom Folding or press `Ctrl+Alt+Period`.
2. Select the target folding region. (The regions in the list are identified by their descriptions.)

Improving Visibility of the Source Code

This section describes how to make your source code more clear and readable:

- [Reformatting Source Code](#)
- [Changing Indentation](#)
- [Toggling Case](#)
- [Highlighting Braces](#)

Reformatting Source Code

On this page:

- [Basics](#)
- [Reformatting the code of a directory](#)
- [Reformatting the code of the current file](#)
- [Skipping a region when reformatting source code](#)
- [Example of using formatting markers](#)

Basics

PyCharm lets you reformat source code to meet the requirements of your code style. PyCharm will lay out spacing, indents, keywords etc. Reformatting can apply to the selected text, entire file, or entire project.

It is also possible to apply reformatting to the parts of the source code only, using the [formatting markers](#).

Reformatting the code of a directory

To reformat code for a directory, follow these steps:

1. In the Project tool window, select the directory you want to apply your reformatting to.
2. Choose Code | Reformat Code on the main menu or press `Ctrl+Alt+L`.
Alternatively, in the [Project tool window](#), right-click the directory and from the context menu, select Reformat Code.
3. In the [Reformat Code](#) dialog box, specify the necessary options and filters for your reformatting and click Run.

Reformatting the code of the current file

To reformat code for the current file, follow these steps:

1. In the editor of the currently opened file, press `Ctrl+Shift+Alt+L`.
Note that if you select Code | Reformat Code from the main menu or press `Ctrl+Alt+L`, PyCharm will try to reformat the source code automatically without opening the [Reformat File](#) dialog.
2. In the [Reformat File](#) dialog, specify options for the reformatting and click Run.

Skipping a region when reformatting source code

To enable formatter markers, make sure to select the check box Enable formatter markers in comments in the [Code Style](#) page of the Settings/Preferences dialog, and type the markers in the Formatter off/on fields.

To skip a certain region on reformatting, follow these steps:

1. At the beginning of the region, create a line comment (`Ctrl+Slash`), and then manually type the marker specified in the Formatter off field of [Code Style](#) page.
2. At the end of the region, create a line comment (`Ctrl+Slash`), and then manually type the marker specified in the Formatter on field of [Code Style](#) page.
3. Perform code reformatting, as described above.

Alternatively, create a [live template](#) to surround a block of code with formatter off/on markers, see [Creating and Editing Live Templates](#).

Example of using formatting markers

The original source code **The code after reformatting**

```
@@formatter:off
- "scripts": {
-   "post-install-cmd": [
-     "php artisan optimize"
-   ],
-   "post-update-cmd": [
-     "php artisan clear-compiled",
-     "php artisan optimize"
-   ],
-   "post-create-project-cmd": [
-     "php artisan key:generate"
-   ]
- },
@@formatter:on
```

When the formatting markers are disabled, the original formatting is broken:

```
@@formatter:off
- "scripts": {
-   "post-install-cmd": [
-     "php artisan optimize"
-   ],
-   "post-update-cmd": [
-     "php artisan clear-compiled",
-     "php artisan optimize"
-   ],
-   "post-create-project-cmd": [
-     "php artisan key:generate"
-   ]
- },
@@formatter:on
```

When the formatting markers are enabled, the original formatting is preserved:

```
@@formatter:off
- "scripts": {
-   "post-install-cmd": [
-     "php artisan optimize"
-   ],
-   "post-update-cmd": [
-     "php artisan clear-compiled",
-     "php artisan optimize"
-   ],
-   "post-create-project-cmd": [
-     "php artisan key:generate"
-   ]
- },
@@formatter:on
```

Changing Indentation

PyCharm makes it possible to:

- [Indent or unindent](#) text. This action applies to a selection, or to a line at caret.
- [Fix wrong indentation](#) according to the code style.
- Choose [tabs or spaces](#) for indentation. This action applies to a selection, or the whole current file in the active editor.

To change indentation of a text fragment, do one of the following

- On the main menu, choose Edit | Indent Selection / Edit | Unindent Selection.
- Press `Tab` / `Shift+Tab`.

To fix indentation

Sometimes it is necessary to change indentation of a line at caret.

1. Place the caret at a line with wrong indentation.
2. Press `Ctrl+Alt+I`.

To toggle between tabs and spaces

- On the main menu, choose Edit | Convert Indents , and then choose To Spaces or To Tabs respectively.

Toggle Case

On this page:

- [Toggle Case between upper and lower cases](#)
- [Tips and tricks](#)

Toggle Case between upper and lower cases

To toggle between upper case and lower case

1. Select text fragment, or just place the caret at the line you want to change case in.
2. On the main menu, choose Edit | Toggle Case, or press `Ctrl+Shift+U`.

Tips and tricks

Did you know that

- Applying Toggle Case to CamelCase name format converts the name format to lower case?
- The lowercase names are converted to the uppercase format?
- Applying Toggle Case to the uppercase name format converts the name format to lower case?

Highlighting Braces

This editor feature significantly improves readability of the code, and simplifies search for unclosed blocks or tags.

To highlight block borders

- Place the caret immediately after the block closing brace/bracket or before block opening brace/bracket.

If the editor can find the block border, its braces, brackets or tags are highlighted with blue and a blue outline appears in the gutter area.

If the opening brace, bracket or tag is currently out of sight, you don't need to scroll to the beginning of the block. The editor shows a pop-up window on top that displays the beginning of the block.

```
if (argv[i] == '--port'):  
    del argv[i]  
    retVal['port'] = int(argv[i])  
    del argv[i]  
elif (argv[i] == '--vm_type'):
```

If the editor cannot find the pair brace, bracket or tag, the unmatched one is highlighted with pink when the caret is placed next to it, and is underlined with a red curly line.

```
43 }  
44 )  
45
```

Using Macros in the Editor

Macros provide a convenient way to automate repetitive procedures you do frequently while writing code. You can record, edit and playback macros, assign them a shortcut, and share them. Generally speaking, macros are designed for rather simple operations, and as such have the following limitations:

- Macros can be used for editor-related actions within a file.
- You cannot record such actions as button clicks, navigating to pop-up dialog boxes, and accessing tool windows or menus.

If a macro is intended for temporary use only, it is unnamed; permanent macros have unique names.

This section describes how to:

- [Record Macros](#)
- [Bind Macros With the Keyboard Shortcuts](#)
- [Play Back Macros](#)
- [Edit Macros](#)

To bind a macro with a keyboard shortcut

1. [Open the Settings dialog](#) and click [Keymap](#).
2. [Create a new keymap](#) or select an editable keymap from the list of keymaps.
3. Expand the Macros node and select the macro for which a keyboard shortcut should be created.
4. Right-click on the macro and choose Add Keyboard Shortcut in the context menu.
5. In the [Enter Keyboard Shortcut](#) dialog, press the keys to be used as a shortcut. The keystrokes are immediately reflected in the First Stroke field. Optionally, select the Second stroke check box and specify the second stroke. As you press the keys, the Preview field displays the keystrokes you pressed, and the Conflicts field displays warnings, if the keystrokes are already in use.
6. Click OK using the mouse pointer to create a shortcut and bind it with the macro.

Tip It is important that you use the mouse pointer, because any keystroke is interpreted as a shortcut.

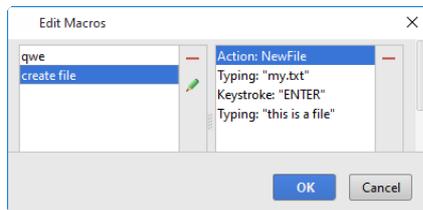
7. Click Apply to save the settings.

Editing Macros

After recording, you can remove or rename any or all of the macros from the list of available macros. The actions list of each macro is also editable - you can remove unnecessary actions.

To edit macros

1. On the main menu, choose Edit | Macros | Edit macros.
2. In left-hand pane of the [Edit Macros Dialog](#), select the macro to be edited or deleted:



3. To delete a macro, click . To change the macro name, click , and specify the new name in the Rename Macro dialog.
4. To change the list of actions for a macro, select an action in the action list, and click .

Playing Back Macros

You can play back the recorded macros using the Edit | Macros menu commands, or [custom shortcuts](#).

To play back a temporary macro

- On the main menu, choose Edit | Macros | Play Back Last Macro.

To play back a named macro

- On the main menu, choose Edit | Macros | <Macro name>.

To play back a macro with a keyboard shortcut

1. Select a keymap with the [macro bindings](#) on the [Keymap](#) settings page.
2. Press keyboard shortcut that corresponds to the desired macro.

To record a macro

1. On the main menu, choose Edit | Macros | Start Macro Recording. From that moment on, all your recordable actions are recorded.
2. When you are done with the procedure, choose Edit | Macros | Stop Macro Recording.
3. In the Enter Macro Name dialog, specify the name of the new macro, and click OK. If the macro is intended for temporary use only, you can leave the name blank.

Spellchecking

On this page:

- [Basics](#)
- [Checking the spelling of a word](#)
- [Configuring the dictionaries to use](#)
- [Configuring spellchecking options](#)

Basics

PyCharm helps you make sure that all your source code, including textual strings, comments, and literals, and commit messages, is spelt correctly. For this purpose, PyCharm suggests a dedicated **Typo** inspection, which is supported by the corresponding bundled plugin and is enabled by default.

Correctness of spelling is checked against pre-defined dictionaries (as of now, `jetbrains.dic` and `english.dic`), and any number of user-defined custom dictionaries.

A **user dictionary** is a textual file with the `dic` extension, containing the words you want to be accepted by the **Typo** inspection as correct. The words in such dictionaries are delimited with the newline.

Besides that, you can define your own list of words that will be skipped by the inspection. You can add words to this list "on-the-fly", or intentionally while setting up your spellchecker options.

With the **Typo** inspection enabled, PyCharm detects and highlights words not included in dictionaries and user's words list. It up to the user to provide correct spelling, accept word as is, or disable inspection.

If a word is accepted, it will be added to the user's words list, and skipped by the spellchecker in future. If inspection is disabled, all typos will be ignored.

In the textual strings and comments, spelling of a word at caret can be changed to a correct one. In the contexts that enable **Rename** refactoring, the inspection suggests to rename all occurrences of a symbol.

Checking the spelling of a word

1. Place the caret on a word highlighted by the **Typo** inspection.
2. Press `Alt+Enter` to show the available intention actions.
3. Choose one of the following actions:
 - Change to: select the desired spelling of a textual string or comment from the suggestion list.
 - Save to dictionary: add word to the user's list and skip it in future.

Configuring the dictionaries to use

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Spelling under Editor.
2. Define the set of dictionaries to be used for spellchecking.
 1. Switch to the Dictionaries tab.
 2. The Dictionaries area in the bottom of the page shows a list of the dictionaries that can be used in spellchecking. The list contains the dictionaries that come bundled with PyCharm by default and user-defined dictionaries detected in the folders from the Custom Dictionaries Folder area above.
 - To have a default dictionary applied in the current project, select the check box next to it.
 - To exclude a default dictionary from spellchecking within the scope of the current project, clear the check box next to it.
 3. In the Custom Dictionaries Folder area, configure your custom dictionaries to use. This area displays a list of directories that contain user-defined dictionary files (text files with the `dic` extension, containing words separated with a newline).
 - To add a new folder to the list, click `+` and choose the required folder in the [Select Path Dialog](#) dialog that opens.
The full path to the folder is added to the Custom Dictionaries Folder list, and all the `*.dic` files found in this folder are added to the Dictionaries list.
 - To remove a folder from the list, select it and click `-`.
3. Besides configuring a custom dictionary, you can create your own **Word List** with the words that you want to be skipped during spell checking without being included in a custom dictionary.
 1. Switch to the Accepted Words tab.
 2. Create a **Word List**:
 - Click the `+` icon to open the Add New Word dialog box and specify a new entry there. **CamelCase** or **snake_case** are not supported. If you try to add a word that is already included in one of the spelling dictionaries, PyCharm displays an error message **The word <just typed word> is already in the dictionary**.
 - To remove an item from the list, select it and click `-`.

Configuring spellchecking options

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Inspections under Editor.
2. On the [Inspections](#) page, that opens showing a list of inspection types, expand the Spelling node and click Typo in the central pane.
3. In the right-hand pane, configure the **Typo** inspection:
 - In the Options area, define the type of contents to be inspected by selecting or clearing the Process Code, Process Literals, and Process Comments check boxes.
 - In the Severity area, choose the [inspection severity level](#) and the **scope** to apply this level in.

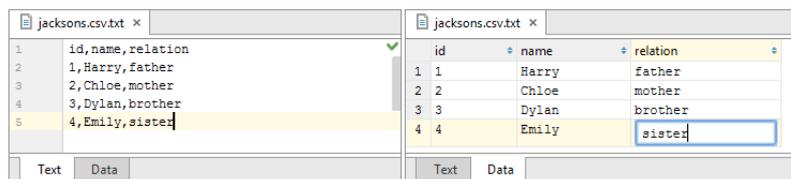
Editing CSV and Similar Files in Table Format

This feature is supported in the Professional edition only.

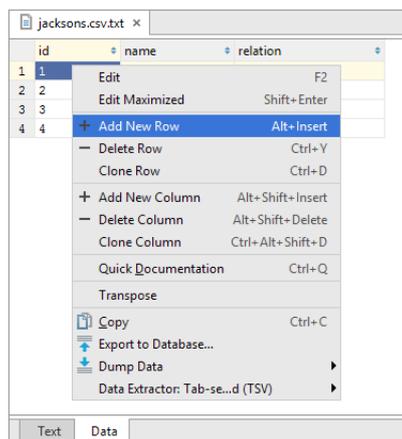
- [Overview](#)
- [Prerequisites](#)
- [Opening the table editor](#)
- [Sorting data](#)
- [Hiding and showing columns](#)
- [Transposing the table](#)
- [Enabling coding assistance for a column](#)
- [Modifying cell contents](#)
- [Adding and deleting rows and columns](#)
- [Copying data to the clipboard or saving them in a file](#)
- [Specifying data output format and options](#)
- [Exporting the data to a database](#)

Overview

For text files containing delimiter-separated values (e.g. CSV, TSV), PyCharm provides an alternative, table editor. (See [Opening the table editor](#).)



Most of the functions in the table editor are accessed as context menu commands. Many of the commands have keyboard shortcuts.



Note that the context menus for the header row and the rest of the table are different.

Prerequisites

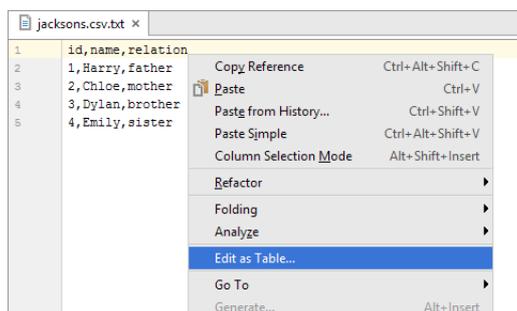
For the table editor and associated features to be available:

- You should be using the Professional Edition of PyCharm. (The corresponding functionality is not available in the Community Edition.)
- The Database Tools and SQL [plugin](#) must be enabled. (This plugin is bundled with the IDE and enabled by default.)
- The file name extension must be associated with the text file type. See e.g. [File Types](#).

Opening the table editor

You can open the table editor for a whole file or for its fragment.

1. Open the file of interest in the editor.
2. If you want to open the table editor for a fragment, select that fragment.
3. Select Edit as Table from the context menu.



4. In the [dialog that opens](#), specify conversion setting and click OK.

Sorting data

You can sort table data by any of the columns by clicking the cells in the header row.

Each cell in this row has a sorting marker in the right-hand part and, initially, a cell may look something like this: . The sorting marker in this case indicates that the data is not sorted by this column.

If you click the cell once, the data is sorted by the corresponding column in the ascending order. This is indicated by the sorting marker appearance:

. The number to the right of the marker (1 on the picture) is the sorting level. (You can sort by more than one column. In such cases, different columns will have different sorting levels.)

When you click the cell for the second time, the data is sorted in the descending order. Here is how the sorting marker indicates this order: .

Finally, when you click the cell for the third time, the initial state is resorted. That is, sorting by the corresponding column is canceled: .

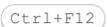
Hiding and showing columns

To hide a column, right-click the corresponding header cell and select Hide column.

To show a hidden column:

1. Do one of the following:

– Right-click any of the cells in the header row and select Column List.

– Press .

In the list that appears, the names of hidden columns are shown struck through.

2. Select (highlight) the column name of interest and press .

3. Press  or  to close the list.

Transposing the table

The transposed table view is available. In this view, the rows and columns are interchanged.

To turn this view on or off, use the Transpose context menu command.

Enabling coding assistance for a column

You can assign a column one of the supported languages (e.g. SQL, HTML or XML): right-click the corresponding header cell, select Edit As and select the language. As a result, you get coding assistance for the selected language in all the cells of the corresponding column.

You can also assign a language to an [individual cell](#).

Modifying cell contents

1. To start editing a value, do one of the following:

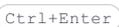
– Double-click the corresponding table cell.

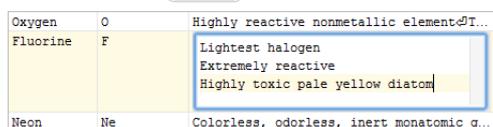
– Right-click the cell and select Edit or Edit Maximized from the context menu.

– Select the cell and press  or . In the latter case, the cell will be maximized.

– Select the cell and start typing. Note that in this case the initial cell contents are deleted right away and is replaced with the typed value.

2. When in the editing mode, you can:

– Modify the value right in the cell. To start a new line, use . To enter the value, press . To restore an initial value and quit the editing mode, press .

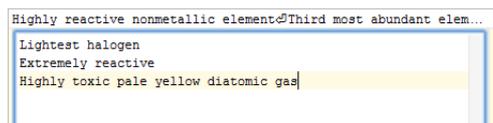


Oxygen	O	Highly reactive nonmetallic element@T...
Fluorine	F	Lightest halogen Extremely reactive Highly toxic pale yellow diatom
Neon	Ne	Colorless, odorless, inert monatomic g...

– Use value completion. Press  to open the suggestion list. The list contains the values from the current column that match your input.

– Maximize the cell if you need more room for editing. To do that, press , or right-click the cell and select Maximize.

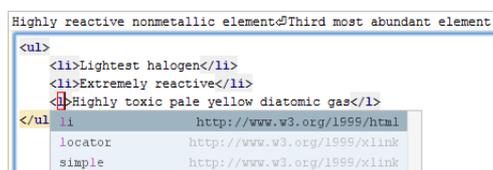
When working in a maximized cell, use  to start a new line and  to enter the value. To restore an initial value and quit the editing mode, press .



Highly reactive nonmetallic element@Third most abundant elem...
Lightest halogen Extremely reactive Highly toxic pale yellow diatomic gas

– Insert the contents of a text file into the cell. To do that, right-click the cell and select Load File. Then select the necessary file in the dialog that opens.

– Edit a value in the cell as a fragment in one of the supported languages (e.g. SQL, HTML or XML). To do that, right-click the cell, select Edit As and select the language. As a result, you get coding assistance for the language you have selected.



Highly reactive nonmetallic element@Third most abundant element
 Lightest halogen Extremely reactive Highly toxic pale yellow diatomic gas
locator http://www.w3.org/1999/xhtml
simple http://www.w3.org/1999/xhtml

Adding and deleting rows and columns

Use the following context menu commands and shortcuts:

- Add New Row ([Alt+Insert](#)).
- Delete Row ([Ctrl+Y](#)). To delete more than one row at once, first, select the corresponding rows or cells in the corresponding rows.
- Clone Row ([Ctrl+D](#)). This command creates a copy of the current row.
- Add New Column ([Shift+Alt+Insert](#)).
- Delete Column ([Shift+Alt+Delete](#)). To delete more than one column at once, first, select the cells in the corresponding columns.
- Clone Column ([Ctrl+Shift+Alt+D](#)). This command creates a copy of the current column.

Copying data to the clipboard or saving them in a file

1. Use one of the following context menu commands:

- Copy ([Ctrl+C](#)). This command copies the data for the selected cells to the clipboard.
- Dump Data | To Clipboard. This command copies the data for the whole table to the clipboard.
- Dump Data | To File. This command saves the data for the whole table in a file. Before actually saving the data, the dialog is shown which lets you select the output format and see how your data will look in a file.

2. If you are saving the data in a file, specify the file name and location.

See also, [Specifying data output format and options](#).

Specifying data output format and options

To specify the output format and options for the Copy and Dump Data commands (see [Copying data to the clipboard or saving them in a file](#)), right-click the table and point to Data Extractor: <current_format>.

In the menu that opens, the output formats are in the upper part: SQL Inserts, SQL Updates, etc. (The options that look like file names are also the output formats or, to be more exact, the scripts that implement corresponding data converters.)

The output option are:

- Allow Transposition. This option affects only delimiter-separated values formats (TSV, CSV). If the table is shown transposed and you are copying selected cells or rows to the clipboard (e.g. [Ctrl+C](#)), the selection is copied transposed (as shown) if the option is on and non-transposed (as in the original table) otherwise.
- Skip Generated Columns (SQL). This is the option for SQL INSERTs and UPDATEs. When on, auto-increment fields are not included.
- Add Table Definition (SQL). This is also the option for SQL INSERTs and UPDATEs. When on, the table definition (CREATE TABLE) is added.

Additionally:

- Configure CSV Formats. This command opens the [CSV Formats Dialog](#) that lets you manage your delimiter-separated values formats (e.g. CSV, TSV).
- Go to Scripts Directory. This command lets you switch to the directory where the scripts that convert table data into various output formats are stored.

Exporting the data to a database

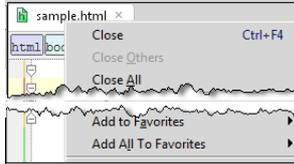
You can export the data to a database (your database must be defined as a [data source](#)):

1. Select Export to Database from the context menu.
2. Select the target schema (a new table will be created) or table (the data will be added to the selected table).
3. In the [dialog that opens](#), specify the data mapping info and the settings for the target table.

You can group most needed items into Favorite lists and get quick access to them through the [Favorites](#) tool window.

To add one or more items to Favorites

1. Do one of the following:
 - Open the desired files in the Editor.
 - Select one or more items in the [Project](#) tool window.
2. Right-click the editor tab or the selection in the Project tool window, and choose Add to Favorites on the context menu.



3. On the submenu, specify the Favorites list to add the selected items to. Do one of the following:
 - To add the items to an existing list, select the desired list in the submenu.
 - To create a new list, choose Add to New Favorites List. In the Add New Favorites List dialog box that opens enter the desired group name or accept default settings.

Managing Editor Tabs

In this section:

- Managing Editor Tabs
 - [Introduction](#)
- [Configuring Behavior of the Editor Tabs](#)
- [Navigating Between Editor Tabs](#)
- [Pinning and Unpinning Tabs](#)
- [Splitting and Unsplitting Editor Window](#)
- [Detaching Editor Tabs](#)
- [Editing Multiple Files Using Groups of Tabs](#)
- [Changing Placement of the Editor Tab Headers](#)
- [Sorting Editor Tabs](#)

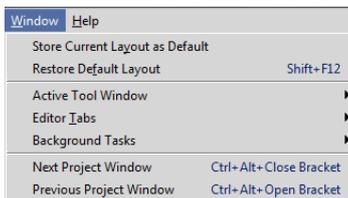
Introduction

Every time you open a file for editing, a dedicated tab is added to the editor window, next to the active editor tab.

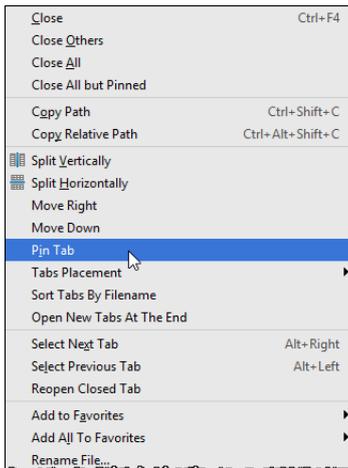
PyCharm can limit the number of tabs opened in the editor simultaneously. When the number of tabs reaches its limit, the editor closes the tabs according to the tab closing policy, that is defined in the [Editor Tabs](#) settings page. By default, the tab limit is 10, but you can change it if necessary.

All commands, related to managing editor tabs, are available from:

- Window | Editor Tabs menu.



- Context menu of a tab.



On this page:

- [Changing the number of editor tabs](#)
- [Disabling editor tabs](#)
- [Tips and tricks](#)

Changing the number of editor tabs

To change the maximum allowed number of tabs

1. Open the Editor settings. To do that, click  on the main toolbar, then click the Editor node in the [Settings](#) dialog box that opens.
2. In the [Editor Tabs](#) page of the Editor settings, type the desired maximum allowed number of the editor tabs to be opened at a time in the Tab limit field.

Disabling editor tabs

To disable editor tabs

- In the [Editor Tabs](#) page of the Editor settings, select None from the Tab placement drop-down list.

Tips and tricks

- If the tab limit equals to 1, the tabs will be disabled. If you want the editor to never close the tabs, type some unreachable number.
- With the disabled tabs, use the View | Recent files () command to quickly switch between files.

On this page:

- [Navigating between editor tabs](#)
- [Navigating through the previously visited tabs](#)
- [Viewing all opened editor tabs and choosing the active editor](#)

To navigate from the current tab to the next or previous tab

- Right-click the current editor tab and choose Select Next/Previous Tab on the context menu.
- Press `Alt+Right` or `Alt+Left`. So doing, the focus moves to the editor tab located next to the right or to the left from the active editor tab.
- Press `Ctrl+Tab` to use the [Switcher](#).

This approach allows jumping from one tab to another as your editing session requires. While you move between the editor tabs, PyCharm remembers the caret position within each opened file.

To go back and forth through the history of visited tabs

- On the main toolbar, click  or .
- On the main menu, choose `Navigate | Back / Forward`.
- Press `Ctrl+Alt+Left` or `Ctrl+Alt+Right`.

This approach enables you to move back and forth through the history of your navigation, same way as it is done in a Web browser. As you move from file to file during your editing session, PyCharm keeps track of the visited locations and enables you to go back, using the `Navigate | Back / Forward` commands.

Note On an macOS computer, you can also use the three-finger right-to-left and left-to-right swipe gestures.

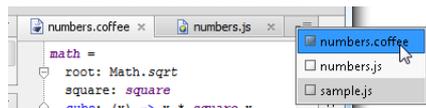
To view all editor tabs and select the active editor

If all opened tabs are shown in a single row, some of the tabs may become invisible. PyCharm helps display the list of the opened editor tabs that did not fit in the editor window and have become invisible whereupon you can choose the tab to activate.

1. Do one of the following:
 - On the main menu, choose `Window | Editor Tabs | Show Hidden Tabs`
 - Click 

The list of all the tabs that are opened but invisible appears. So doing the names of the tabs, which are currently visible, are displayed on the light background; the names of the tabs outside of the main window are shown on the darker background.

2. Click the name of the desired editor tab:



The selected editor tab becomes active and gets the focus.

The command Show All Tabs and the icon  are only available when:

- The Show tabs in single row check box is selected in the [Editor tabs](#) page of the Editor settings.
- Some of the opened tabs are not visible because they do not fit in the editor window.

Pinning and Unpinning Tabs

On this page:

- [Basics](#)
- [Pinning an editor tab](#)
- [Unpinning a tab](#)

Basics

PyCharm can limit the number of tabs opened in the editor simultaneously. When the number of tabs reaches its limit, the editor closes the tabs according to the tab closing policy, that is defined in the [Editor Tabs](#) settings page. By default, the tab limit is 10, but you can change it if necessary.

To prevent a tab from being closed automatically, you can pin this tab. Besides, when you close the editor tabs, you have an option to preserve pinned tabs opened and close only the unpinned tabs.

When a tab is pinned, there is a special marker on it.

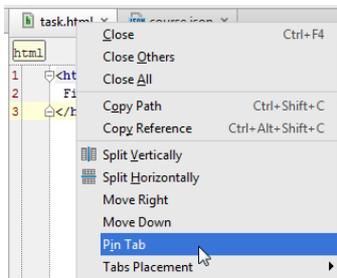
A marker for a pinned tab



Tip The pinned status of a tab is helpful when you open a file for reference, rather than for editing.

Pinning an editor tab

1. Switch to the desired editor tab.
2. Right-click the editor tab, and choose Pin Tab on the context menu:



Unpinning a tab

1. Switch to the desired editor tab.
2. Right-click the editor tab, and choose Unpin Tab on the context menu.

Splitting and Unsplitting Editor Window

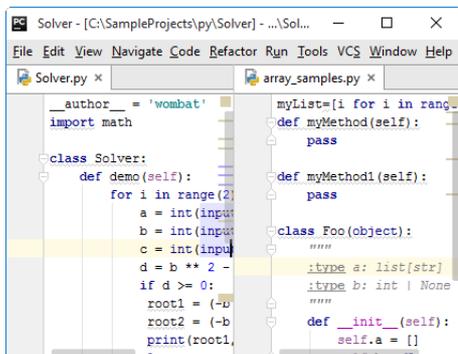
On this page:

- [Basics](#)
- [Splitting editor tab](#)
- [Changing splitter orientation](#)
- [Removing a splitter](#)

Basics

Splitting the editor window divides it into independent panes. You can split the editor window into as many panes as required, each one containing multiple tabs.

Each pane can be allocated vertically or horizontally. Thus, splitting helps create different editor layouts, organize tabs into groups, and [edit multiple files simultaneously](#). For example, you can scroll through a part of a file, having at the same time the possibility to view the lines in its other part.



Splitting editor tab

To split an editor tab creating a file copy

1. Switch to the desired tab.
2. Right-click the tab header and choose Split Vertically or Split Horizontally on the context menu.

To split an editor tab without copying a file

1. Switch to the desired tab.
2. Right-click the tab header and choose Move Right or Move Down on the context menu.

Changing splitter orientation

To change splitter orientation

1. Switch to the desired tab.
2. Right-click the tab header and choose Change Splitter Orientation on the context menu.

Removing a splitter

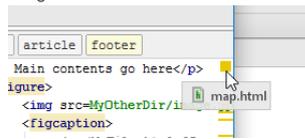
To remove splitter

1. Switch to the desired tab.
2. Right-click the tab header and choose one of the following commands on the context menu:
 - To remove splitting in the active tab, choose Unsplit.
 - To remove splitting in all the open editor tabs, choose Unsplit All.

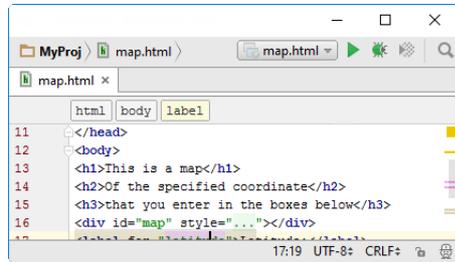
PyCharm makes it possible to detach editor tabs, and move them to separate frames.

To detach an editor tab, do one of the following

- Drag this tab outside of the main window. A preview thumbnail appears:



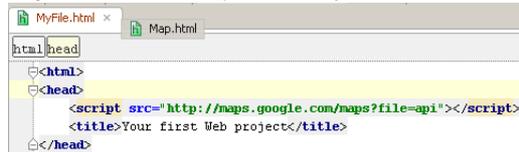
- Press **Shift+F4**.



The content of the editor tab opens in a separate frame.

To attach an editor tab

- drag it from its frame and drop to the main PyCharm frame until tab name appears:



Tip The detached editor tabs can be [split](#) or [unsplit](#).
- You can move editor tabs between split panes.

Editing Multiple Files Using Groups of Tabs

You can arrange tabs into groups to facilitate working with multiple files at a time. PyCharm allows you to have an unlimited number of tab groups, thus enabling you to view several files or several places within the same file.

To create a new group of tabs

- Just [split](#) the desired editor tab.

To move a tab from one group to another

1. Switch to the desired tab.
2. Right-click the desired editor tab and choose Move Tab to Opposite Tab Group on the context menu.

Changing Placement of the Editor Tab Headers

By default, the tab headers appear on top of the editor, but you can change their location as required, and have the headers at the bottom, left, or right sides of the editor. Note that the tab header placement is a global setting that applies to all projects.

To change location of the editor tab headers, do one of the following

- In the [Editor Tabs](#) settings page, select editor tab headers position from the Placement drop-down list.
- Right-click an editor tab, point to the menu item Tabs Placement, and then select the desired placement in the sub-menu.

Sorting Editor Tabs

On this page:

- [Overview](#)
- [Enabling alphabetical sorting](#)
- [Sorting editor tabs alphabetically](#)

Overview

The alphabetical sorting of the editor tabs is available regardless of the tabs position. However, alphabetical order for top and bottom placement becomes available, when the Show tabs in single row check command is selected on the Window | Editor Tabs | Tabs Placement menu.

Enabling alphabetical sorting

To enable alphabetical sorting:

1. On the main menu, choose Window menu.
2. Point to Editor Tabs | Tabs Placement.
3. Select the check command Show Tabs in Single Row.

Sorting editor tabs alphabetically

To sort editor tabs alphabetically:

1. Right-click an editor tab.
2. Select the check command Sort Tabs by Filename.

If this check command is selected, the tabs headers are presented in alphabetical order. Otherwise, the editor tab headers appear in the opening order of the corresponding files.

Using TODO

In this section:

- Using TODO
 - [Introduction](#)
- [Defining TODO Patterns and Filters](#)
- [Creating TODO Items](#)
- [Viewing TODO Items](#)

Introduction

Working on a large project, you often need to create the lists of tasks, and keep your team mates informed about the issues that require their attention. Such issues can include the questions that should be answered, certain changes that should be done later, areas of optimization and improvement etc.

PyCharm suggests to use special TODO comments in the source code. Such comments can be used in all supported file types, and should match a certain TODO pattern. PyCharm comes with one pre-defined pattern, but you can define as many TODO patterns as required. When a matching occurrence is encountered, it is interpreted as a TODO item. PyCharm highlights such comments in accordance with the [Colors and Fonts](#) settings.

Defining TODO Patterns and Filters

On this page:

- [Basics](#)
- [Defining TODO patterns](#)
- [Defining filters](#)

Basics

TODO items in the source code are defined by a certain pattern.

Whenever a pattern is changed, or a new pattern is added, PyCharm scans the whole project and rebuilds the index of TODO items. Results display in the [TODO tool window](#), as described in the section [Viewing TODO Items](#).

By default, PyCharm provides two patterns:

- `\btodo\b.*`
- `\bfixme\b.*`

A generic pattern looks like `todo.*`

You might want to view the TODO comments of certain a type, and hide the others. For this purpose, PyCharm suggests to use filters. This way you can show those items that match certain patterns only.

Defining TODO patterns

To define a TODO pattern, follow these general steps

1. Open the [TODO page](#) of the Settings dialog.
2. In the Patterns section, click the Add button **+** to create a new pattern, or the Edit button  to update an existing one. The [Add/Edit Pattern dialog](#) opens.
3. In the Pattern field, enter the regular expression that describes the desired pattern.
4. In the Icon list, select the desired icon that will mark the matching TODO items in the [TODO tool window](#).
5. Specify the color and font properties, which PyCharm will use to highlight the matching comments in the source code.
6. Select the Case sensitive check box, if you want the pattern to be case-sensitive.

Defining filters

To define a filter that will be used to show specific types of TODO items, follow these general steps

1. Open the [TODO page](#) of the Settings dialog.
2. In the Filters section, click the Add button **+** to create a new filter, or the Edit button  to update an existing one.
3. In the [Add/Edit Filter](#) dialog, specify the filter name, and select the patterns to be included in the filter.

To create TODO items

1. Open the desired file in the editor and position the caret at the place where a TODO item should be created.
2. [Create a comment](#). For example, you can use the `Ctrl+Slash` keyboard shortcut.
3. In the comment, type the string that matches one of your TODO patterns. By default, any string that starts with `TODO` (regardless of the case) is interpreted as a TODO item and is highlighted accordingly.
4. View the list of TODO items in the TODO tool window.

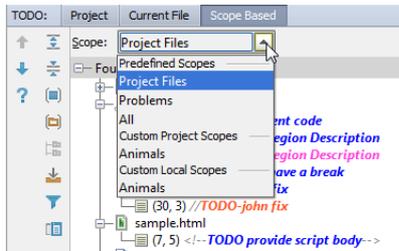
To view TODO items in project, follow these general steps

1. Open TODO tool window, as described in the procedure [Showing a tool window](#).

The tool window displays the encountered TODO items in several tabs:

- All over the project (Project tab)
- In the file currently active in the editor(Current File tab)
- In one of the already defined scopes (Scope Based tab), which is quite useful for large projects.
- In the current changelist, if version control support is enabled.

2. Click the desired tab (view), and explore the list of encountered TODO items. For example, with the Scope Based view selected, one has to choose scope from the drop-down list.



3. Narrow down the list of search results by choosing scope, and applying filters.

Configuring Project and IDE Settings

In this section:

- [Configuring Project and IDE Settings](#)
 - [Introduction](#)
- [Accessing Settings](#)
- [Accessing Default Settings](#)
- [Configuring Colors and Fonts](#)
- [Configuring Keyboard Shortcuts](#)
- [Configuring Menus and Toolbars](#)
- [Configuring Quick Lists](#)
- [Configuring Third-Party Tools](#)
- [Creating and Registering File Types](#)
- [Exporting and Importing Settings](#)
- [Configuring Code Style](#)
- [Copying Code Style Settings](#)
- [Configuring Scopes and File Colors](#)
- [Configuring Individual File Encoding](#)
- [Configuring Line Separators](#)
- [Sharing Your IDE Settings](#)
- [Switching Between Schemes](#)
- [Switching Boot JDK](#)

Introduction

Project settings refer to a set of preferences related to resources, file colors, version control options, code styles, etc. Project settings are stored with each specific project as a set of `.xml` files under the `.idea` folder.

You can configure project settings on the two possible levels:

- The level of a template project. The settings defined for a template project, apply to any project you create.
- The project level. The settings defined on this level apply to the current project only.

Accessing Settings

On this page:

- [Introduction](#)
- [Opening the Settings / Preferences dialog](#)
- [Finding an option or setting](#)
- [Finding an option or setting using Search Everywhere or Find Action](#)

Introduction

This section describes simple steps required to access the Settings/Preferences dialog. Note that the settings that pertain to the current project, are marked with  icon.

Opening the Settings / Preferences dialog

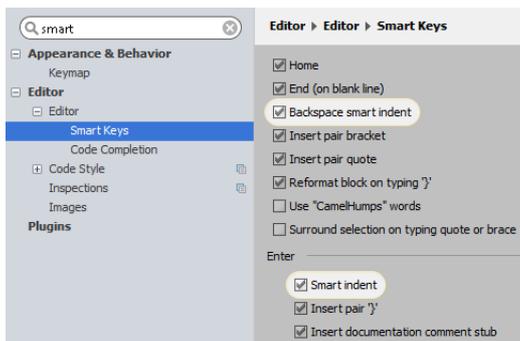
Do one of the following:

- Press `Ctrl+Alt+S`.
- On the main toolbar, click .
- On the main menu, choose File | Settings for Windows and Linux or PyCharm | Preferences for macOS
- Press `Ctrl+Shift+A`, type `settings` and press `Enter`. See [Navigating to Action](#).
- Click  in the upper-right corner of the PyCharm window, and type `#`.

Refer to [Finding an option or setting](#) using Search Everywhere or Find Action below.

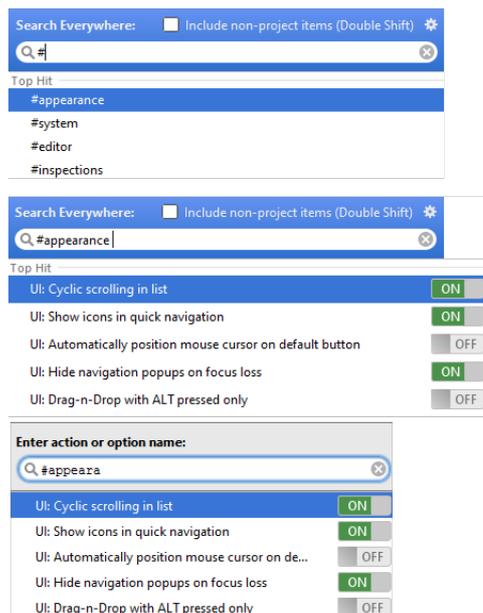
Finding an option or setting

1. [Open the Settings / Preferences dialog](#).
2. In the search field, start typing the text that you expect to find in the name of the setting. As soon as the specified text is found, the matching element is highlighted and the corresponding page is displayed.



Finding an option or setting using Search Everywhere or Find Action

You can also use [Searching Everywhere](#) or [Find Action](#). To find an option or setting, first type `#` character, and then choose one of the suggested categories:



Accessing Default Settings

PyCharm helps define settings of a default project. These settings are used as defaults every time you create a new project.

To access default project settings

1. On the main menu, choose File | Default Settings.
2. Define the desired settings in the [Settings](#) dialog box that opens.

Configuring Colors and Fonts

On this page:

- [Basics](#)
- [Configuring colors and fonts](#)
- [Example](#)
 - [Changing language defaults](#)
 - [Changing font for JavaScript](#)
- [Semantic highlighting](#)

Basics

With PyCharm, you can maintain your preferable colors and fonts layout for syntax and error highlighting in the editor, search results, Debugger and consoles via font and color schemes.

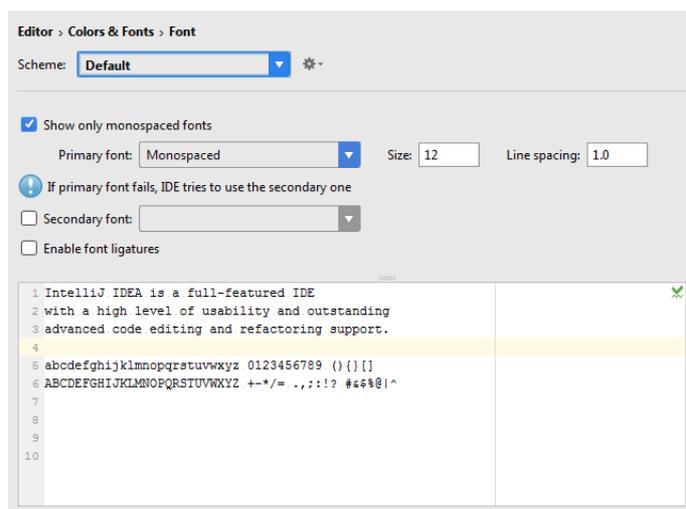
PyCharm comes with a number of pre-defined color schemes. You can select one of them, or create your own one, and configure its settings to your taste.

It's important to mention the node Language Defaults - it contains the settings that are common for all the supported languages. It's enough to change one of the settings there, and then inherit this setting from the defaults.

Configuring colors and fonts

To configure color and font scheme

1. Open Settings/Preferences dialog, and under the Editor node, click [Colors & Fonts](#).
2. Select the desired scheme from the Scheme name drop-down list.
3. Under the [Colors and Fonts](#) node, change the **font families** used in the editor and in the console:
 - Define font family for the editor and console. When you just open the Font or Console Fonts pages under the [Colors and Fonts](#) node, PyCharm displays Editor Font area where you can configure primary and secondary fonts, their size and line spacing.



4. Under the [Colors and Fonts](#) node, open pages to configure specific **color preferences** and **font types** for the different supported languages and PyCharm components.

Tip You can view how your customized scheme looks in the editor. To do that, just look at the preview - all the changes are reflected automatically.

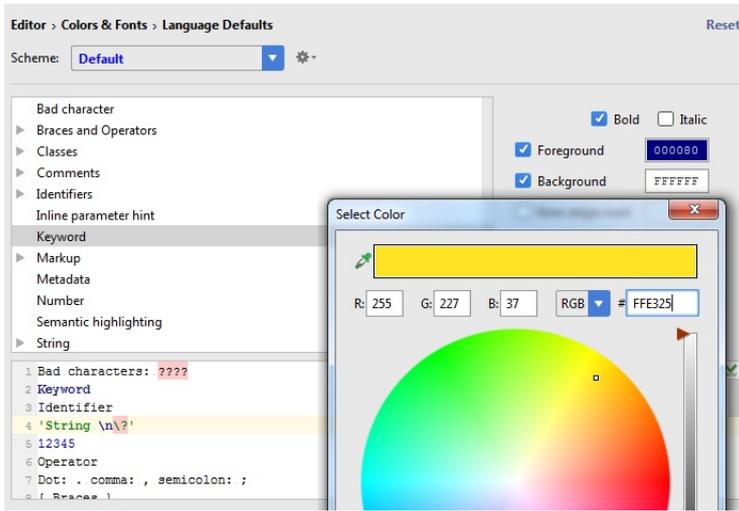
Example

Changing language defaults

The node Language Defaults is actually language-agnostic. It contains the settings that are common to the majority of the supported languages (keywords, dots, commas, parenthesis etc.)

Select the node Language Defaults, and in the list of textual components, select the component Keyword. The background of the keywords is white; let's make it yellow.

To do that, select the check box to the left of the field name Background, and then click the white swatch. [Color Picker](#) opens; all you need to do is to select the desired color and click OK:

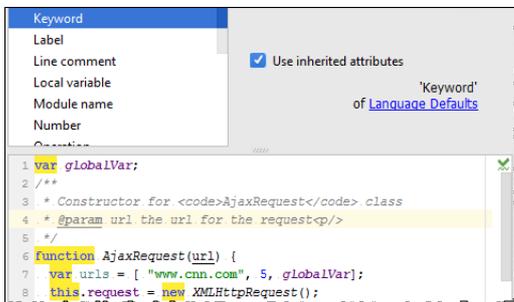


So far, the changes to the language defaults are made; now let's look how they can be inherited.

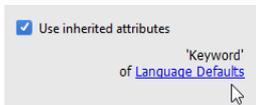
Changing font for JavaScript

Click JavaScript node.

In the list of language components select Keyword, and see that the keywords now have the yellow background:



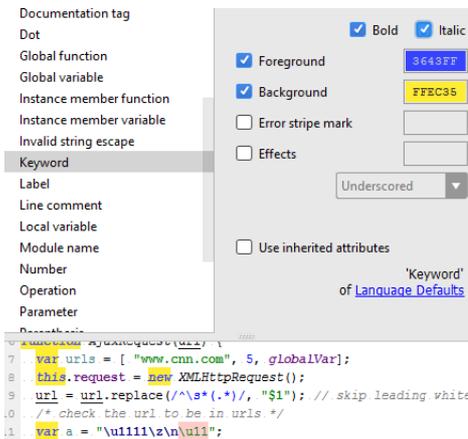
It's important to note that the check box Use inherited attributes is selected!



Clicking the link below this check box leads you to the respective page under Colors and Fonts node, in this case to Language Defaults.

Next, clear the check box Use inherited attributes, and define the desired font type using the Bold and Italic check boxes. In this case, these textual components will change for the selected language only!

Observe results in the preview pane.



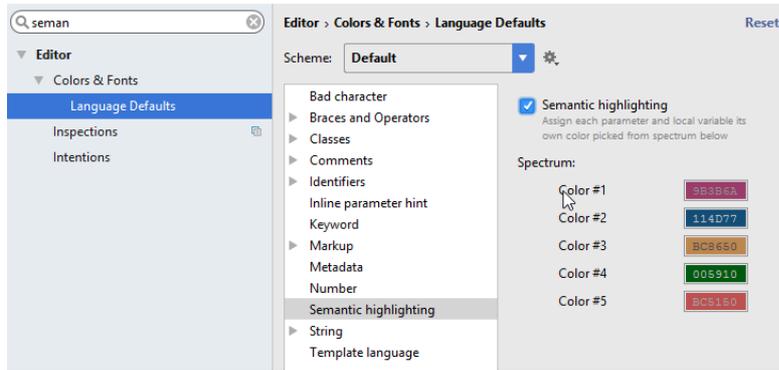
Semantic highlighting

What happens if there is a function/method with a long list of not highlighted parameters? One can easily make PyCharm tell each parameter from the others using the semantic highlighting.

To turn semantic highlighting on, follow these steps:

1. In the Settings/Preferences dialog, click **Colors and Fonts**, and then click the page Language Defaults.
2. In the list of the supported Python components, choose Semantic highlighting.

3. In the right-hand pane, select the check box Semantic highlighting:



After that, all the parameters in a lengthy list will get the colors from the suggested swatches. If one is not happy with the suggested colors, [click on a swatch](#) to choose the suitable color.

Configuring Keyboard Shortcuts

On this page:

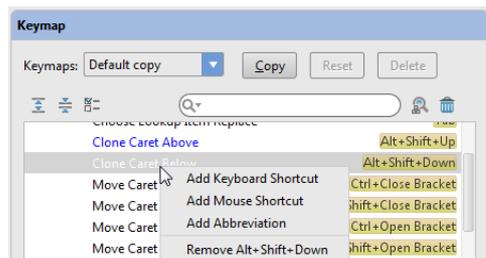
- [Basics](#)
- [Where the keymaps are stored?](#)
- [Configuring keyboard shortcuts and mouse shortcuts](#)
- [Searching for actions](#)

Basics

PyCharm is a keyboard-centric IDE. Most of the actions (navigation, refactoring, debugging, etc.) can be carried out without using a mouse, which lets dramatically increase coding speed. If you had used another IDE for a while and have memorized your favorite keyboard shortcuts, you can use them all in PyCharm.

PyCharm completely suits your *shortcut habits* by supporting customizable keymaps. A keymap is a set of keyboard and mouse shortcuts that invoke different actions - menu commands, editor operations, etc. PyCharm comes with a set of pre-configured keymaps.

Pre-configured keymaps are not editable. If you need to change some shortcuts, the copy of the current selected pre-defined keymap is created automatically:



Where the keymaps are stored?

All user-defined keymaps are stored in separate configuration files under the `config/keymaps` subdirectory in the PyCharm profile directory:

- Windows and *NIX systems: `<User home>/PyCharm<xx>/config/keymaps`
- macOS: `~/Library/Preferences/PyCharm<xx>/keymaps/`

Each keymap file contains only differences between the current and the parent keymaps.

Configuring keyboard shortcuts and mouse shortcuts

To configure keyboard shortcuts and mouse shortcuts

1. [Open the Settings dialog box](#), and click [Keymap](#).
2. Select one of the pre-configured Keymaps, which you want to use as the base for the new one, and click Copy. Accept the default name, or change it as required.
3. In the content pane of actions, select the desired action.
4. Configure keyboard shortcuts. To do that, follow these steps:
 1. Click  on the toolbar, or right-click the selected action, and choose Add Keyboard Shortcut. [Enter Keyboard Shortcut](#) dialog box opens.
 2. Press the keys to be used as shortcuts. The keystrokes are immediately reflected in the First Stroke field. Optionally, select the check box next to Second Stroke and press keys to be used as alternative keyboard shortcuts.

As you press the keys, the Preview field displays the suggested combination of keystrokes, and the Conflicts field displays warnings, if some of the keystrokes are already assigned to the other actions.

3. Click OK with the mouse pointer to create a shortcut and bind it with an action.

It is important to use the mouse pointer, because any keystroke is interpreted as a shortcut.

5. Configure mouse shortcuts. To do that, follow these steps:
 1. Click  on the toolbar, or right-click the selected action, and choose Add Mouse Shortcut on the context menu, if you need to bind an action to a mouse click. [Enter Mouse Shortcut](#) dialog box opens.
 2. In the Click Count section, click a radio button to choose a Single Click or Double Click.
 3. Hover your mouse pointer over the section Click Pad and click the desired mouse button. Use , , and  modifiers for diversity. As you click, the Shortcut Preview field displays the suggested shortcut, and the Conflicts field displays warnings, if some of the shortcuts are already assigned to the other actions.
 4. Click OK or Press  to create a shortcut and bind it with an action.

If a conflict is reported, a warning message appears. You can choose one of the following options:

- Remove to remove all other bindings and preserve the new one.
- Leave to preserve all bindings including the new one.
- Cancel to return to the keymap definition.

Although you can ignore conflict and bind a shortcut with several actions, it is strictly recommended to avoid binding two actions with the same shortcut.

because the order of performing such actions is not defined.

Searching for actions

Tip Use the [Keymap](#) page to search for certain actions by name, or by shortcut.

To find an action by name, type this name in the search field . As you type, the content pane shows actions with the matching names.

To find an action by shortcut, click . In the Filter Settings dialog box, start pressing keys. The content pane shows only the actions with the matching shortcuts.

Click your mouse somewhere outside the Filter Settings dialog box to close it. Avoid using keys, since any keystroke is interpreted as a shortcut.

Configuring Menus and Toolbars

You can customize menu and toolbar command lists to regroup features or make your favorites easier to access.

To customize menus and toolbars

1. Open Settings/Preferences dialog, and click [Menus and Toolbars](#). Alternatively, right-click the main toolbar, and choose Customize Menus and Toolbars on the context menu.

2. In the list of available menus and bars, expand the node you want to customize and select the desired item.

3. Customize the list of items in the selected menu or bar using the buttons on the right from the list:

- To add a new command, select the desired location in the list and click the Add After button. In the [Choose Action To Add](#) dialog box that opens, select the desired action.

Optionally associate the action with an icon using the Icon Path text box. In this text box, specify the location of the file with the icon you want to assign to the selected action. If necessary, use the Browse button  to select the file in the [corresponding dialog](#).

Tip The image file should have `.png` extension.
– The size of the toolbar icons should be 16x16.

- To change the icon associated with a command, select the desired command in the list and click the Edit Action Icon button. In the Choose Actions Icon Path dialog box that opens, specify the location of the desired image. If necessary, use the Browse button  to select the image in the [Select Path Dialog](#).

- To delete an item from the list, select it and click the Remove button.

- To have logical groups of commands separated from each other by a separator, select the desired location in the list and click the Add Separator button.

- To change the order in which commands appear in the selected menu or on the selected bar, use the Move Up and Move Down buttons.

4. To abandon the changes and return to the default settings, click the Restore Default button.

Configuring Quick Lists

On this page:

- [Introduction](#)
- [Using a quick list](#)
- [Configuring a quick list](#)

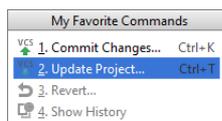
Introduction

A Quick List is a pop-up menu of PyCharm commands, configured by the user and associated with a keyboard or mouse shortcut. You can create as many quick lists, as necessary. Each command, included in a quick list, is identified by a sequential number. Numbering starts from the numerals (0 to 9), and then proceeds with the letters in alphabetical order.

Using a quick list

To invoke a command from a quick list

1. Invoke quick list by its keyboard shortcut.
2. Select the desired command, using its number, the mouse cursor, or navigation keys and the `Enter` key.



Configuring a quick list

To configure a quick list

1. Open Settings/Preferences dialog, and click [Quick Lists](#) page.
2. Click `+` to create a new quick list.
3. In the Display name field, specify the name of the quick list. Optionally, provide the quick list description.
4. Configure the quick list. Use:
 - Add to add actions to the list. Select the actions in the Add Actions to Quick List dialog that opens.
 - Add Separator to add a separator at the end of the list.
 - Move Up and Move Down to move the selected item one line up or down in the list.
 - Remove to remove the selected item from the list.
5. Apply the changes.
6. Bind the new quick list with one or more shortcuts:
 - In the [Keymaps](#) page of the Settings/Preferences dialog, expand the Quick Lists node and select the new quick list.
 - Perform the [key binding procedure](#). Note that you can only modify a custom keymap.
7. Apply the changes and close the dialog.

Configuring Third-Party Tools

You can define third-party standalone applications (code generators and analyzers, pre- and post-processors, database utilities, etc.) as external tools and then run them from PyCharm.

You can pass contextual information (like the currently selected file, or your project source path) to the external tools, view the tool output, and more.

The tools are defined on the [External Tools page](#) in the [Settings dialog](#) and appear as commands in the Tools menu and in various context menus. They can also be assigned keyboard shortcuts (see [Configuring Keyboard Shortcuts](#)).

Creating and Registering File Types

On this page:

- [Introduction](#)
- [Creating a file type](#)
- [Registering a file type](#)

Introduction

You can [create custom file types](#) to enable parsing these files in the editor by defining highlighting schemes for keywords, comments, numbers, etc. To enable PyCharm decide how to treat a file, you need to [associate each file type with relevant extensions](#).

Creating a file type

To create a new file type

1. Open the Settings/Preferences dialog by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS. Then select Editor | File Types. Find more on page [Accessing Settings](#).
2. On the [File Types](#) page that opens, click **+**.
3. In the [New File Type](#) dialog box that opens, specify the name of the new type and optionally provide a description.
4. In the Syntax Highlighting section, specify the characters for line and block comments, hex prefixes, and number postfixes.
5. In the Keywords section, specify sets of keywords using the tabs from 1 to 4. To do so, select the desired tab, click **+** (`Alt+Insert`), and enter the keyword name in the Add New Keyword dialog box that opens.

Tip Each set of keywords has its own highlighting. You can change the highlighting color scheme for each set, on the [Colors and Fonts](#) page. Click the Custom tab and edit the Keyword1, Keyword2, Keyword3, and Keyword4 properties.

Registering a file type

To associate a file type with extensions

1. Open the [File Types](#) settings page.
2. From the Recognized File Types list, select the desired type.
3. In the Registered Patterns area, complete the list of patterns that define the file extensions to indicate that the corresponding files belong to the selected type. Do one of the following:
 - To register a new pattern, click **+** (`Alt+Insert`) and enter the desired extension pattern in the Add Wildcard dialog box that opens.
 - To update a pattern, select it in the list, click the Edit button  and make the necessary changes in the Edit Wildcard dialog box that opens.
 - To remove a pattern from the list, select it and click **-** (`Alt+Delete`).

Exporting and Importing Settings

On this page:

- [Introduction](#)
- [Exporting settings to a JAR archive](#)
- [Importing settings from a JAR archive](#)

Introduction

PyCharm enables you to preserve and share your working environment. You can archive and store your preferred IDE settings, put the settings file under version control and thus make it available to your colleagues. On the other hand, you can use the settings, defined by the other team members, or your own ones intended for a different usage.

Exporting settings to a JAR archive

To export IDE settings to a JAR archive

1. On the main menu, choose File | Export Settings.
2. In the Export Settings dialog box that opens specify the settings to export by selecting the check boxes next to them. By default, all settings are selected.
3. In the Export settings to text box, specify the fully qualified name of the target archive. Type the path manually or click the Browse button  and specify the target file in the [dialog that opens](#) .

Importing settings from a JAR archive

To import settings from a JAR archive

1. On the main menu, choose File | Import Settings.
2. In the Import File Location dialog box that opens select the desired archive.
3. In the Select Components to Import dialog box that opens specify the settings to be imported, and click OK. By default, all settings are selected.

Configuring Code Style

On this page:

- [Basics and definitions](#)
- [Configuring code style for a language](#)
- [Copying code style settings from other languages](#)
- [Configuring the code style for a project using EditorConfig](#)

Basics and definitions

If certain coding guidelines exist in a company, one has to follow these guidelines when creating source code. PyCharm helps maintain the required code style.

Code styles are defined at the project level and at the IDE level (global).

- At the **Project** level, settings are grouped under the Project scheme, which is predefined and is marked in bold. The **Project** style scheme is applied to the current project only.
You can copy the Project scheme to the IDE level, using the Copy to IDE... command.
- At the **IDE** level, settings are grouped under the predefined Default scheme (marked in bold), and any other scheme created by the user by the Duplicate command (marked as plain text). Global settings are used when the user doesn't want to keep code style settings with the project and share them.
You can copy the IDE scheme to the current project, using the Copy to Project... command.

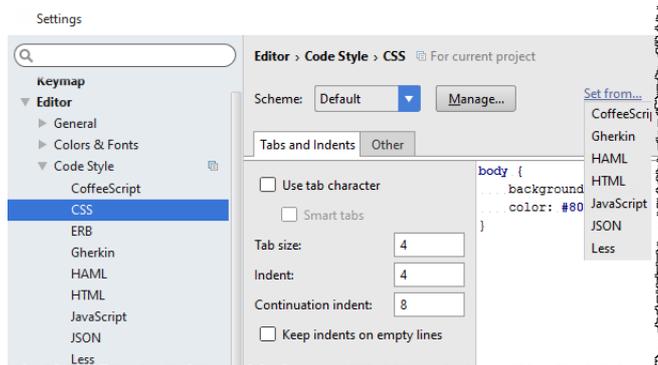
Configuring code style for a language

1. In the Settings/Preferences dialog, click [Code Style](#), and then click the language in question.
2. Choose the code style scheme to be used as the base for your custom coding style for the selected language.
3. Browse through the tabs of the selected language page, and configure code style preferences for it.

Copying code style settings from other languages

For most of the supported languages, you can copy code style settings from other languages or frameworks.

1. In the Settings/Preferences dialog, click [Code Style](#), and then click the language in question.
2. Click the link Set From in the upper-right corner. This link appears for those languages only, where defining settings on the base of the other languages is applicable.
3. In the drop-down list that appears, click the language to copy the code style from:



Configuring the code style for a project using EditorConfig

Before you start working with EditorConfig, make sure that the EditorConfig plugin is enabled. The plugin is bundled with PyCharm and is activated by default. If the plugin is not activated, enable it on the [Plugins](#) page of the [Settings / Preferences Dialog](#) as described in [Enabling and Disabling Plugins](#).

Make sure that the check box Enable EditorConfig Support is selected in Editor | Code Style.

For more information, see [EditorConfig Website](#).

To configure the code style for a project using EditorConfig:

1. In the project tree, right-click a directory where you want to create the EditorConfig settings file and select New | File.
2. In the dialog that opens, enter `.editorconfig` and click OK.
PyCharm creates an EditorConfig settings file and displays a [notification](#) in the pop-up window.

Every time you open a file, the EditorConfig plugin looks for a file named `.editorconfig` in the directory of the opened file and in every parent directory. A search for `.editorconfig` files will stop if the root file path is reached or an EditorConfig file with `root = true` is found. Therefore, if you want to use the IDE settings instead of the EditorConfig settings, clear the Enable EditorConfig Support check box in Editor | Code Style that is selected by default.

3. Start defining your code style settings. Save (Ctrl+S) your file. Every time you modify the `.editorconfig` file, save the file to apply changes to your project.

The EditorConfig code style configuration overrides the code style configuration in the IDE settings.

Copying Code Style Settings

On this page:

- [Introduction](#)
- [Creating a copy of a code style scheme](#)
- [Managing code style schemes](#)

Introduction

You can define the code styles that differ from the pre-defined ones. These code style schemes are stored in XML files, in the `config/codestyles` folder under the user home directory.

You can use the created copy for modifying code styles, and for export.

If you select a code style scheme other than Project, then this code style will be saved for a project. Thus you can assign a global (IDE) code style for each project.

Creating a copy of a code style scheme

To create a copy of code style settings

1. In the [Code Style page](#), select the desired scheme from the drop-down list, and click .
2. From the drop-down list, select one of the following options:
 - Copy to IDE - select this option to store the selected scheme in a global level.
PyCharm saves the new code style with the specified name in the `config/codestyles/<code_style_name>.xml` file under the PyCharm home directory.
 - Copy to Project - select this option to store the selected scheme in a project level.
The selected code style is saved in the `.idea` directory in the file `codeStyleSettings.xml`.
 - Duplicate - select this option to simply make a copy of the selected scheme and store it in the same level.
3. In the Scheme field, type the name of the new scheme and press  to save the changes.
4. Apply changes.

Managing code style schemes

PyCharm lets you modify existing names of code style schemes, export or import code style settings.

To manage a code style scheme

1. In the [Code Style page](#), select the desired scheme from the drop-down list, and click .
2. From the drop-down list, select one of the following options:
 - Rename - select this option to change the name of the selected scheme.
 - Export - select this option to export your code style settings to the desired location.
 - Import - select this option to import PyCharm XML code style settings, JSCS config file, or Eclipse XML Profile.
3. In the Scheme field, type the name of the new scheme and press  to save the changes.
4. Apply changes.

Configuring Scopes and File Colors

This section describes how to configure [scopes](#) and coloring of the files belonging to these scopes:

- [Creating a new custom scope](#)
- [Configuring the list of items in a custom scope](#)
- [Associating file color with a scope](#)
- [Arranging the order of scopes](#)

Creating a new custom scope

Project scopes are configured in the [Scopes](#) page of the [Settings/Preferences](#) dialog box.

To create a new custom scope

1. In the [Scopes](#) settings page, click Add scope **+**.
2. Select Local Changes or Shared from the drop-down list. Shared scopes are defined for the current project and accessible for the team members via VCS, while Local scopes are intended for personal use only and are stored in your workspace.

Tip You can change the sharing state later using the Share scope check box in the bottom of the page.

3. Specify the name for the new scope.
4. Apply changes.

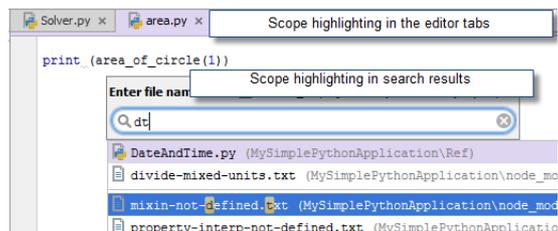
Configuring the list of items in a custom scope

To configure the list of items in a custom scope

1. In the [Scopes](#) settings page, select the scope that you want to configure.
2. Do one of the following:
 - Choose files and folders to be included in the scope and use buttons on the right. Based on the inclusion/exclusion, PyCharm creates an expression and displays it in the Pattern field.
 - Specify pattern in the Pattern field manually, using the [scope language syntax](#).
3. Apply changes.

Associating file color with a scope

Files belonging to different scopes can be highlighted in different colors throughout the PyCharm's user interface: in [navigation lists](#), in the editor tabs, in the [Project](#) window . This allows much faster and easier navigation in large projects.



To associate file color with a scope

1. Open the [File Colors](#) settings page.
2. Decide whether you want the scope-color association to be only available to you or to be shared with the team. Depending on that, select or clear the check box Share colors section of the page.
3. Click Add **+**.
4. In the dialog box that opens, select a scope and pick a color for it.
5. If necessary, use the check boxes on top of the page to define where in the user interface files belonging to the scopes are highlighted.
6. Apply changes.

Arranging the order of scopes

If some file is included into several scopes, the order of the scopes becomes important: PyCharm uses the color of the uppermost scope (shown in the [Scopes](#) settings page) to highlight such file. Of course, you can change the order of the scopes, and thus the resulted highlighting.

To arrange the order of scopes

1. Open the [Scopes](#) settings page.
2. Select a scope whose position in the order you want to change.

3. Click Move Up  / Move Down 

4. Apply the changes.

Configuring Individual File Encoding

On this page:

- [Basics](#)
- [Configuring encoding for a directory or file without embedded encoding information](#)
- [Changing encoding of a file with explicit encoding](#)
- [Changing encoding of a file without explicit encoding](#)

Basics

There are two modes of dealing with file encoding:

- **Converting:** the contents of the editor are stored in a different encoding. So doing, the contents of the underlying file change, but the contents of the editor stay unchanged.
- **Reloading:** the underlying file, opened in the editor, is shown in an encoding that differs from its original one. So doing, the contents of the editor can change, but the underlying files does not.

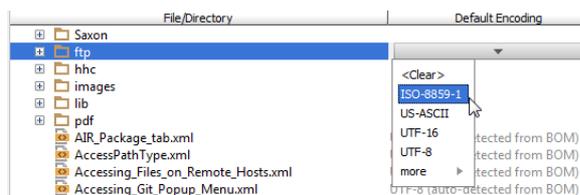
PyCharm suggests the following major ways to change encoding:

- [Using the File Encodings](#) page of the Settings dialog, for directories and for the files that do not contain encoding information.
- [Using the Status bar or menu command](#), for individual files that do not contain encoding information.
- [Using the editor](#), for individual files that contain encoding information.

Configuring encoding for a directory or file without embedded encoding information

To configure encoding for a directory or file without embedded encoding information

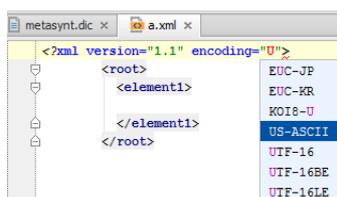
1. In Settings, expand the Editor node and select [File Encodings](#).
2. The File/Directory column shows the tree view of your project. The Default Encoding column shows encoding for directories or files. Click the Default encoding column for a directory or file you want to define encoding for, and then choose the desired encoding from the drop-down list:



Changing encoding of a file with explicit encoding

To change encoding of a file that contains explicit encoding

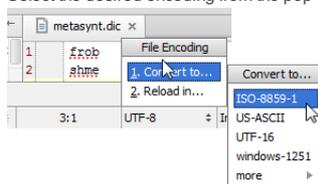
1. [Open the desired file](#) in the editor.
2. Change explicit encoding information. Use error highlighting to recognize wrong encoding and press `Ctrl+Space` to have a list of available encodings displayed:



Changing encoding of a file without explicit encoding

To change encoding of a single file that doesn't contain explicit encoding

1. [Open the desired file](#) for editing.
2. Do one of the following:
 - On the main menu, point to File | File encoding.
 - Click file encoding on the [Status bar](#).
3. Select the desired encoding from the pop-up window.



4. If the selected encoding will change the file contents, PyCharm shows a dialog box, where you can choose to Reload file from disk, or Convert it

to a different encoding.

Configuring Line Separators

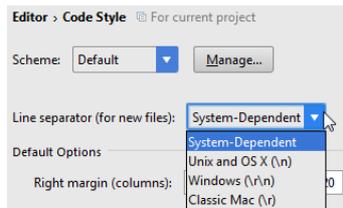
PyCharm makes it possible to set up line separators (line endings) for the newly created files, and change line separator style for the existing files.

On this page:

- [Setting up line separators for newly created files](#)
- [Viewing line ending style for the current file](#)
- [Changing line separator for a file, currently opened in the editor](#)
- [Changing line separator for a selection in the Project view](#)
- [Tips and tricks](#)

To set up line separators for new files

1. In Settings, click [Code Style](#).
2. From the Line separator (for new files) drop-down list, select the desired line separator style:



3. Apply changes and close the dialog.

To view line ending style for the current file

1. Open the desired file in the editor, as described in the section [Opening and Reopening Files](#).
2. View the [Status bar](#): the current line endings style is denoted by the dedicated icon with the specified style, for example, `CRLF`.

To change line separator for a file, currently opened in the editor

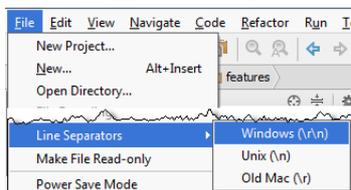
1. Open the desired file in the editor, as described in the section [Opening and Reopening Files](#).
2. Do one of the following:
 - Click the line separator spin box in the [Status bar](#), and choose the desired line ending style from the pop-up menu:



- Choose [File | Line Separators](#) on the main menu, and then choose the desired line ending style from the sub-menu.

To change line separator for a file or directory, selected in the Project view

1. Select a file or directory in the [Project tool window](#).
Note that if a directory is selected, the line ending style applies to all nested files recursively.
2. Choose [File | Line Separators](#) on the main menu, and then select the desired line ending style from the sub-menu.



Tips and tricks

- Use multiple selection in the Project view.
- Changing line separator is reflected in the [Local history](#) of a file.
- [Run the inspection 'Inconsistent line separators'](#) to find out, which files use line separator different from project's default.

Sharing Your IDE Settings

On this page:

- [Introduction](#)
- [Prerequisites](#)
- [Configuring settings repository](#)
- [Authentication](#)
- [Configuring a read-only source](#)

Introduction

PyCharm allows you to share your IDE settings between different instances of PyCharm (or other IntelliJ platform-based) products installed on different computers.

This is useful if you are using several PyCharm installations, or want to implement the same settings among your team members or company-wide.

Prerequisites

Before you start working with Settings Repository, make sure that the Settings Repository plugin is enabled. The plugin is bundled with PyCharm and is activated by default. If the plugin is not activated, enable it on the [Plugins](#) page of the [Settings / Preferences Dialog](#) as described in [Enabling and Disabling Plugins](#).

Configuring settings repository

If you want to share your IDE settings, perform the following steps:

1. Create a Git repository on any hosting service, such as [Bitbucket](#) or [GitHub](#).
2. On the computer where the PyCharm instance whose settings you want to share is installed, navigate to File | Settings Repository. Specify the URL of the repository you've created and click Overwrite Remote.
3. On each computer where you want your settings to be applied, in the [Settings/Preferences dialog](#), expand the Tools node and choose [Settings Repository](#), specify the URL of the repository you've created, and click Overwrite Local.
You can click Merge if you want the repository to keep a combination of the remote settings and your local settings. If any conflicts are detected, a dialog will be displayed where you can resolve these conflicts.

If you want to overwrite the remote settings with your local settings, click Overwrite Remote.

Tip If you select to use [Bitbucket](#) to host your repository, the use of [App passwords](#) is recommended for authentication. You need to set the read/write permissions for your repositories.

Your local settings will be automatically synchronized with the settings stored in the repository each time you perform an Update Project or a Push operation, or when you close your project or exit PyCharm.

If you want to disable automatic settings synchronization, navigate to File | Settings | Tools | Settings Repository and disable the Auto Sync option. You will be able to update your settings manually by choosing VCS | Sync Settings from the main menu.

Authentication

On the first sync, you will be prompted to specify a username and password.

It is recommended to use an [access token](#) for GitHub authentication. If, for some reason, you want to use a username and password instead of an access token, or your Git hosting provider doesn't support it, it is recommended to configure the [Git credentials helper](#).

Note that the [macOS Keychain](#) is supported, which means you can share your credentials between all IntelliJ Platform-based products (you will be prompted to grant access if the original IDE is different from the requestor IDE).

Configuring a read-only source

Apart from the [Settings Repository](#), you can configure any number of additional repositories containing any types of settings you want to share, including live templates, file templates, schemes, deployment options, etc.

These repositories are referred to as **Read-only sources**, as they cannot be overwritten or merged, just used as a source of settings as is.

To configure such repositories, do the following:

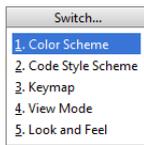
1. In the [Settings/Preferences dialog](#), expand the Tools node and choose [Settings Repository](#).
2. Click **+** and add the URL of the GitHub repository that contains the settings you want to share.

Synchronization with the settings from **read-only sources** is performed in the same way as for the [Settings Repository](#).

Switching Between Schemes

You can quickly switch between various color schemes, keyboard layouts, and look-and-feels without actually invoking the corresponding page of the [Settings](#) dialog box.

1. Choose View | Quick Switch Scheme on the main menu or press `Ctrl+Back Quote`.
2. In the pop-up window that opens select the desired scheme (Colors and Fonts, Code Style, etc.).



3. In the suggestion list, click the desired option.



Switching Boot JDK

Warning! Regardless of your choice, the selected version of JDK shall be not lower than 1.8.
– Currently, this option is available for macOS and Linux users only

On this page:

- [Introduction](#)
- [Switching the IDE boot JDK](#)

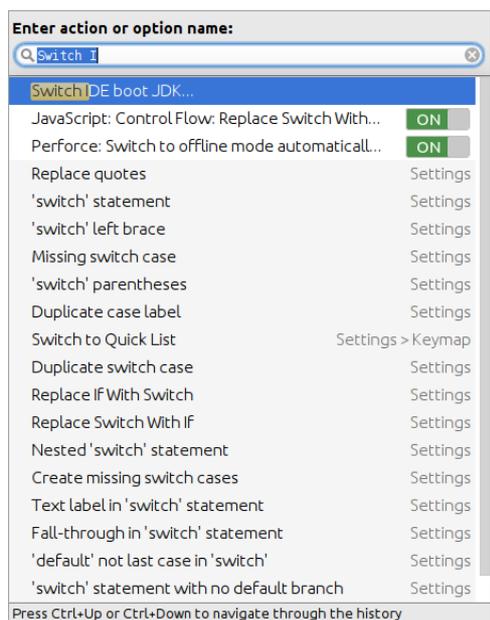
Introduction

In case when you prefer JDK other than bundled with PyCharm, you can choose between the latter and another kit, installed on your system.

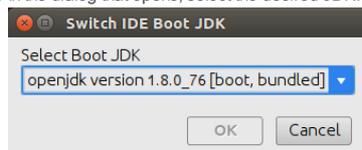
Switching the IDE boot JDK

To switch the IDE boot JDK, do the following:

1. On the main menu, choose Help | Find Action or press `Ctrl+Shift+A`.
2. In the list of actions that appears, find Switch IDE boot JDK action and select it. Simplify your search by typing the first letters:



3. In the dialog that opens, select the desired JDK:



4. Click OK to apply changes.

Creating and Managing Projects

In this section:

- [Creating Projects from Scratch in PyCharm](#)
- [Importing Project from Existing Source Code](#)
- [Generating a Project from a Framework Template](#)
- [Creating a Project Using Yeoman Generator](#)
- [Cleaning System Cache](#)
- [Opening, Reopening and Closing Projects](#)
- [Opening Multiple Projects](#)
- [Switching Between Open Projects](#)
- [Creating Project Remotely](#)

Creating Projects from Scratch in PyCharm

PyCharm makes it possible to create projects of the [various types](#).

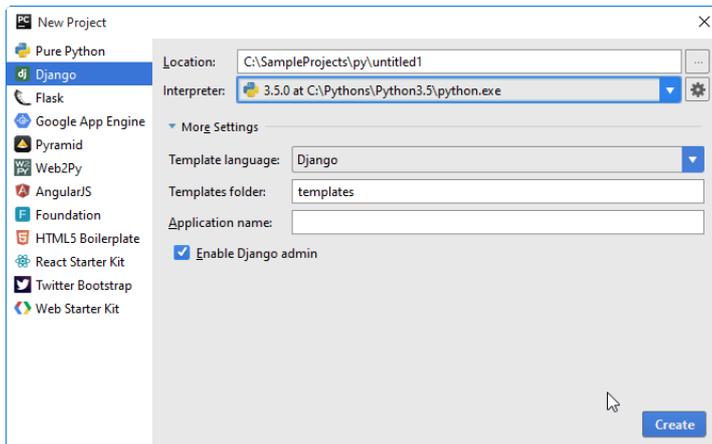
On this page:

- [Creating a project](#)

Creating a project

Follow these general steps:

1. Do one of the following:
 - On the [Welcome screen](#), click the link Create New Project.
 - On the main menu, choose File | New Project.
2. In the dialog box that opens, specify the project location (or accept the default one), and choose its type. From the Python interpreter drop-down list, choose the interpreter to be used for the new project.



Note that the list shows all previously configured interpreters and virtual environments. If you cannot find the desired interpreter in the list, click  to configure the desired [Python SDK](#) or [virtual environment](#).

3. Follow the steps of the wizard, depending on the selected project type, for example:
 - [Pure Python Project](#)
 - [Creating Django Project](#)
 - [Creating Google App Engine Project](#)
 - [Creating Flask Project](#)
 - If you have selected one of the frameworks for the client-side development, choose the desired version of the framework.

Importing Project from Existing Source Code

You can set up a project around the existing source code created externally, in other words, in another IDE or in a dedicated editor. PyCharm analyzes the code base, adds the `.idea` directory with settings, and marks the project with the special icon .

Creating projects from existing source code

To create a new project from existing source code

1. On the main menu, choose File | Open.
2. In the [dialog that opens](#), select the directory that contains the desired source code. Note that applications created externally are marked with the regular directory icon .
3. Click OK.
4. Specify whether you want the new project to be opened in a separate window or close the current project and reuse the existing one. Refer to the section [Opening Multiple Projects](#) for details.

Generating a Project from a Framework Template

This feature is supported in the Professional edition only.

On this page:

- [Overview](#)
- [Generating a HTML5 Boilerplate, Web Starter Kit, React Starter Kit, Twitter Bootstrap, or Foundation application stub](#)
- [Generating a Node.js Express application stub](#)
- [Generating a Meteor application stub](#)
- [Generating a PhoneGap/Cordova/Ionic application stub](#)
- [Generating an AngularJS application stub](#)

Overview

During project creation, PyCharm can generate a project stub for developing Web applications. The project structure is set up and some sources are generated based on external templates and frameworks downloaded upon your request.

PyCharm generates project stubs based on the following templates:

- [HTML5 Boilerplate](#), [Twitter Bootstrap](#), and [Foundation](#) for client-side application stubs.
- [Node.js Express](#) for the server-side applications using [Node.js](#) and [Node.js Boilerplate](#)
- [Dart](#) Web application stubs for developing the client side of applications.
- [Meteor](#) project stubs for both the client and the server sides of applications.
- [PhoneGap/Apache Cordova/Ionic](#) project stubs.
- [AngularJS](#) project stubs.

Generating a HTML5 Boilerplate, Web Starter Kit, React Starter Kit, Twitter Bootstrap, or Foundation application stub

1. Choose File | New Project on the main menu or click the Create New Project button on the Welcome screen. The [New Project Dialog](#) dialog box opens.
2. In the left-hand pane, choose the template to use depending on your preferences and the functionality to be implemented in your application:
 - Empty: choose this option to get just a project folder without any contents.
 - HTML5 Boilerplate: choose this option to use the [HTML5 Boilerplate](#) template. This is a starting project template that can be easily adapted to your needs.
 - Web Starter Kit: choose this option to use the [Web Starter Kit](#).
 - React Starter Kit: choose this option to use the [React Starter Kit](#).
 - Twitter Bootstrap: choose this option to use the [Twitter Bootstrap](#) template, which is an elaborate modular toolkit with rich HTML, CSS, and JavaScript support.
 - Foundation: choose this option to use the [Foundation](#) framework.
3. The set of controls in the right-hand pane depends on the chosen template.
 1. In the Location text box, specify the path to the project folder where the project-related files will be stored.
 2. From the Version drop-down list, choose the template version to use and click Create.

Generating a Node.js Express application stub

Generating such application stubs requires that [Node.js](#) is supported in PyCharm:

1. The [Node.js](#) runtime environment is downloaded and installed on your computer.
2. The [Node.js](#) repository plugin is installed and enabled. The plugin is not bundled with PyCharm, but it can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).

Learn more in [Node.js](#)

To generate a Node.js application stub:

1. Choose File | New Project on the main menu or click the Create New Project button on the Welcome screen. The [New Project Dialog](#) dialog box opens.
2. In the left-hand pane, choose Node.js Express App.
3. In the right-hand pane, specify the following:
 1. In the Location text box, specify the path to the project folder where the project-related files will be stored.
 2. The path to the Node.js executable file `node.exe` and to the [Node.js package manager](#) file `npm.cmd`.
 3. The [Express template engine](#) to use. From the Template engine drop-down list, choose one of the following:
 - [Jade - haml.js](#) successor.
 - [EJS](#) - embedded JavaScript.
 - [Hogan.js](#).
 - [Handlebars](#).
 4. The CSS engine to use. From the CSS engine drop-down list, choose one of the following:
 - [Plain CSS](#)
 - [Stylus](#)
 - [Less](#)
 - [Compass](#).
 - [Sass](#).

4. Click Create, when ready. PyCharm launches the Express Project Generator tool that downloads all the required data, sets up the project structure, and opens the project either in the current window or in a new one, depending on your choice.

Generating a Meteor application stub

When you click Create, PyCharm generates a skeleton of a **Meteor** application, including an HTML file, a JavaScript file, a CSS file, and a `.meteor` folder with subfolders. The `.meteor/local` folder, which is intended for storing the built application, is automatically marked as **excluded** and is not involved in indexing.

By default, **excluded** files are shown in the project tree. To have the `.meteor/local` folder hidden, click the  button on the toolbar of the Project tool window and remove a tick next to the Show Excluded Files option.

PyCharm also automatically attaches the predefined **Meteor library** to the project, thus enabling syntax highlighting, resolving references, and code completion. See [Configuring JavaScript Libraries](#) for details.

Meteor uses **Spacebars** templates that are an extension of the **Handlebars/Mustache** templates. PyCharm recognizes **Spacebars** templates, but as a side effect marks HTML files in **Meteor** projects with the **Handlebars/Mustache** icon .

1. Choose File | New Project on the main menu or click the Create New Project button on the Welcome screen. The [New Project Dialog](#) dialog box opens.
2. In the left-hand pane, choose Meteor App.
3. In the right-hand pane:
 1. In the Location text box, specify the path to the project folder where the project-related files will be stored.
 2. Specify the location of the **Meteor** executable file (see [Installing Meteor](#)).
 3. From the Template drop-down list, choose the sample to generate. To have a basic project structure generated, choose the Default option.
 4. In the Filename text box, type the name for the mutually related `.js`, `.html`, and `.css` files that will be generated. The text box is available only if the **Default** sample type is selected from the Template drop-down list.

Generating a PhoneGap/Cordova/Ionic application stub

1. Choose File | New Project on the main menu or click the Create New Project button on the Welcome screen. The [New Project Dialog](#) dialog box opens.
2. In the left-hand pane, choose PhoneGap/Cordova App.
3. In the right-hand pane:
 1. In the Location text box, specify the path to the project folder where the project-related files will be stored.
 2. Specify the location of the executable file `phonegap.cmd`, or `cordova.cmd`, or `ionic.cmd` (see [Installing PhoneGap/Cordova/Ionic](#)).

When you click Create, PyCharm generates a skeleton of a **PhoneGap/Cordova/Ionic** application with the framework-specific structure.

Generating an AngularJS application stub

This requires that the **AngularJS** plugin is installed and enabled. The plugin is bundled with PyCharm and activated by default. If the plugin is not activated, enable it on the [Plugins](#) page of the [Settings / Preferences Dialog](#) as described in [Enabling and Disabling Plugins](#).

PyCharm can generate a project stub for developing applications using **AngularJS**. To generate an **Angular** stub, use **Angular CLI**, see [Using Angular](#). To generate an Angular 1 application stub:

1. Choose File | New Project on the main menu or click the Create New Project button on the Welcome screen. The [New Project Dialog](#) dialog box opens.
2. In the left-hand pane, choose AngularJS.
3. In the right-hand pane, in the Location text box, specify the path to the project folder where the project-related files will be stored.
4. When you click Create, PyCharm generates the **AngularJS**-specific project structure with all the required configuration files based on the [AngularJS seed project](#)
5. Download the **AngularJS** dependencies that contain the **AngularJS** code and the tools that support development and testing: launch the embedded Terminal (View | Tool Windows | Terminal or by hovering your mouse pointer over  in the lower left corner of PyCharm and choosing Terminal from the menu and type `npm install` at the command line prompt. Learn more about the installation of dependencies in the [Install Dependencies](#) section of the `readme.md` file.

When you click Create, PyCharm generates a skeleton of a **AngularJS** application with the framework-specific structure.

Creating a Project Using Yeoman Generator

This feature is supported in the Professional edition only.

Warning! The following is only valid when Yeoman and Node.js Plugins are installed and enabled!

PyCharm supports integration with the [Yeoman](#) tool and provides interface for using it in generating framework-specific project stubs.

In this section:

- [Before you start](#)
- [Installing Yeoman](#)
- [Configuring a list of project stub generators](#)
- [Creating a project by a generator](#)

Before you start

1. Download and install [Node.js](#). The runtime environment is required for two reasons:
 - The Yeoman tool is started through [Node.js](#).
 - [NPM](#), which is a part of the runtime environment, is also the easiest way to download the Yeoman tool.
2. If you are going to use the command line mode, make sure the path to the parent folder of the [Node.js](#) executable file and the path to the `npm` folder are added to the `PATH` variable. This enables you to launch the Yeoman tool and `npm` from any folder.
3. Make sure the [Yeoman](#) and the [NodeJS](#) plugins are installed and enabled. The plugins are not bundled with PyCharm, but they can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#). Once enabled, the plugins are available at the IDE level, that is, you can use them in all your PyCharm projects.

Installing Yeoman

The easiest way to install [Yeoman](#) is to use the [Node Package Manager \(npm\)](#), which is a part of [Node.js](#). [Yeoman](#) can be installed both [globally](#) or [locally](#), in a project. It is recommended that you install [globally](#) because in this case you can run it from any folder but not from its installation folder.

PyCharm provides interface for global and local installation modes. Alternatively, you can install [Yeoman](#) manually through the command line.

To install [Yeoman](#) from PyCharm using the Node.js and NPM page of the Settings dialog box:

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Node.js and NPM under Languages & Frameworks.
2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the [global](#) and at the [project](#) level. Click `+`.
3. In the Available Packages dialog box that opens, select the `yo` package from the list.
4. For global installation, select the Options check box and type `-g` in the text box next to it.
5. Click Install Package to start installation.

To run the installation from the command line: launch the embedded Terminal (View | Tool Windows | Terminal or by hovering your mouse pointer over  in the lower left corner of PyCharm and choosing Terminal from the menu and type `npm install -g yo` at the command line prompt, where `-g` means [global installation](#).

Configuring a list of project stub generators

1. Choose File | New Project on the main menu or click the Create New Project button on the Welcome screen. The [New Project Dialog](#) dialog box opens.
2. In the left-hand pane, choose Yeoman.
3. The right-hand pane shows all the previously installed [Yeoman generators](#). Click Install Generator.
4. From the dialog box that opens showing all the available generator packages, select the required package in the left-hand pane and click the Install Generator button that appears in the right-hand pane. You can install several packages one after another without leaving the dialog box. When the installation is over, click Close to return to the list of generators which is already expanded with the newly added package.

Creating a project by a generator

1. Choose File | New Project on the main menu or click the Create New Project button on the Welcome screen. The [New Project Dialog](#) dialog box opens.
2. In the left-hand pane, choose Yeoman.
3. The right-hand pane shows all the previously installed [Yeoman generators](#). To expand the list, click Install Generator, see [Creating a Project Using Yeoman Generator](#) above. Select the required generator from the list and click Next.
4. Specify the required settings in the New Project wizard that starts. The number of pages and their contents depend on the chosen generator.
5. On the last page of the wizard, select or clear the Run npm install&bower install check box to specify whether you want to run [Node Package Manager](#) and [Bower](#) to install the packages that are required for developing the new project.
6. Click Next and choose to open the new project in the current window or in the new one.

Cleaning System Cache

PyCharm caches a great number of files, therefore the system cache may one day become overloaded. In certain situations the caches will never be needed again, for example, if you work with frequent short-term projects. Also, the only way to solve some conflicts is to clean out the cache.

Warning! Cleaning out the system caches, keep in mind that:

- It results in [clearing the local history](#).

To avoid losing data, check in the changes to your version control system before invalidating caches.

- Causes a complete rebuild of all the projects ever run in the current version of PyCharm.

Cleaning out the system caches

To clean out the system caches:

- On the main menu, choose File | Invalidate Caches/Restart. The Invalidate Caches message appears informing you that the caches will be invalidated and rebuilt on the next start. Use buttons in the dialog to invalidate caches, restart PyCharm or both.

It is important to note the following:

- The files are not actually deleted until PyCharm restarts.

- Opening and closing a project does not result in deleting any files.

Opening, Reopening and Closing Projects

On this page:

- [Basics](#)
- [Opening an existing project](#)
- [Reopening a recent project](#)
- [Closing a project](#)

Basics

PyCharm allows opening several projects simultaneously in different windows. By default, each time you open a project while another one is opened, PyCharm prompts you to choose whether to open the project in the same window or in a new window. If necessary, you can change this behavior using controls on the [System Settings](#) page.

In the Project Opening section of the [System Settings](#) page, choose one of the following options:

- Open project in a new window to open a new PyCharm window each time a new project is opened.
- Open project in the same window to stay in the same PyCharm window.
- Confirm window to open project in to keep the default behavior and display a dialog box to choose how to open each new project.

Opening an existing project

To open an existing project

This operation refers both to the projects created in PyCharm, and the projects from existing sources.

1. Do one of the following:
 - On the [Welcome screen](#), click Open.
 - On the main menu, choose File | Open.
2. In the [Select Path](#) dialog box, select the directory that contains the desired project.

Tip Drag the desired project from your file chooser right to the Open Project dialog without locating it there. The respective file in the dialog will be found automatically.

3. Specify whether you want to open the project in a new frame, close the current project and reuse the existing frame, or open the new project in the same frame with the current one. Refer to the section [Opening Multiple Projects](#) for details.

Note If a directory with the project created outside of PyCharm, contains a virtual environment, PyCharm will use this virtual environment and make it the project interpreter.

Reopening a recent project

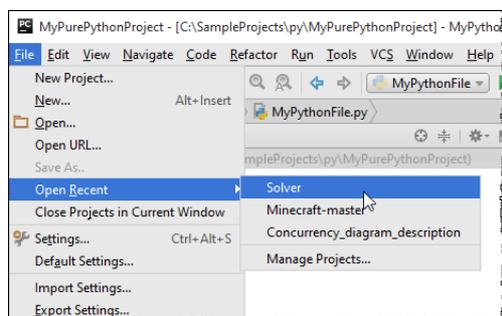
PyCharm keeps the history list of the recent projects, from which you can select the desired one.

When PyCharm starts, the most recent project reopens by default (unless this option is disabled in [System Settings](#) page of the [Settings/Preferences](#) dialog).

To reopen a recent project

1. On the main menu, choose File | Open Recent.
2. Select the desired project from the list of the recent ones.

Note that the list also contains the projects that share one frame:



3. Specify whether you want to open the project in a new frame, or reuse the current frame.

You can also:

- Reopen a recent project from the [Welcome screen](#). To do that, click the project of interest in the left-hand pane where the list of your recent projects is shown. Start typing in the Welcome screen to filter the list of recent projects.
- Terminate the project opening process, without waiting when it finishes loading, by clicking Cancel in the progress window.
- Open projects and individual files from the [command line](#), by specifying the project or file name as a command line argument.
- Remove all projects from the list of recent projects File | Open Recent - Clear List.
- [Delete a certain recent project](#) on the Welcome screen.

Closing a project

When the only open project is closed, the [Welcome screen](#) is displayed. In case of multiple projects, each one is closed with its frame.

To close a project

- On the main menu, choose File | Close Project.

Opening Multiple Projects

On this page:

- [Basics](#)
- [Opening multiple projects](#)
- [Deleting a project from view](#)
- [Important notes](#)

Basics

PyCharm allows you to work with several projects simultaneously. So doing, PyCharm suggests the following options:

- Each project is opened in its own window. The projects are independent, and cannot share information, except for the Clipboard operations. All the projects run in the same instance of PyCharm and use the same memory space.
- A newly opened project shares the same window as the already opened one. So doing, the project that has already been opened, is considered the primary project, and is always shown first in the Project tool window. All the other projects are added to the primary project. Symbols from the added projects are visible from the primary one, but not vice versa.

Some settings ([Django](#), [Buildout](#), [Google App Engine](#), [template languages](#), [project interpreters](#), [content roots](#)) can be configured separately for each project.

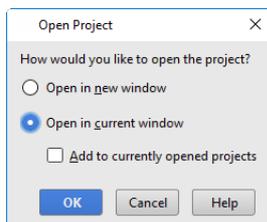
Opening multiple projects

To open multiple projects

- [Open a project](#), while another one is already opened.

Then, depending on the option selected in the [Project Opening section of the System Settings](#) page of the Settings/Preferences dialog, the following happens:

- If the option Open project in a new window is selected, then the new project silently opens in a new window. So doing, the command Attach Project appears in the File menu. Refer to [the description of command](#).
- If the option Open projects in the same window is selected, then the new project silently opens in the same window, replacing the currently opened project. So doing, the command Attach Project appears in the File menu. Refer to [the description of command](#).
- If the option Confirm window to open project in is selected, then PyCharm prompts you to select whether you want to open the project in a new window, or reuse the existing window. This dialog box features the check box Add to currently opened projects. If this check box is selected, the new project opens in the project tree, and adds to the currently opened project. So doing, the project that has already been opened, is the **primary project**.

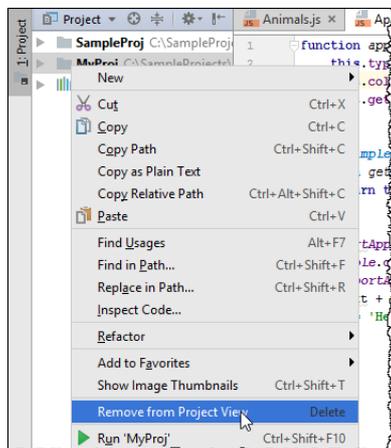


Tip Clicking Open in current window without selecting the check box Add to currently opened projects results in closing the current project and opening the new one in the same window.

Deleting a project from view

If you want to close a project that has been added to the currently opened one (primary project), follow these steps:

1. In the [Project tool window](#), right-click the project to be deleted.
2. On the context menu of the selection, choose Remove from Project View, or just press `Delete` :



Important notes

- When [reopening](#), PyCharm suggests to reopen the whole group of projects.
- The check box Add to currently opened projects appears when you open a new project in a window, where another project is already opened.

Switching Between Open Projects

To switch between open projects, you can use the following commands of the Window menu:

- Window | Next Project Window (`Ctrl+Alt+Close Bracket`)
- Window | Previous Project Window (`Ctrl+Alt+Open Bracket`)
- Window | <ProjectNameOrLocation>

Creating Project Remotely

On this page:

- [Introduction](#)
- [Configuring remote interpreter](#)
- [Creating a project with the remote interpreter](#)

Introduction

If you don't have a Python interpreter on your local development machine, you can create a project remotely. Using PyCharm, create a new project, setting it up for running and debugging on a remote machine.

Configuring remote interpreter

It is important to configure the default interpreter that will be used for any newly created project.

To configure a remote interpreter, follow these general steps:

1. On the main menu, choose File | Default Settings, or on the Welcome screen, click the drop-down Configure and choose Settings.
2. In the [Project Interpreter](#) page of the project settings, click .
3. In the drop-down list, choose Add Remote.



4. Depending on the selection in the dialog box that opens, follow one of the procedures:
 - [Configuring Remote Interpreters via SSH](#)
 - [Configuring Remote Interpreters via Vagrant](#)
 - [Configuring Remote Interpreters via Deployment Configuration](#)
 - [Configuring Remote Interpreters via Docker](#)
 - [Configuring Remote Interpreters via Docker Compose](#)
 - [Configuring Remote Interpreters via WSL](#)

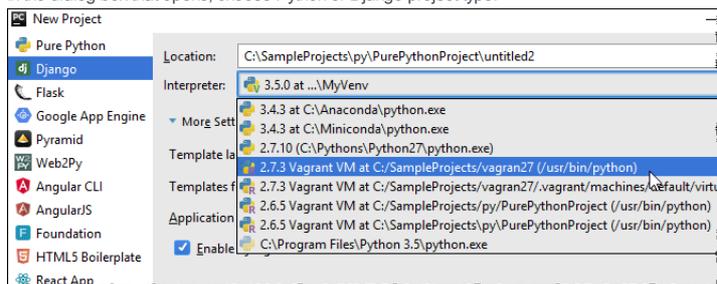
Creating a project with the remote interpreter

Note that PyCharm allows creating both pure Python and Django projects.

To create a remote project, follow these general steps:

1. Do one of the following:
 - On the [Welcome screen](#), click the link Create New Project.
 - On the main menu, choose File | New Project.

2. In the dialog box that opens, choose Python or Django project type.



3. In the Interpreter field, select the desired remote interpreter from the drop-down list.
If the desired interpreter is still missing, click  next to the Interpreter field, choose Add Remote and configure the desired interpreter. Refer to the section [Configuring Remote Python Interpreters](#) for details.

Note that your remote interpreter is unable to sync a project outside of the mapped folders. If your project is located outside of the mapped folder, a red message is shown, until you specify the project location within the mapped folder.

4. Specify the values under More Settings.
5. Click Create.

Configuring Project Structure

In this part:

- [Configuring Project Structure](#)
 - [Basics](#)
- [Accessing Project Structure](#)
- [Configuring Content Roots](#)
- [Configuring Folders Within a Content Root](#)
- [Excluding Files from Project](#)

Basics

To access project structure, use the [Project Structure](#) page of the Settings/Preferences dialog, which is invoked by  icon on the main toolbar, or keyboard shortcut `Ctrl+Alt+S`.

Accessing Project Structure

This section describes simple steps required to access the project structure.

To access project structure

1. Do one of the following:
 - On the main menu, choose File | Settings (for Windows and *NIX) or PyCharm | Preferences for macOS.
 - On the main toolbar, click .
 - Press `Ctrl+Alt+S`.
2. In the Settings/Preferences dialog, click [Project Structure](#) page.

Configuring Content Roots

On this page:

- [Introduction](#)
- [Creating a content root](#)
- [Removing a content root](#)

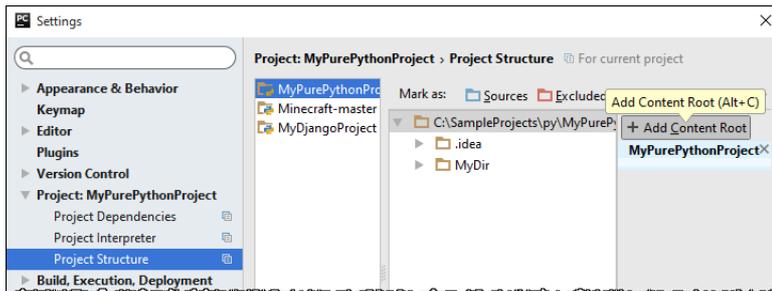
Introduction

Any project contains at least one [content root](#) created together with the project.

You can create additional content roots as well as remove the unnecessary ones.

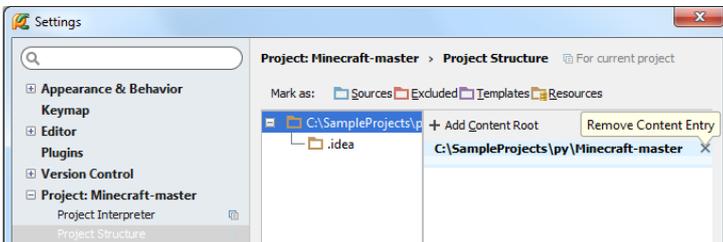
Creating a content root

1. Open the [Project Structure](#) settings.
2. In the Projects pane of the [Project Structure](#) page, click the project you want to configure content roots for.
3. In the Content roots of the [Project Structure](#) page, click the Add Content Root button +.
4. In the [dialog that opens](#), locate the desired directory and click OK.



Removing a content root

- Open the [Project Structure](#) settings.
- In the Projects pane of the [Project Structure](#) page, click the project you want to configure content roots for.
- In the Content roots pane of the [Project Structure](#) page, select the content root to be deleted.
- Click the Remove button .



- Confirm deletion.

Configuring Folders Within a Content Root

In this section:

- [Overview](#)
- [Marking directories](#)
- [Unmarking directories](#)

Overview

Within a **content root**, PyCharm can distinguish between the folders that contain source code, and the ones to be ignored while searching, parsing, watching etc. To do so, you can mark any folder below a content root as a source folder, or as excluded so it becomes invisible for PyCharm.

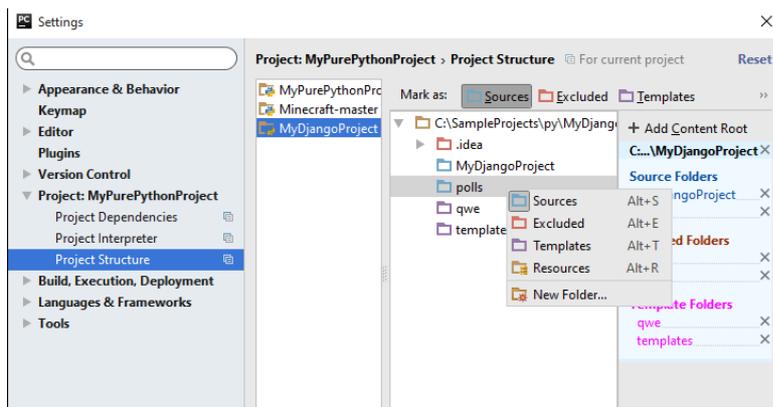
Marking directories

You can assign a folder to a category in two different ways:

- Using the [Project Structure Dialog](#).
- Using the context menu of a folder in the [Project tool window](#).

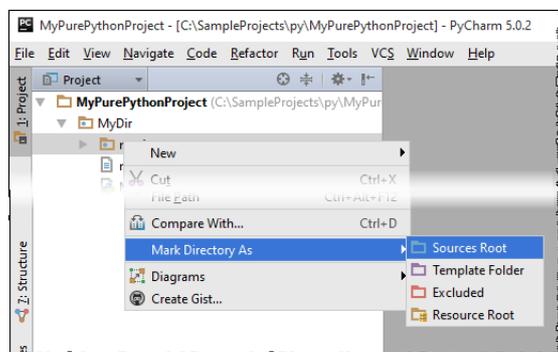
To mark directories under the content root via the Project Structure

1. [Open the Project Structure](#) settings.
2. In the Projects pane of the [Project Structure page](#), click the project you want to configure content roots for.
3. In the Content roots pane of the [Project Structure page](#), click the desired content root. The directories under this content root are displayed as a tree view.
4. Select the directory you want to mark and do one of the following:
 - Click one of the icons on top to assign the desired status to this directory.
 - Choose the corresponding status command on the context menu of the directory.
 - Press **Alt+<first letter of the directory status>** (for example, **Alt+E** for the excluded roots, **Alt+R** for resources, or **Alt+S** for the source roots).



To mark directories using the context menu:

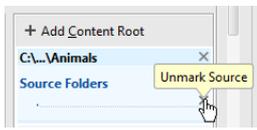
1. Right-click the desired directory in the [Project Tool Window](#).
2. On the context menu, point to Mark Directory As node.
3. Choose Mark as <directory status>.



Unmarking directories

To return a folder to its regular status, do one of the following

- In the [Project Structure Dialog](#):
 - Select the directory in question in the list of folders under the content root, and click **⌘**.



- Click the folder's status icon once more.
- Choose the corresponding command on the context menu of the directory.
- In the [Project Tool Window](#), right-click the desired directory, point to Mark Directory As node, and then choose Unmark as <directory status>.

>

Excluding Files from Project

On this page:

- [Basics](#)
- [Marking a file as plain text](#)
- [Marking a file with its regular type](#)

Basics

Sometimes you might need to exclude a single file from your project, so that it will be ignored by inspections, code completion, etc. This is done using the Mark as plain text action.

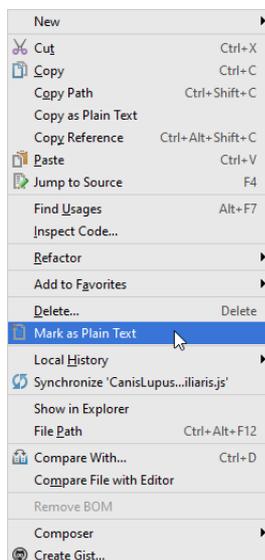
When a file is marked as plain text, PyCharm does not use it anymore for code completion and navigation. Such file is shown as plain text in the editor, and is denoted with a special icon  in the [Project tool window](#).

The reverse action is also available: you can return a file to its original type, using the Mark as <file type> action.

Marking a file as plain text

To mark a file as plain text, follow these steps

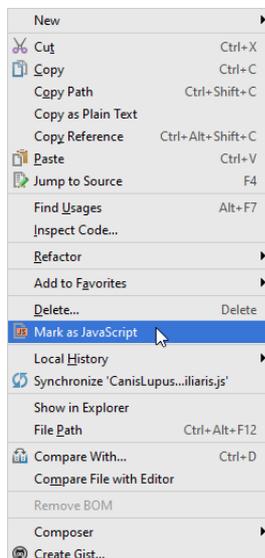
1. In the [Project tool window](#), select the desired file.
2. On the context menu of the selection, choose Mark as plain text:



Marking a file with its regular type

To mark a file with its regular type, follow these steps

1. In the [Project tool window](#), select the desired file, marked with  icon.
2. On the context menu of the selection, choose Mark as <file type>:



File and Code Templates

In this part:

- File and Code Templates
 - [Overview](#)
 - [Per-project vs default scheme](#)
 - [Predefined, internal, and custom templates](#)
 - [When are file and code templates used?](#)
- [File Template Variables](#)
- [#parse Directive](#)
- [Creating and Editing File Templates](#)

Overview

File templates are specifications of the contents to be generated when creating a new file. They let you create the source files that already contain some initial code.

You can view, edit and create the templates on the [File and Code Templates page](#) of the Settings/ Preferences dialog.

Tip In case of [multiple projects](#), settings apply to the main project, which is first in the list in the Project tree.

File and code templates are written in the [Velocity Template Language](#) (VTL).

So they may include:

- Fixed text (markup, code, comments, etc.). In a file based on a template, the fixed text is used literally, as-is.
- [File template variables](#). When creating a file, the variables are replaced with their values.
- [#parse directives](#) to include other templates defined in the Includes tab on the File and Code Templates page of the Settings/Preferences dialog box.
- Other VTL constructs.

Here is a typical template example. (This template is used for creating a JavaScript file.)

```
/**
 * Created by ${USER} on ${DATE}
 */
```

In this template, `${USER}` and `${DATE}` are template variables.

Applying this template leads to generating a file whose contents look similar to this:

```
/**
 * Created by John.Smith on 6/7/11
 */
```

Per-project vs default scheme

PyCharm suggests using file and code templates on the **project** or **default (global)** level.

If you need a sharable set of file and code templates, then these templates should be per-project; otherwise the templates are global and pertain to the entire workspace.

The file and code templates are stored in the following locations:

- The default (global) templates are stored in the PyCharm home directory, in the folder `config | fileTemplates`.
- The per-project file and code templates are stored in the `.idea | fileTemplates` folder. These templates can be shared among the team members.

Refer to the section [Project and IDE Settings](#) to learn where the settings are stored, and to the [File and Code Templates](#) dialog for the description of the Schema field.

Predefined, internal, and custom templates

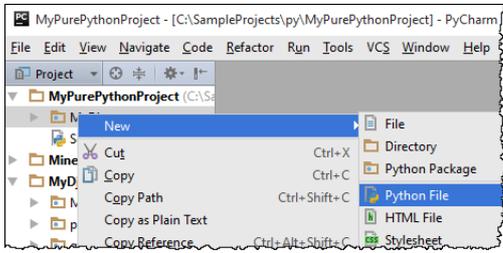
PyCharm comes with a set of predefined file and code templates. You can use these templates as-is or modify them as necessary. You can as well create your own templates (custom templates).

Internal file and code templates are a subset of the predefined templates. These templates differ from all the other templates in that they cannot be deleted.

On the [File and Code Templates page](#) of the Settings/Preferences dialog, the names of internal templates are shown in bold. The names of the custom templates and the predefined templates that you have modified are shown in blue.

When are file and code templates used?

Whenever you create a new file, you can choose to create an empty file (e.g. File | New | File) or use a file template. In the latter case, the initial contents of the new file will be generated according to the template you have selected. (Basically, all the options in the New menu except File, Package and Directory correspond to using a template.)



File Template Variables

On this page:

- [Basics](#)
- [Predefined template variables](#)
- [Custom template variables](#)

Basics

A [file template](#) can contain template variables. When a template is applied, the variables are replaced with their values.

A file template variable is a string that starts with a dollar sign which is followed by the variable name. The variable name may be enclosed in curly braces. For example: `$MyVariable` or `${MyVariable}`.

Predefined template variables

PyCharm comes with a set of predefined template variables.

The available predefined file template variables are:

- `${PROJECT_NAME}` - the name of the current project.
- `${NAME}` - the name of the new file which you specify in the New File dialog box during the file creation.
- `${USER}` - the login name of the current user.
- `${DATE}` - the current system date.
- `${TIME}` - the current system time.
- `${YEAR}` - the current year.
- `${MONTH}` - the current month.
- `${DAY}` - the current day of the month.
- `${HOUR}` - the current hour.
- `${MINUTE}` - the current minute.
- `${PRODUCT_NAME}` - the name of the IDE in which the file will be created.
- `${MONTH_NAME_SHORT}` - the first 3 letters of the month name. Example: Jan, Feb, etc.
- `${MONTH_NAME_FULL}` - full name of a month. Example: January, February, etc.

Custom template variables

In addition to the predefined template variables, it is possible to specify custom variables. If necessary, you can define the values of custom variables right in a template using the `#set` VTL directive.

For example, if you want to use your full name instead of your login name defined through the pre-defined variable `${USER}`, write the following construct:

```
#set( $MyName = "John Smith" )
```

If the value of a variable is not defined in the template, PyCharm will ask you to specify it when the template is applied.

You can prevent treating dollar characters (`$`) in template variables as prefixes. If you need a dollar character (`$`) inserted as is, use the `#{DS}` file template variable instead. When the template is applied, this variable evaluates to a plain dollar character (`$`).

#parse Directive

Using the `#parse` directive, you can include other templates in [file templates](#). This is useful for inserting reusable contents (e.g. standard headers, copyright statements, etc.) into multiple file templates.

The syntax for the `#parse` directive is:

```
#parse("<template_name.extension>")
```

For example: `#parse("ActionScript File Header.as")`.

The templates that can be referenced like this in other templates, are shown on the Includes tab of the [File and Code Templates](#) settings page.

Creating and Editing File Templates

PyCharm supports several ways of creating [file templates](#):

- [Creating a file template from scratch](#)
- [Creating a file template from an existing one](#)
- [Creating a file template from a file](#)
- [Creating and referencing include templates](#)

Creating a file template from scratch

1. Open [Settings/Preferences dialog](#) and under the Editor node, select [File and Code Templates page](#).
2. Switch to the Files tab.
3. Click  on the toolbar and specify the template name, file extension, and the body of the template, which can contain:
 1. Plain text.
 2. `#parse` directives to work with [includes](#).
 3. Predefined variables to be expanded into corresponding values in the format `${<variable_name>}`.
The available predefined file template variables are:
 - `${PROJECT_NAME}` - the name of the current project.
 - `${NAME}` - the name of the new file which you specify in the New File dialog box during the file creation.
 - `${USER}` - the login name of the current user.
 - `${DATE}` - the current system date.
 - `${TIME}` - the current system time.
 - `${YEAR}` - the current year.
 - `${MONTH}` - the current month.
 - `${DAY}` - the current day of the month.
 - `${HOUR}` - the current hour.
 - `${MINUTE}` - the current minute.
 - `${PRODUCT_NAME}` - the name of the IDE in which the file will be created.
 - `${MONTH_NAME_SHORT}` - the first 3 letters of the month name. Example: Jan, Feb, etc.
 - `${MONTH_NAME_FULL}` - full name of a month. Example: January, February, etc.
 4. Custom variables. Their names can be defined right in the template through the `#set` directive or will be defined during the file creation.
4. To have the dollar character (\$) in a variable rendered "as is", use the `#{DS}` variable instead. This variable evaluates to a plain dollar character (\$).
5. Apply the changes and close the dialog box.

Creating a file template from an existing one

1. Open the [File Templates](#) settings page and switch to the Files tab.
2. Click  on the toolbar and change the template name, extension, and source code as required.
3. Apply the changes and close the dialog box.

Creating a file template from a file

1. Open the desired file in the editor.
2. On the main menu, choose Tools | Save File as Template.
3. In the [File and Code Templates](#) dialog box that opens specify the new template name and edit the source code, if necessary.
4. Apply the changes and close the dialog box.

Creating and referencing include templates

Include templates are used to define reusable pieces of code to be inserted in file templates through `#parse` directives.

1. In the [File and Code Templates](#) settings page, switch to the Includes tab.
2. Click  on the toolbar and specify the template name, extension, and the source code. Do one of the following:
 - Use the [predefined file template variables](#).
 - Create custom template variables and define their values right in the include template using the `#set` VTL directive.
For example, if you want to your full name inserted in the file header instead of your login name defined through the `${USER}`, write the following construct:

```
#set( $MyName = "John Smith" )
```

If, when applying a template, the values of certain template variable are not known, PyCharm will ask you to specify them.

You can prevent treating dollar characters (\$) in template variables as prefixes. If you need a dollar character (\$) inserted as is, use the `#{DS}` file template variable instead. When the template is applied, this variable evaluates to a plain dollar character (\$).

3. To use the include template, switch to the Templates tab, select the desired template and click Edit.
4. To include a template, insert the [#parse directive](#) in the source code.

Live Templates

In this part:

- Live Templates
 - [Overview](#)
 - [Important notes](#)
- [Simple, Parameterized and Surround Live Templates](#)
- [Live Template Abbreviation](#)
- [Live Template Variables](#)
- [Groups of Live Templates](#)
- [Creating and Editing Live Templates](#)
- [Creating and Editing Template Variables](#)
- [Sharing Live Templates](#)

Overview

Live templates let you insert frequently-used or custom code constructs into your source code file quickly, efficiently, and accurately.

Live templates are stored in the following location:

- **Windows:** `<your_user_home_directory>\.PyCharm<version_number>\config\templates`
- **Linux:** `~/.PyCharm<version>/config/templates`
- **macOS:** `~/Library/Preferences/PyCharm<version>/templates`

Important notes

- To create or edit the **Live templates**, use the [Live Templates](#) page of the Settings/Preferences dialog.
- Having [created a custom live template](#), you can export all the **live templates** together with the other settings, and import them. Refer to the section [Exporting and Importing Settings](#) for details.

Simple, Parameterized and Surround Live Templates

On this page:

- [Simple live templates](#)
- [Parameterized live templates](#)
- [Surround live templates](#)
- [Examples](#)

Simple live templates

Simple **templates** contain some fixed code that expands into plain text. When invoked and expanded in the editor, the code specified in the template is automatically inserted into your source code, replacing the abbreviation.

Parameterized live templates

Parameterized templates contain plain text and **variables** that enable user input.

Tip If you need a dollar sign (\$) in the template text, escape it by duplicating this character (\$\$).

After a template is expanded, variables appear in the editor as input fields whose values can be either filled in by the user or calculated by PyCharm automatically.

When a parameterized live template is invoked and expanded in the Editor, PyCharm can suggest some predefined values at the input positions of the variables. For example, if a parameterized template contains code for an iteration, then, on expanding the template, PyCharm will suggest:

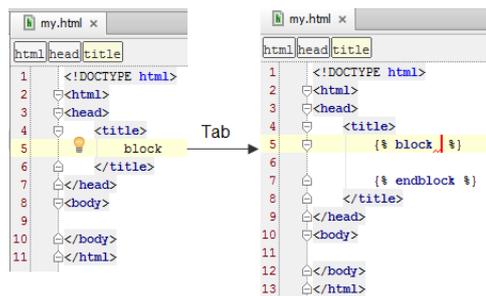
- A name for the index variable (i, j, etc.).
- A name for the assigned variable that holds the current container element.
- Type of the elements in the iterated container.

Surround live templates

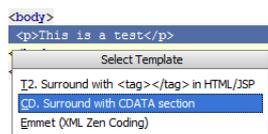
Surround templates work only with blocks of selected text. Such templates place code before and after the selected block.

Examples

Insert parameterized live template (Ctrl+J):



Surround with live template (Ctrl+Alt+J):



Refer also to the page [Wrapping a Tag. Example of Applying Surround Live Templates.](#)

Live Template Abbreviation

Each live template is identified by a template abbreviation.

The template abbreviations work like shortcuts and are expanded into fragments of source code, depending on the surrounding context. So doing, PyCharm can format the generated code fragments according to the code style settings.

An abbreviation may contain alphanumeric characters, dots, and hyphens, and must be unique within its [group](#). However, you can use the same abbreviation for different templates provided that they are in different groups.

Whole code constructs can be created using the template abbreviations. To do that, type the template abbreviation, and press expansion key (for example, `Tab`). Refer to the section [Creating Code Constructs by Live Templates](#) for details.

Live Template Variables

On this page:

- [What are template variables](#)
- [Declaring template variables](#)
- [Creating and editing template variables](#)
- [Predefined template variables](#)
- [Predefined functions to use in live template variables](#)

What are template variables

Template variables in live templates enable user input. After a template is expanded, variables appear in the editor as **input fields**.

Declaring template variables

Variables within templates are declared in the following format:

```
<code>$<variable_name>$</code>
```

Creating and editing template variables

Variables are defined by expressions, and can accept some default values.

This expression may contain constructs of the following basic types:

- String constants in double quotes.
- The name of another variable defined in a live template.
- Predefined functions with possible arguments.

Template variables are editable in the [Edit Template Variables Dialog](#), which contains a complete list of available functions. See the [list of predefined functions](#) below on this page.

Predefined template variables

PyCharm supports two predefined live template variables: `END` and `$SELECTION$`.

You cannot edit the predefined live template variables `END` and `$SELECTION$`.

- `END` indicates the position of the cursor after the template is expanded. For example, the template `return END;` will be expanded into

```
<code>return ;</code>
```

with the cursor positioned **right before** the semicolon.

- `$SELECTION$` is used in surround templates and stands for the code fragment to be wrapped. After the template is expanded, the selected text is wrapped as specified in the template.

For example, if you select `EXAMPLE` in your code and invoke the `"$SELECTION$"` template via the assigned abbreviation or by pressing `Ctrl+Alt+T` and selecting the desired template from the list, PyCharm will wrap the selection in double quotes as follows:

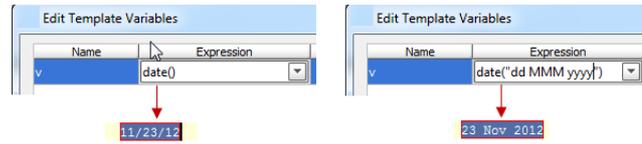
```
<code>"EXAMPLE"</code>
```

Predefined functions to use in live template variables

ItemDescription

<code>camelCase(String)</code>	Returns the string passed as a parameter, converted to camel case. For example, <code>my-text-file / my text file / my_text_file</code> will be converted to <code>myTextFile</code> .
<code>capitalize(String)</code>	Capitalizes the first letter of the name passed as a parameter.
<code>capitalizeAndUnderscore(sCamelCaseName)</code>	Capitalizes the all letters of a CamelCase name passed as a parameter, and inserts an underscore between the parts. For example, if the string passed as a parameter is <code>FooBar</code> , then the function returns <code>FOO_BAR</code> .
<code>classNameComplete()</code>	This expression substitutes for the class name completion at the variable position.
<code>clipboard()</code>	Returns the contents of the system clipboard.
<code>snakeCase(String)</code>	Returns CamelCase string out of snake_case string. For example, if the string passed as a parameter is <code>foo_bar</code> , then the function returns <code>fooBar</code> .
<code>collectionElementName</code>	Removes <code>_list</code> and plural ending (s).
<code>complete()</code>	This expression substitutes for the code completion invocation at the variable position.
<code>completeSmart()</code>	This expression substitutes for the smart type completion invocation at the variable position.
<code>date(sDate)</code>	Returns the current system date in the specified format.

By default, the current date is returned in the default system format. However, if you specify date format in double quotes, the date will be presented in this format:



<code>decapitalize(sName)</code>	Replaces the first letter of the name passed as a parameter with the corresponding lowercase letter.
<code>djangoBlock</code>	Shows completion popup for the available Django blocks.
<code>djangoFilter</code>	Shows completion popup for the available Django filters.
<code>djangoTemplateTags</code>	Shows completion popup for the available Django template tags
<code>djangoVariable</code>	Shows completion popup for the available Django variable.
<code>enum(sCompletionString1,sCompletionString2,...)</code>	List of comma-delimited strings suggested for completion at the template invocation.
<code>escapeString(sEscapeString)</code>	Escapes the specified string.
<code>expectedType()</code>	Returns the type which is expected as a result of the whole template. Makes sense if the template is expanded in the right part of an assignment, after return, etc.
<code>fileName(sFileName)</code>	Returns file name with extension.
<code>fileNameWithoutExtension()</code>	Returns file name without extension.
<code>firstWord(sFirstWord)</code>	Returns the first word of the string passed as a parameter.
<code>lineNumber()</code>	Returns the current line number.
<code>lowercaseAndDash(String)</code>	Returns lower case separated by dashes, of the string passed as a parameter. For example, the string <code>MyExampleName</code> is converted to <code>my-example-name</code> .
<code>snakeCase(sCamelCaseText)</code>	Returns <code>snake_case</code> string out of CamelCase string passed as a parameter.
<code>spaceSeparated(String)</code>	Returns string separated with spaces out of CamelCase string passed as a parameter. For example, if the string passed as a parameter is <code>fooBar</code> , then the function returns <code>foo bar</code> .
<code>pyClassName()</code>	Returns the name of the current Python class (the class where the template is expanded).
<code>pyFunctionName()</code>	Returns the name of the current Python function.
<code>time(sSystemTime)</code>	Returns the current system time.
<code>underscoresToCamelCase(sCamelCaseText)</code>	Returns the string passed as a parameter with CamelHump letters substituting for underscores. For example, if the string passed as a parameter is <code>foo_bar</code> , then the function returns <code>fooBar</code> .
<code>underscoresToSpaces(sParameterWithSpaces)</code>	Returns the string passed as a parameter with spaces substituting for underscores.
<code>user()</code>	Returns the name of the current user.
<code>jsArrayVariable</code>	Returns JavaScript array name.
<code>jsClassName()</code>	Returns the name of the current JavaScript class.
<code>jsComponentType</code>	Returns the JavaScript component type.
<code>jsMethodName()</code>	Returns the name of the current JavaScript method.
<code>jsQualifiedClassName</code>	Returns the complete name of the current JavaScript class.
<code>jsSuggestIndexName</code>	Returns a suggested name for an index.
<code>jsSuggestVariableName</code>	Returns a suggested name for a variable.

Groups of Live Templates

On this page:

- [Overview](#)
- [Managing groups of live templates](#)

Overview

Live Templates are managed on the [Live Templates](#) page of the [Settings / Preferences Dialog](#). For your convenience, templates are arranged in **groups**, most often in accordance with the **context** they are sensitive to.

PyCharm comes with a set of **predefined** groups of templates. In addition to them, you can define your own **custom** groups and templates. It is strongly recommended that you avoid storing custom templates in predefined PyCharm groups. For this purpose, use the **user** group, or a new group.

PyCharm stores definitions of custom live template groups and templates added to predefined template groups in automatically generated configuration files

`<group_name>.xml`.

- For a custom group, the file contains definitions of all the templates the group includes.
- For a customized predefined group, the file contains definitions of the added templates only.

Depending on the operating system you are using, the `<group_name>.xml` files are stored at the following locations:

- **Windows:** `<your_user_home_directory>\.PyCharm<version_number>\config\templates`
- **Linux:** `~/.PyCharm<version>/config/templates`
- **macOS:** `~/Library/Preferences/PyCharm<version>/templates`

Managing groups of live templates

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Live Templates under Editor. The [Live Templates](#) page that opens shows all the groups of live templates that are currently configured in PyCharm.
2. Do one of the following:
 - To create a new custom group, click **+** on the toolbar and choose Template Group on the context menu. Then specify the name for the group in the Create New Group dialog box that opens.
 - To remove a group, select it in the list and click **-** on the toolbar.
 - To add a template to a group, select the group, click **+**, and choose Live Template on the context menu. In the [Template editing area](#), specify the settings for the new template as described in [Creating and Editing Live Templates](#).

Creating and Editing Live Templates

On this page:

- [Introduction](#)
- [Modifying existing templates](#)
- [Creating a new live template from scratch](#)
- [Creating a new live template from a text fragment](#)
- [Searching through the list of live templates](#)
- [Restoring defaults](#)

Introduction

PyCharm comes with a set of the predefined [Live Templates](#). You can use them as is, or modify them to suit your needs. If you want to create a new live template, you can do it from scratch, on the base of the copy of an existing template, or from a fragment of source code.

If a template has been changed, it is always possible to restore its default settings.

Modifying existing templates

To modify an existing template

1. In the Settings/Prefereces dialog, open the [Live Templates](#) page.
2. Expand the desired template group, and select the template you want to change.
3. In the [Template Text](#) area, change the [template abbreviation](#) as required.
4. In the Template Text field, edit the template body which may contain plain text and variables in the format `<variable name>$`.

When editing the live template variables, mind the following helpful hints:

- If you need a dollar sign (`$`) in the template text, escape it by duplicating this character (`$$`).
- To change the variables in a template, click the Edit Variables button and configure the variables as described in [Creating and Editing Template Variables](#).

The Edit Variables button is enabled only if the template body contains at least one user-defined variable, that is, a variable different from `END` or `$SELECTION$`.

Side note about predefined template variables

PyCharm supports two predefined live template variables: `END` and `$SELECTION$`.

You cannot edit the predefined live template variables `END` and `$SELECTION$`.

- `END` indicates the position of the cursor after the template is expanded. For example, the template `return END;` will be expanded into

```
return ;
```

with the cursor positioned **right before** the semicolon.

- `$SELECTION$` is used in surround templates and stands for the code fragment to be wrapped. After the template is expanded, the selected text is wrapped as specified in the template.

For example, if you select `EXAMPLE` in your code and invoke the `"$SELECTION$"` template via the assigned abbreviation or by pressing `Ctrl+Alt+T` and selecting the desired template from the list, PyCharm will wrap the selection in double quotes as follows:

```
"EXAMPLE"
```

5. In the Options section, specify how the template will be expanded and reformatted.
6. In the Available in section, specify the languages and places of code where the editor should be sensitive to the template abbreviation.
7. Click OK when ready.

Creating a new live template from scratch

To create a new template from scratch

1. In the Settings dialog, open the [Live Templates](#) page, and expand the template group where you want to create a new template.
2. Click `+`. A new template item is added to the group and the focus moves to the [Template Text](#) area.
3. Specify the new template abbreviation, type the template body, define the variables and the template group, configure the options, as described in the [template modification](#) procedure.
4. Click OK when ready.

Creating a new live template from a text fragment

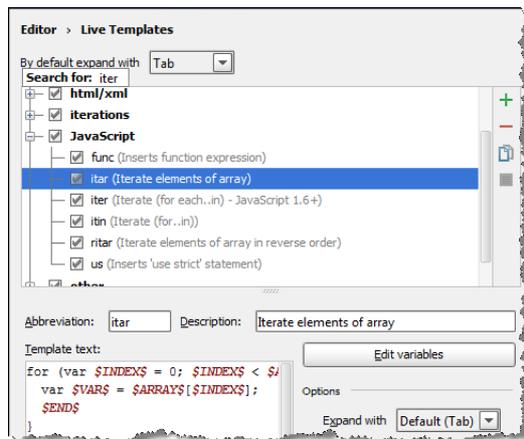
To create a live template from a text fragment

1. In the editor, select the text fragment to create a live template from.
2. On the main menu, choose Tools | Save as Live Template. The [Live Templates](#) page opens, with the [Template Text](#) area in focus.
3. In the Abbreviation field, type abbreviation to identify your new live template.
4. Specify the new template abbreviation, type the template body, define the variables and the template group, configure the options, as described in the [template modification](#) procedure.
5. Click OK when ready.

Searching through the list of live templates

To search through the list of live templates

- In the [Live Templates](#) page, start typing any string, which you expect to exist in the template abbreviation, body, or description. PyCharm shows all matching templates:



Restoring defaults

To restore default settings of a template

- Note that a modified template is color-coded - it is shown blue.
1. In the [Live Templates](#) page, right-click a modified template to reveal its context menu.
 2. Choose Restore defaults on the context menu of the modified template.

Creating and Editing Template Variables

On this page:

- [Basics](#)
- [Configuring variables used in a template](#)
- [Predefined functions to use in live template variables](#)

Basics

After a [template](#) is expanded, its variables are presented in the editor as input fields. The values of these fields can be either filled in by the user or calculated by PyCharm.

To have it done automatically, for each variable you need to specify the following:

- Expression to be calculated in association with the variable.
- Default value to be entered in the input field if the calculation fails.

The order in which PyCharm will process input fields after the template expansion, is determined by the order of variables in the list.

Configuring variables used in a template

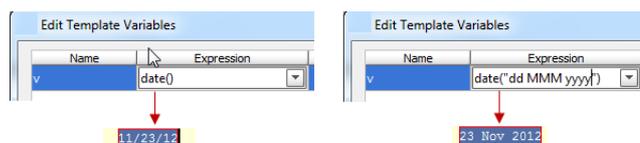
1. [Open the template settings](#), and in the [Template Text](#) area click the Edit Variables button.
The Edit Variables button is enabled only if the template body contains at least one user-defined variable, that is, a variable different from `END` or `$SELECTION$`.

The [Edit Template Variables](#) dialog box opens, where you can define how the variables will be processed when the template is used.
2. In the Name text box, specify the variable name to be used in the template body.
3. In the Expression drop-down list, specify the expression to be calculated by PyCharm when the template is expanded. Do one of the following:
 - Type a string constant in double quotes.
 - Type a predefined function with possible arguments or select one from the drop-down list.
An argument of a function can be either a line constant or another predefined function. See the [list of predefined functions](#) below on this page.
4. To enable PyCharm to proceed with the next input field, if an input field associated with the current variable is already defined, select the Skip if defined check box.
5. To arrange variables in the order you want PyCharm to switch between associated input fields, use the Move Up and Move Down buttons.

Predefined functions to use in live template variables

ItemDescription

<code>camelCase(String)</code>	Returns the string passed as a parameter, converted to camel case. For example, <code>my-text-file / my text file / my_text_file</code> will be converted to <code>myTextFile</code> .
<code>capitalize(String)</code>	Capitalizes the first letter of the name passed as a parameter.
<code>capitalizeAndUnderscore(sCamelCaseName)</code>	Capitalizes the all letters of a CamelCase name passed as a parameter, and inserts an underscore between the parts. For example, if the string passed as a parameter is <code>FooBar</code> , then the function returns <code>FOO_BAR</code> .
<code>classNameComplete()</code>	This expression substitutes for the class name completion at the variable position.
<code>clipboard()</code>	Returns the contents of the system clipboard.
<code>snakeCase(String)</code>	Returns CamelCase string out of snake_case string. For example, if the string passed as a parameter is <code>foo_bar</code> , then the function returns <code>fooBar</code> .
<code>collectionElementName</code>	Removes <code>_list</code> and plural ending (s).
<code>complete()</code>	This expression substitutes for the code completion invocation at the variable position.
<code>completeSmart()</code>	This expression substitutes for the smart type completion invocation at the variable position.
<code>date(sDate)</code>	Returns the current system date in the specified format. By default, the current date is returned in the default system format. However, if you specify date format in double quotes, the date will be presented in this format:



<code>decapitalize(sName)</code>	Replaces the first letter of the name passed as a parameter with the corresponding lowercase letter.
<code>djangoBlock</code>	Shows completion popup for the available Django blocks.
<code>djangoFilter</code>	Shows completion popup for the available Django filters.
<code>djangoTemplateTags</code>	Shows completion popup for the available Django template tags
<code>djangoVariable</code>	Shows completion popup for the available Django variable.

<code>enum(sCompletionString1,sCompletionString2,...)</code>	List of comma-delimited strings suggested for completion at the template invocation.
<code>escapeString(sEscapeString)</code>	Escapes the specified string.
<code>expectedType()</code>	Returns the type which is expected as a result of the whole template. Makes sense if the template is expanded in the right part of an assignment, after return, etc.
<code>fileName(sFileName)</code>	Returns file name with extension.
<code>fileNameWithoutExtension()</code>	Returns file name without extension.
<code>firstWord(sFirstWord)</code>	Returns the first word of the string passed as a parameter.
<code>lineNumber()</code>	Returns the current line number.
<code>lowercaseAndDash(String)</code>	Returns lower case separated by dashes, of the string passed as a parameter. For example, the string <code>MyExampleName</code> is converted to <code>my-example-name</code> .
<code>snakeCase(sCamelCaseText)</code>	Returns <code>snake_case</code> string out of CamelCase string passed as a parameter.
<code>spaceSeparated(String)</code>	Returns string separated with spaces out of CamelCase string passed as a parameter. For example, if the string passed as a parameter is <code>fooBar</code> , then the function returns <code>foo bar</code> .
<code>pyClassName()</code>	Returns the name of the current Python class (the class where the template is expanded).
<code>pyFunctionName()</code>	Returns the name of the current Python function.
<code>time(sSystemTime)</code>	Returns the current system time.
<code>underscoresToCamelCase(sCamelCaseText)</code>	Returns the string passed as a parameter with CamelHump letters substituting for underscores. For example, if the string passed as a parameter is <code>foo_bar</code> , then the function returns <code>fooBar</code> .
<code>underscoresToSpaces(sParameterWithSpaces)</code>	Returns the string passed as a parameter with spaces substituting for underscores.
<code>user()</code>	Returns the name of the current user.
<code>JsArrayVariable</code>	Returns JavaScript array name.
<code>jsClassName()</code>	Returns the name of the current JavaScript class.
<code>jsComponentType</code>	Returns the JavaScript component type.
<code>jsMethodName()</code>	Returns the name of the current JavaScript method.
<code>jsQualifiedClassName</code>	Returns the complete name of the current JavaScript class.
<code>jsSuggestIndexName</code>	Returns a suggested name for an index.
<code>jsSuggestVariableName</code>	Returns a suggested name for a variable.

Sharing Live Templates

On this page:

- [Configuration files with definitions of custom live templates](#)
- [Sharing live templates manually through configuration files](#)
- [Sharing live templates through export/import](#)
 - [Example of sharing templates among different IDE](#)
- [Sharing live templates among template groups](#)

Configuration files with definitions of custom live templates

PyCharm stores definitions of custom live template groups and templates added to predefined template groups in automatically generated configuration files

`<group_name>.xml`.

- For a custom group, the file contains definitions of all the templates the group includes.
- For a customized predefined group, the file contains definitions of the added templates only.

Depending on the operating system you are using, the `<group_name>.xml` files are stored at the following locations:

- **Windows:** `<your_user_home_directory>\.PyCharm<version_number>\config\templates`
- **Linux:** `~/.PyCharm<version>/config/templates`
- **macOS:** `~/Library/Preferences/PyCharm<version>/templates`

Sharing live templates manually through configuration files

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Live Templates under Editor.
2. Create the required custom template groups and update the relevant predefined groups as necessary and click OK. Based on these changes, PyCharm generates the `<group_name>.xml` files, see [Location of Custom Live Templates Definitions](#) above.
3. Locate the generated `<group_name>.xml` files and do one of the following:
 - To share the templates among your teammates, send the relevant files to them with the instruction to save the files in the `templates` folder.
 - To use the templates in another PyCharm installation on your computer, copy the relevant files to the `templates` folder under the relevant `PyCharm<version>` folder.

Sharing live templates through export/import

PyCharm allows you to easily share live templates among your team members, numerous PyCharm installations, and even different IDE by using the standard **Export/Import** functionality. You can share custom template groups and updates to predefined groups.

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Live Templates under Editor.
2. Create the required custom template groups and update the relevant predefined groups as necessary and click OK. Based on these changes, PyCharm generates the `<group_name>.xml` files, see [Location of Custom Live Templates Definitions](#) above.
3. On the main menu, choose File | export Settings.
4. In the Export Settings dialog box that opens, select the Live Template check box and specify the name of the `.jar` file where the exported settings will be stored. When you click OK, PyCharm generates a file with the specified name based on the `.xml` configuration files stored in the `templates` folder.
5. Do one of the following:
 - To share the templates among your teammates, pass the generated `.jar` file to them with the following instructions:
 1. Save the received `.jar` file on your computer.
 2. Choose File | Import Settings on the main menu and specify the location of the received `.jar` file.
 3. In the Select Components to Import dialog box that opens, select the Live Templates check box and click OK.

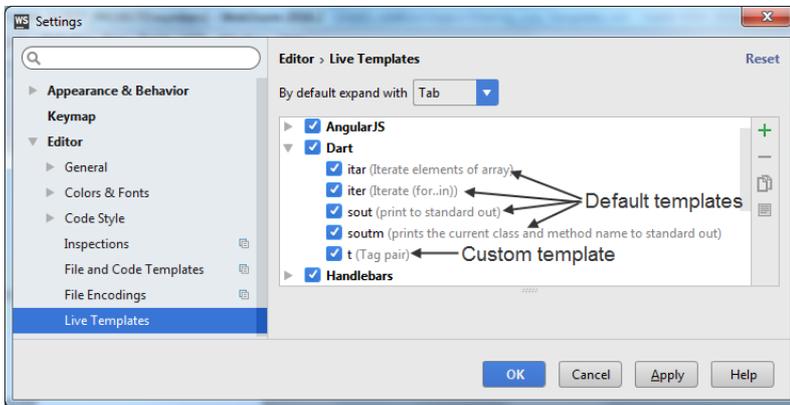
PyCharm restarts whereupon the imported templates are displayed on the Live Templates page.

- To use the templates in another PyCharm installation or in another IDE on your computer, open the required installation, choose File | Import Settings on the main menu, and specify the location of the generated `.jar` file.

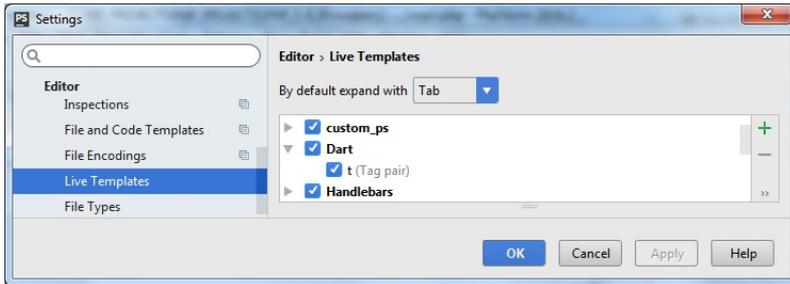
Example of sharing templates among different IDE

Be careful when sharing templates among different IDE. If you import custom templates (updates) from a group which is predefined in the source IDE but is not predefined in the target IDE, such group will be created but will contain only the custom templates. The example below shows what happens if we add a template to a predefined group in **WebStorm** and then reuse it in **PhpStorm**.

In **WebStorm**, the **Dart** template group is predefined. If we add the `t (tag pair)` template to it, this update will be saved in the `Dart.xml` file:



In **PhpStorm**, there is no predefined template group **Dart**. So when we export the live templates from **WebStorm** and then import them into **PhpStorm**, a **Dart** group is created but it contains only one template `t (tag pair)`, which we added to the group in **WebStorm** before export:



Sharing live templates among template groups

You can copy and move templates from one group to another.

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Live Templates under Editor.
2. Do one of the following:
 - To copy a template to another group:
 1. Select the template in interest. Use `Ctrl` and `Shift` keys for multiple selection.
 2. Choose Copy on the context menu of the selection.
 3. Select the group to copy the template to and choose Paste on the context menu of the selection.
 - To move a template to another group, select the required template, choose Move on the context menu of the selection and choose the group to move the template to.

Populating Projects

In this part:

- Populating Projects
 - [Basics](#)
- [Creating Python Packages](#)
- [Creating Directories](#)
- [Creating Files from Templates](#)
- [Creating Empty Files](#)

Basics

Populate your project by creating new elements of various types (directories , packages and files). PyCharm suggests the following alternative ways of accessing the corresponding functionality:

- File | New on the main menu.
- The New context menu command.
- The keyboard shortcut `Alt+Insert`.

As a result, the New menu is shown in which you select the type of an element to be created.

Creating Python Packages

PyCharm makes it possible to create Python packages recursively, thus creating the whole package structure.

The Python package nodes are marked with the  icon.

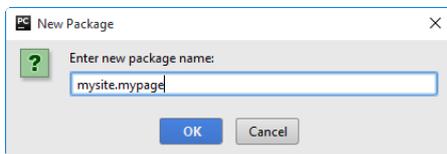
To create a new Python Package

1. In the [Project](#) tool window, select the destination directory.
2. On the context menu of the selection, choose New | Python Package, or press `Alt+Insert` to invoke the context menu:

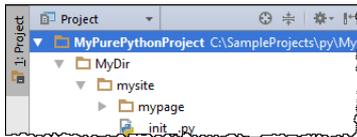


3. In the New Package dialog box that opens, specify the package name.

You can also specify nested packages; in this case, the names should be delimited with dots:



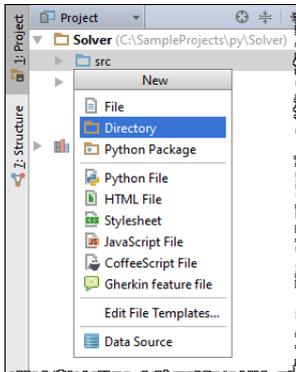
Click OK. PyCharm creates the new package or package structure:



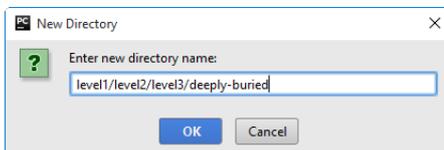
Creating Directories

PyCharm enables you to create directories recursively, thus producing a whole directory structure.

1. Open the Project tool window (e.g. View | Tool Windows | Project).
2. Select the destination directory.
3. Do one of the following:
 - Select File | New | Directory.
 - Select New | Directory from the context menu.
 - Press `Alt+Insert` and select Directory.



4. In the dialog that opens, specify the directory name. If you want to create a number of nested directories, specify the directory names separated with slashes.



Creating Files from Templates

On this page:

- [Introduction](#)
- [Creating a new file from a template](#)

Introduction

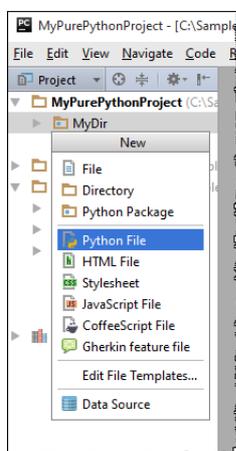
PyCharm provides [file templates](#) for most of the [languages that it supports](#). This lets you create the files with the initial content appropriate for the file purpose. For example, there are file templates for Python, HTML/HTML5/XHTML, and JavaScript files.

Generally, the file name extension for a template-based file is set automatically so you don't need to specify it. For example, if you create a Python script, it automatically gets `.py` extension, a JavaScript file gets `.js` extension. New HTML file gets `.html` extension.

Creating a new file from a template

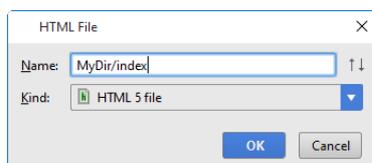
To create a new file from a template

1. Do one of the following:
 - In the [Project tool window](#), select the directory or package in which you want to create a new file, and then choose File | New on the main menu.
 - Right-click the corresponding directory or package and select New from the context menu.
 - Press `Alt+Insert`
2. Select the desired file type. Generally, all the options except File, Package and Directory correspond to using a file template.



An existing file template may be missing from the list if this is a custom template whose file name extension (template extension) does not match registered patterns of any of the recognized file types. In such a case, you may want to register the corresponding pattern for an existing recognized file type or add a new file type and register the corresponding pattern for this new type. For more information, see [Creating and Registering File Types](#).

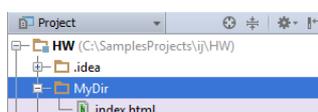
3. In the dialog that opens, type the name of the file in the corresponding field. Note that you should not type the file name extension. You can specify the whole directory structure preceding the new file name. If the nested directories do not yet exist, they will be created:



If required, specify the kind of the new template-based file. For example, if you select to create an HTML file, you'll be able to create HTML, HTML4, or XHTML file. In other words, use one of the corresponding related file templates.

Specify other information as required. For example, you may be asked to define the values of custom variables if the corresponding file template contains such variables and their values are not currently set.

4. Click OK. The new file that corresponds to the selected file template will be created under the target location. If the names of non-existent sub-directories were specified before the new file name, the whole structure will be created under the target directory:



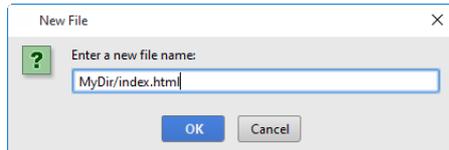
Sometimes, you may want to change the auto-generated file name extension. To do that, use the [Rename refactoring](#) (Refactor | Rename).

Creating Empty Files

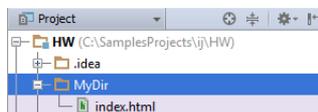
Generally, all the files that you create when developing applications are [template-based](#). However, sometimes you may want to create empty files.

To create an empty file

1. Do one of the following:
 - In the [Project tool window](#), select the directory or package in which you want to create a new file, and then choose File | New on the main menu.
 - Right-click the corresponding directory or package and select New from the context menu.
 - Press `Alt+Insert`
2. Select File.
3. In the New File dialog, in the field under Enter a new file name, type the file name and extension.
You can specify the whole directory structure prepending the new file name. If the nested directories do not yet exist, they will be created:



4. Click OK.
If the names of non-existent sub-directories were specified before the new file name, the whole structure will be created under the target directory:



5. If the extension you have specified is not associated with any of the file types recognized by PyCharm, the Register New File Type Association dialog is displayed. In this dialog, you can associate the extension with one of the recognized file types. To do that, select the file type under Open matching files in PyCharm and click OK. As a result, the extension is associated with the specified file type.

Tip If there are no appropriate file types for the new extension, you may want to create a new file type and associate the extension with that type. For more information, see [Creating and Registering File Types](#).

Generating Code

This section describes how to quickly and accurately populate your source code with the complicated code constructs.

In this part:

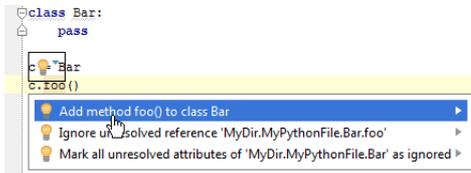
- [Creating Symbols From Usage](#)
- [Creating Code Constructs by Live Templates](#)
- [Creating Code Constructs Using Surround Templates](#)
- [Implementing Methods of an Interface](#)
- [Overriding Methods of a Superclass](#)
- [Surrounding Blocks of Code with Language Constructs](#)
- [Unwrapping and Removing Statements](#)
- [Wrapping a Tag. Example of Applying Surround Live Templates](#)

Creating Symbols From Usage

Suppose you are referring a class or a method that has not yet been created. With PyCharm, you can easily stub out the missing symbol with the aid of the dedicated [intention action](#).

To create a symbol from usage, follow these general steps

1. Type the name that references a non-existent symbol. PyCharm highlights the reference.
2. Press `Alt+Enter`, and choose the corresponding option from the suggestion list, for example Create class <class name>, or Add method <method name> to class <class name>



Creating Code Constructs by Live Templates

On this page:

- [Introduction](#)
- [Inserting a live template](#)

Introduction

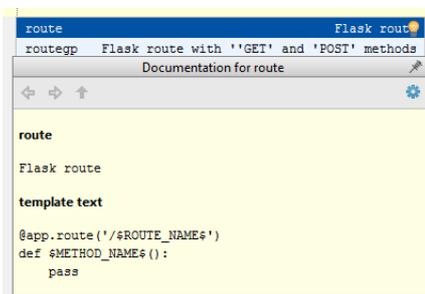
This page describes how to generate source code using [live templates](#).

Using Live Templates enables you to create such code constructs as Django templates, JavaScript iterations, Zen coding, HTML and XML tags.

To explore the list of available live templates, in the Settings/Preferences dialog, open the [Live Templates](#) page.

Inserting a live template

1. Place the caret at the desired position, where the new construct should be added.
2. Do one of the following:
 - On the main menu, choose Code | Insert Live Template.
 - Press `Ctrl+J`.
 - Type some initial letters of the [template abbreviation](#) to get the matching abbreviations in the suggestion list. Note that the suggestion list may contain same abbreviations for different templates.
3. From the [suggestion list](#), select the desired template. While the suggestion list is displayed, it is possible to view Quick Documentation for the items at caret, by pressing `Ctrl+Q`.



4. Press the template invocation key (this may be `Space`, `Tab` or `Enter`, depending on the template definition). The new code construct is inserted in the specified position.
5. If the selected template is [parametrized and requires user input](#), the editor enters the template editing mode and displays the first input field highlighted with the red frame. Type your value in this frame and press `Enter` or `Tab` to complete input and pass to the next input field. After completing the last input field, the caret moves to the end of the construct, and the editor returns to the regular mode of operation.

It is also possible to type a template abbreviation, and then press `Ctrl+J`.

Creating Code Constructs Using Surround Templates

This section describes how to wrap fragments of code according to [surround templates](#).

To surround a block of code with a live template

1. In the editor, select the piece of code to be surrounded.
2. Do one of the following:
 - On the main menu, choose Code | Surround With Live Template....
 - Press `Ctrl+Alt+J`.
3. Select the desired template from the suggestion list.

Implementing Methods of an Interface

On this page:

- [Introduction](#)
- [Implementing methods](#)
- [Changing method body](#)

Introduction

If a class is declared as implementing a certain interface or extending a class with abstract methods, it has to implement the methods of such interface or class. PyCharm creates stubs of the implemented methods, with the default return values for the primitive types, and null values for the objects.

Implementing methods

To implement method of an interface or abstract class , follow these steps:

1. Do one of the following:
 - On the main menu, choose Code | Implement method.
 - Press `Ctrl+I`
 - Right-click the editor, choose Generate on the context menu, or press `Alt+Insert` , and choose Implement methods.

The Select methods to implement dialog appears, displaying the list of classes and interfaces with the methods that can be implemented.

2. Select one or more methods to implement. For multiple selection, use `Ctrl` and `Shift` keys.
3. Click OK.

Changing method body

File template responsible for implementing a method (Implemented method body) accepts predefined template variables from "File Header" File | Settings/PyCharm | Preferences - Editor - File and Code Templates - Code - File Header e.g. `${USER}` , `${DATE}` , etc.

For example, consider the following file template:

Overriding Methods of a Superclass

On this page:

- [Overview](#)
- [Overriding methods](#)
- [Changing method body](#)

Overview

You can override any method of a parent class, using the code generation facility. PyCharm creates a stub that contains a call to the method of the superclass, leaving the developer with the task of providing some meaningful source code.

Overriding methods

To override methods, follow these steps:

1. With the class in question having the focus, invoke the Override method command in one of the following ways:

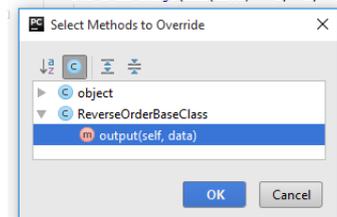
- Press `Ctrl+O`.
- On the main menu, choose Code | Override method.
- Right-click the editor, choose Generate on the context menu, or press `Alt+Insert`, and choose Override methods.

2. Select methods that can be overridden from the Select methods to override dialog box. The list of methods does not include the methods that are already overridden, or cannot be accessed from the current subclass.

```
class ReverseOrderBaseClass:
    def reverse(self, data):
        pass

    def output(self, data):
        pass
```

```
class ReverseOrderChild(ReverseOrderBaseClass):
    def reverse(self, data):
        for i in range(len(data) - 1, -1, -1):
```



3. Select one or more methods to override.

4. Having generated the overriding method, create the required source code. Note the  icon that marks the overriding method in the left gutter. Use this icon to view the name of the base class, and [navigate to the overridden method](#).

```
class ReverseOrderDescendant(ReverseOrderBaseClass):
     Overrides method in ReverseOrderBaseClass
    for index in range(len(data)-1, -1, -1):
        yield data[index]
```

Changing method body

File template responsible for overriding a method (Overridden method body) accepts predefined template variables from "File Header" (File | Settings - Editor - File and Code Templates - Code - File Header), e.g. `${USER}`, `${DATE}`, etc.

For example, consider the following file template:

Surrounding Blocks of Code with Language Constructs

On this page:

- [Applicable contexts](#)
- [Surrounding blocks of code](#)

Applicable contexts

The Surround with feature (Code | Surround with or `Ctrl+Alt+T`) lets you put expressions or statements within blocks or language constructs. This feature in PyCharm applies to:

Context Surround with Example

XML/HTML /XHTML tags

- Tag
- CDATA section
- `<% ... %>`
- Emmet

```
<xsl:when test="//topic[@id=$thisRef]
  <xsl:value-of select="concat(//t
</xsl:when>
<xsl:otherwise
  <xsl:va
</xsl:other
</xsl:choose>
```

Python statements

- if
- while
- try/except
- try/finally

```
def view2(request):
    food = ['beautiful soup', 'steak']
    template = context.Context({
        'f': food
    })
    response = t.render(c)
```

Django templates

- `{% if %}`
- `{% for %}`
- `{% with %}`
- `{% block %}`
- Zen coding

```
<h1>Header</h1>
{{ choice.choice |}}
```

JavaScript statements

- `(expr)`
- `!(expr)`
- if
- if / else
- while
- do / while
- for
- try / catch
- try / finally
- try / catch / finally
- with
- function
- { }
- function expression

```
function Sum(value1, value2) {
    return value1 + value2;
}
if ( ) {
    return value1 + value2;
} else {
}
```

Custom folding region comments

Any fragment of code, where Surround With is applicable

```
math = {
  root:
  square:
  cube:
  | re
}
};
race = fur
var ru
winner
return
};
if (typeof
var _j
_result
```

Surrounding blocks of code

To surround a block of code

1. Select the desired code fragment.
2. Do one of the following:
 - On the main menu, choose Code | Surround With
 - Press `Ctrl+Alt+T`.A pop-up window displays the list of enclosing statements according to the context.
3. Select the desired surround statement from the list. To do that, use the mouse cursor, up and down arrow keys, or a shortcut key displayed next to each element of the list.

Unwrapping and Removing Statements

PyCharm enables you to quickly unwrap or extract expressions from the enclosing statements. This action is available for:

- Python
- JavaScript: [if...else](#), [for](#), [while](#), and [do...while](#) control structures.
- XML and HTML.

To unwrap or remove a statement

1. Place the caret on the expression you want to extract or unwrap.
2. Choose Code | Unwrap/Remove on the main menu or press `Ctrl+Shift+Delete`. PyCharm shows a pop-up window with all the actions that are available in the current context. Statements to be extracted are displayed on the blue background, statements to be removed are displayed on the grey background.

```
if pet == "cats":
    print("You chose cats.")
elif pet == "dogs":
    print("You chose dogs.")
else:
    print("You chose other choices.")
input()
```

3. Click the desired action or select it using the up and down arrow keys and press `Enter`.

Wrapping a Tag. Example of Applying Surround Live Templates

As an example of applying a surround template, let's wrap a piece of XML code with tags.

To surround a code fragment

1. [Open](#) the desired file for editing.

2. Select a code fragment.

```
<company>
  <name>Peter</name>
  <name>John</name>
  <name>Sarah</name>
</company>
```

3. Press invocation shortcut `Ctrl+Alt+J`. PyCharm suggests the following surround templates:

```
<company>
  <name>Peter</name>
  <name>John</name>
  <name>Sarah</name>
</company>
Select Template
T Surround with <tag></tag>
```

4. Select the tag template from the suggestion list. The code fragment is surrounded with empty tags:

```
<company>
  < >
  <name>Peter</name>
  <name>John</name>
  <name>Sarah</name>
  </ >
</company>
```

5. The caret rests within the opening one. On typing the tag name in the opening tag, the name is automatically reproduced in the closing tag:

```
<company>
  <staff>
  <name>Peter</name>
  <name>John</name>
  <name>Sarah</name>
  </staff>
</company>
```

Auto-Completing Code

This section covers various techniques of context-aware code completion that allow you to speed up the coding process:

Basic code completion. Completing names and keywords

Basic code completion helps you complete names of classes, methods, and keywords within the visibility scope. When you invoke code completion, PyCharm analyses the context and suggests the choices that are reachable from the current caret position.

Code completion covers supported and custom file types. However, PyCharm does not recognize the structure of custom file types and suggests completion options regardless of whether a specific type is appropriate in the current context.

If basic code completion is applied to part of a parameter, or a variable declaration, PyCharm suggests a list of possible names depending on the item type.

Invoking Basic code completion for the second time shows the names of classes, functions, modules and variables.

Note Live templates also appear in the basic completion suggestions list.

To use basic code completion:

1. Start typing a name.
2. Press `Ctrl+Space` or choose Code | Completion | Basic from the main menu.

The images below show basic code completion for the following cases:

– Methods:

```
class Poll(models.Model):
    question = models.fie_
```

```
class Poll(models.Model):
    question = models.fdn_
```

– Dictionaries:

```
def bar():
    tasks = {'completed': u'0', 'name': u'The New Motion services sheet',
            tasks []
```

– Django templates:

```
{% filter force_escape | u %}
```

```
{% endfilter %}
```

3. If necessary, press `Ctrl+Space` for the second time (this action produces the same effect as pressing `Ctrl+Alt+Space`).

This shows the names of classes, functions, modules and variables.

First Ctrl+Space	Second Ctrl+Space
<pre>def x(self): s = sock</pre>	<pre>def x(self): s = sock</pre>

Tip You can configure PyCharm to automatically invoke the suggestions list, without having to call basic completion explicitly. To do this, in the main menu select File | Settings (or press `Ctrl+Alt+S`), on the left choose Editor | General | Code Completion, and select the Autopopup code completion option.

You can also select the Insert selected variant by typing dot, space, ect. option to use some keys to accept completion. These keys depend on the language, your context, etc.

Note that while this setting helps you save time, turning it on may result in items being inserted accidentally.

Smart code completion. Completing code based on type information

Smart code completion filters the suggestions list and shows only the types applicable to the current context.

PyCharm supports Smart Type code completion for JavaScript code.

To use smart code completion:

1. Start typing. Press `Ctrl+Shift+Space` or choose Code | Completion | SmartType from the main menu.

SmartType code completion automatically highlights the selection in the suggestions list that is most suitable for the current context.

2. If necessary, press `Ctrl+Shift+Space` once again.

Completing statements

The Smart Enter completion enables you to create syntactically correct code constructs.

To automatically complete a statement, start typing it and press `Ctrl+Shift+Enter`. The punctuation required in the current context is added and the caret moves to the next editing position.

– Completing a method declaration: start typing a method declaration and press `Ctrl+Shift+Enter` after the opening parenthesis:

```
def get_info(self):  
    |  
    ' expected  
    indent expected
```

This will create an entire construct of a method, the caret resting inside the method body:

```
def get_info(self):  
    |  
After pressing Ctrl+Shift+Enter
```

Tip PyCharm automatically completes a method declaration with the mandatory parameter `self`. Start typing a method declaration in a Python class, and PyCharm will insert `self` after the opening bracket of the parameters list:

```
class MyClass:  
    def do_somth(self):
```

This behavior is configurable in the [Smart Keys](#) page of the editor settings.

– Completing code constructs: start typing a code construct and press `Ctrl+Shift+Enter`:

```
def sample(self):  
    if |
```

PyCharm automatically completes the construct, and the caret is placed at the next editing position:

```
def sample(self):  
    if |
```

– PyCharm automatically encapsulates a method call when you directly type a new method call next to it.

For example, type

```
| "test"
```

and then type the method call. When `println` gets the focus in the suggestion list, select it with `Ctrl+Shift+Enter`:

```
System.out.| "test"  
m println () void  
m print (boolean b) void  
m print (char c) void  
m print (char[] s) void
```

The resulting code will look like the following:

```
System.out.println("test");
```

Hippie completion. Expanding words based on context

Hippie completion is a completion engine that analyses your text in the visible scope and draws its completion proposals from the current context. It helps you complete a word with a keyword, class name, method or variable.

To expand a string at caret to an existing word, do the following:

1. Type the initial string and do one of the following:

- Press `Alt+Slash` or choose Code | Completion | Cyclic Expand Word to search for matching words before the caret
- Press `Shift+Alt+Slash` or choose Code | Completion | Cyclic Expand Word (Backward) to search for matching words after the caret and in other open files.

The first suggested value appears, and the prototype is highlighted in the source code:

2. Press `Enter` to accept the suggestion, or hold the `Alt` key and keep pressing `Slash` until the desired word is found.

```
root1 = (-b**2 + math.sqrt(b**2 - 4*a*c)) / (2*a)  
root2 = (-b**2 - math.sqrt(b**2 - 4*a*c)) / (2*a)  
root1
```

Postfix code completion

Postfix code completion helps you reduce backward caret jumps as you write code. It allows you to transform an already typed expression to a different one based on a postfix you type after a dot, the type of expression, and its context.

For example, the `.if` postfix applied to an expression wraps it with an `if` statement.

BeforeAfter

```
function m(arg) {
  arg.if
}
```

```
function m(arg) {
  if (arg) {
  }
}
```

To enable/disable the postfix completion feature or separate templates, in the [Settings / Preferences Dialog](#) dialog, go to Editor | General | Postfix completion. You can also choose which key you want to use to expand postfix templates: `Tab`, `Space`, or `Enter`.

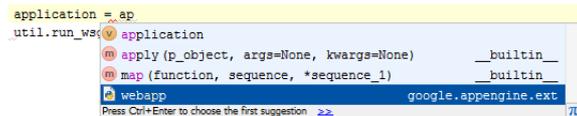
Postfix completion suggestions are shown as part of the basic completion suggestions list. To see a full list of postfix completions applicable in the current context, press `Ctrl+J`.

Completion tips and tricks

Narrow down the suggestions list

You can narrow down the suggestions list by typing any part of a word (even characters from somewhere in the middle), or invoking code completion after a dot separator. PyCharm will show suggestions that include the characters you've entered in any positions.

This makes the use of wildcards unnecessary:



The screenshot shows a code editor with the following code: `application = ap` and `util.run_w3(application`. A completion list is shown with the following items: `application`, `apply(p_object, args=None, kwargs=None)` (builtin), `map(function, sequence, *sequence_1)` (builtin), and `webapp` (google.appengine.ext). A tooltip for `webapp` is visible, and a message at the bottom says "Press Ctrl+Enter to choose the first suggestion >>".

In case of `CamelCase` or `snake_case` names, type the initial letters only. PyCharm automatically recognizes and matches the initial letters.

Accept a suggestion

You can accept a suggestion from the list in one of the following ways:

- Press `Enter` or double-click the desired choice to insert completion to the left from the caret.
- Press `Tab` to replace the characters to the right from the caret.
- Use `Ctrl+Shift+Enter` to make the current code construct syntactically correct (balance parentheses, add missing braces and semicolons, etc.)

View reference information

– You can use the [Quick Definition View](#) by pressing `Ctrl+Shift+I` when you select an entry in the suggestions list:



– You can use the [Quick Information View](#) by pressing `Ctrl+Q` when you select an entry in the suggestions list:



Sort entries in the suggestions list

You can sort the suggestions list alphabetically or by relevance. To toggle between these modes, click **A** or **π** respectively in the lower-right corner of the list.

Note The sorting icons only appear if the list is long and are not displayed for lists containing just a few entries.

PyCharm will remember your choice. You can change the default behavior in the [Code Completion settings page](#).

View code hierarchy

You can view code hierarchy when you've selected an entry from the suggestions list:

- Press `Ctrl+H` to view [type hierarchy](#)
- Press `Ctrl+Shift+H` to view [method hierarchy](#)

Analyzing Applications

In this part:

- [Viewing Structure and Hierarchy of the Source Code](#)
- [Analyzing Duplicates](#)
- [Analyzing External Stacktraces](#)
- [Viewing PSI Structure](#)

Viewing Structure and Hierarchy of the Source Code

PyCharm enables you to examine the hierarchy of classes, methods, and calls in the Hierarchy tool window, and explore the structure of source files in the Structure tool window.

- Both the Hierarchy and the Structure tool windows are available from the View menu.
- the Hierarchy tool window only becomes available, when a hierarchy is built.
- Hierarchies are built in the Navigate menu.

In this part:

- [Building Class Hierarchy](#)
- [Retaining Hierarchy Tabs](#)
- [Viewing Hierarchies](#)
- [Viewing Structure of a Source File](#)

Building Class Hierarchy

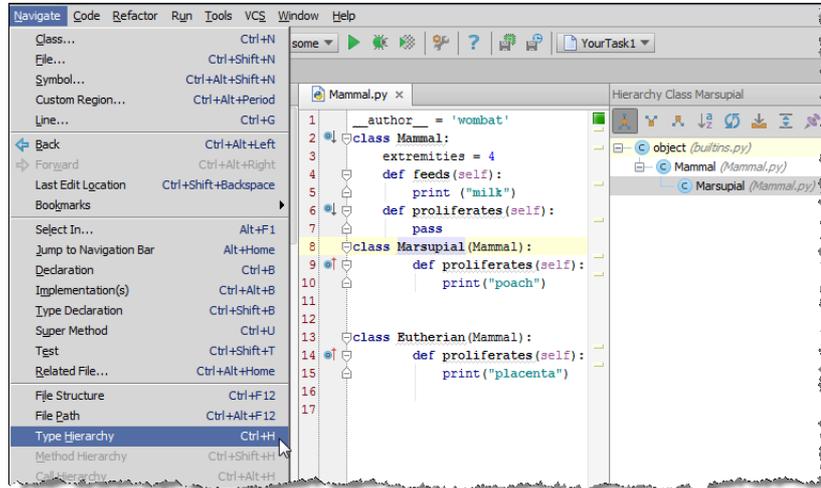
You can explore the hierarchy of parent and children classes of a selected class in the [Hierarchy tool window](#).

Note that the Hierarchy tool window only becomes available, when you build the class hierarchy, as described below.

To build the hierarchy of classes

1. Open file with the desired class in the editor, and place the caret anywhere inside the class, or on the class name.
2. On the main menu, choose Navigate | Type Hierarchy or press `Ctrl+H`.

The class hierarchy appears in the [Hierarchy tool window](#).



Retaining Hierarchy Tabs

By default, every time you build a new hierarchy, PyCharm overwrites the current tab in the [Hierarchy](#) tool window. You can retain the contents of the desired tabs and have next hierarchies built in new tabs.

To retain a hierarchy tab

- In the [Hierarchy](#) tool window, click the Pin Tab button  on the toolbar.

Viewing Hierarchies

Once [built](#), hierarchies can be brought up for close examination in the [Hierarchy tool window](#).

On this page:

- [Showing the Hierarchy tool window](#)
- [Navigating between the tabs of the Hierarchy tool window](#)

To show the Hierarchy tool window, do one of following

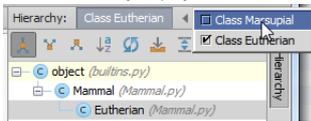
Warning! The Hierarchy tool window is not shown when there are no hierarchies to display. You have to build hierarchies first.

Refer to [Building Class Hierarchy](#) to learn how to build hierarchies.

- On the main menu, choose View | Tool Windows | Hierarchy.
- Use `Alt+8` keyboard shortcut.

To navigate between the tabs of the Hierarchy window, do one of the following

- Right-click the currently displayed tab, and choose Select Next Tab/Select Previous Tab on the context menu.
- Use the `Alt+Right` and `Alt+Left` keyboard shortcuts.
- Click the currently displayed tab, and choose the next tab to display.



Viewing Structure of a Source File

On this page:

- [Basics](#)
- [Viewing the structure of a file](#)
- [Viewing members](#)

Basics

You can examine the structure of the file currently opened in the editor using the Structure tool window or the Structure pop-up window.

By default, PyCharm shows all the classes, methods, etc. presented in the current file.

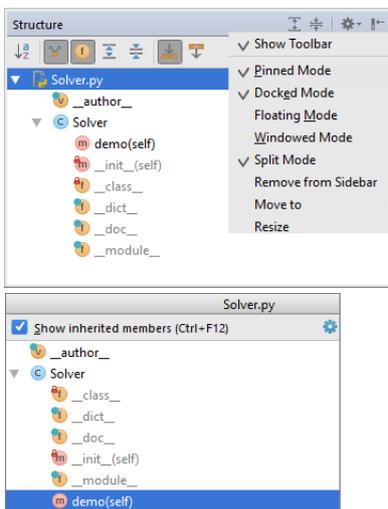
To have [other members displayed](#), click the corresponding buttons on the toolbar of the [Structure](#) tool window.

You can also have class members shown in the [Project](#) tool window.

Viewing the structure of a file

To view the file structure, do one of the following

- On the main menu, choose View | Tool Windows | Structure to show the [Structure tool window](#).
- Press Structure tool button to show the [Structure tool window](#).
- Press `Alt+7` to show the [Structure tool window](#).
- Press `Ctrl+F12` to show the [Structure popup](#).



Viewing members

To have class fields displayed

- Click  on the toolbar of the Structure tool window.

To have inherited members displayed

- Click  on the toolbar of the Structure tool window.

By default, PyCharm shows only methods, constants, and fields defined in the current class. If shown, inherited members are displayed gray.

To have class members shown in the Project tool window

- Turn on the Show members item on the context menu of the Project tool window title bar. If this option is on, the files in the tree that contain classes turn into nodes. When such node is unfolded, the contained classes with their fields, methods, and other members of the selected item are shown.

Analyzing Duplicates

This feature is supported in the Professional edition only.

On this page:

- [Overview](#)
- [Searching for duplicates](#)

Overview

PyCharm helps you find repetitive blocks of code in a certain range. This range can be a single file, a project, a module, or a custom scope. Results of analysis display in the dedicated tab of the [Duplicates tool window](#).

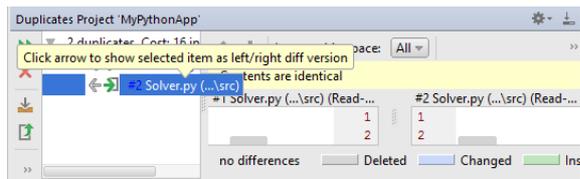
Searching for duplicates

To search for duplicates

1. On the main menu, choose Code | Locate Duplicates.
2. In the [Specify Code Duplication Analysis Scope dialog](#), specify the analysis scope (whole project, current file, uncommitted files (for the projects under version control), or some custom scope). In addition, you can include test sources into the analysis too. Click OK, when ready.
3. In the [Code Duplication Analysis Settings dialog](#), do the following:
 1. Select languages to perform analysis in.
 2. For each language, check the options to define your preferences for the analysis.
For example, you can opt to request identical match for code fragments to be considered duplicates, or specify a certain limit below which the code constructs are not considered duplicates (to avoid reporting about each `if` construct in the source code).

Click OK.

4. In the [Duplicates tool window](#), explore search results.



- View the list of duplicates in the left pane of the tool window.
- View differences between the found duplicates in the right pane. Use the arrow buttons to place the selected duplicate in one of the sections of the differences viewer and compare fragments of the code.
- Navigate to the duplicates in the editor, using Jump to Source or Show Source commands of the duplicates context menu.

Analyzing External Stacktraces

On this page:

- [Overview](#)
- [Analyzing external stacktrace](#)

Overview

You might want to analyze exceptions received by someone else, for example, QA engineers, or investigate a deadlock, or a hang-problem. Unlike the exceptions that you get in the debug mode or when running unit tests, these exceptions do not have links that help you navigate to the corresponding locations in the source code.

With PyCharm, you can simply copy an exception or full thread dump, paste it to the Stacktrace Analyzer, explore information, and navigate to the corresponding source code.

Analyzing external stacktrace

To analyze an external stack trace or thread dump

1. On the main menu, choose Tools | Analyze Stacktrace.
2. In the Analyze Stacktrace dialog box that opens, paste the external stack trace or thread dump into the Put a thread dump here: text area.
3. Click OK. The stacktrace is displayed in the [Run](#) tool window.

Viewing PSI Structure

If you want to explore the internal structure of source code as it is interpreted by PyCharm, use the PSI viewer.

Important note

PSI viewer command is only available when there is at least one plugin module in project.

If you want this information to be available for any project, add the following line to the file `bin/idea.properties` under the product installation:

```
idea.is.internal=true
```

To view PSI structure of the source code

1. On the main menu, choose Tools | View PSI Structure.
2. In the [PSI Viewer](#) dialog box, type or paste the fragment of source code to be analyzed in the Text area, select file type, and specify the other options.
3. Click Build PSI Tree, and preview the generated PSI tree in the PSI Structure pane.

If the source code in the Text area has been changed, click the same button to refresh preview.

Code Inspection

In this section:

- Code Inspection
 - [Code analysis basics](#)
 - [Inspection profiles](#)
 - [Synchronizing profiles between computers](#)
 - [Inspection severity](#)
 - [Inspection scope](#)
 - [Examples of code inspections](#)
 - [Locating dead code](#)
 - [Highlighting unused declarations](#)
 - [Unresolved JavaScript function or method](#)
 - [Python inspections](#)
- [Accessing Inspection Settings](#)
- [Customizing Profiles](#)
- [Changing the Order of Scopes](#)
- [Configuring Inspection Severities](#)
- [Disabling and Enabling Inspections](#)
- [Exporting Inspection Results](#)
- [Resolving Problems](#)
- [Running Inspections](#)
- [Running Inspection by Name](#)
- [Running Inspections Offline](#)
- [Analyzing Inspection Results](#)
- [Viewing Offline Inspections Results](#)
- [Suppressing Inspections](#)
- [Changing Highlighting Level for the Current File](#)

Code analysis basics

PyCharm features robust, fast, and flexible static code analysis. It detects compiler and runtime errors, suggests corrections and improvements before you even compile.

PyCharm performs code analysis by applying inspections to your code. Numerous code inspections exist for Python and for the other supported languages.

The inspections detect not only compiling errors, but also different code inefficiencies. Whenever you have some unreachable code, unused code, non-localized string, unresolved method, memory leaks or even spelling problems – you'll find it very quickly.

PyCharm's code analysis is flexibly configurable. You can [enable/disable](#) each code inspection and [change its severity](#), create [profiles](#) with custom sets of inspections, apply inspections differently [in different scopes](#), [suppress](#) inspections in specific pieces of code, and more.

The analysis can be performed in several ways:

- By default, PyCharm analyses all open files and highlights all detected code issues right in the editor. On the right side of the editor you can see the analysis status of the whole file (the icon in the top-right corner).
When an error is detected, this icon is ; in case of a warning, it is ; if everything is correct, the icon is .
- Alternatively, you can run code analysis in a [bulk mode](#) for the specified scope, which can be as large as the whole project.
- If necessary, you can [apply a single code inspection](#) in a specific scope.

For the majority of the detected code issues, PyCharm provides [quick fix suggestions](#). You can quickly review errors in a file by navigating from one highlighted line to another by pressing  .

For more information and procedural descriptions, see [Configuring Inspection Severities](#).

Inspection profiles

When you inspect your code, you can tell PyCharm which types of problems you would like to search for and get reports about. Such configurations can be preserved as inspection profiles.

An inspection profile defines the types of problems to be sought for, i.e. which code inspections are [enabled/disabled](#) and the [severity](#) of these inspections. Profiles are configurable in the [Inspections](#) settings page.

To set the current inspection profile (the one that is used for the on-the-fly code analysis in the editor), simply select it in the [Inspections](#) settings page and apply changes. When you [perform code analysis](#) or [execute a single inspection](#), you can specify which profile to use for each run.

Inspection profiles can be applicable for the entire IDE or for a specific project:

- Project profiles are shared and accessible for the team members via VCS. They are stored in the project directory:

```
<project>/ .idea/inspectionProfiles .
```

- IDE profiles are intended for personal use only and are stored locally in XML files under the `USER_HOME/.<PyCharm version>/config/inspection` directory.

PyCharm comes with the following pre-defined inspection profiles:

- Default: This local (IDE level) profile is intended for personal use, applies to all projects, and is stored locally in the `Default.xml` file under the `USER_HOME/.<PyCharm version>/config/inspection` directory.
- Project Default: when a new project is created, the Project Default profile is copied from the [settings of a template project](#). This profile is shared and applies to the current project.

After a project is created, any modifications to the project default profile will pass unnoticed to any other projects.

When the settings of the Project Default profile are modified in the [Template Project settings](#), the changed profile will apply to all newly created projects, but the existing projects will not be affected as they already have a copy of this profile.

Project Default profile is stored in the `Project_Default.xml` file located in the `<project>/.idea/inspectionProfiles` directory.

One can have as many inspection profiles as required. There are two ways of creating new profiles: you can [add a new profile](#) as a copy of the Project Default profile or [copy](#) the currently selected profile. The newly created profiles are stored in XML files, located depending on the [type of the base profile](#).

The `<profile_name>.xml` files representing inspection profiles appear whenever some changes to the profiles are done and applied. The files only store differences against the default profile.

Refer to the section [Customizing Profiles](#) for details.

Synchronizing profiles between computers

If an inspection profile is made project-specific, it is synchronized with your project automatically. Each user, who opens this project after checking it out, will have the same inspection profile enabled.

If an IDE Default inspection profile is used, it can be synchronized between multiple machines via the [Settings Repository plugin](#), bundled with PyCharm. So doing, the file is stored in `USER_HOME/.<PyCharm version>/config/inspection/<profile_name>.xml`.

Note that the Global profile may have a different name. Copy your current project profile to the Global Level (click the button Manage and choose Copy as Global) and call it as you like.

Then, on the main menu, choose File | Default Settings and select this global profile as the default one for all the new projects. Now all the newly-created projects will use this global profile by default and this global profile will be synchronized between the various machines via the [Settings Repository plugin](#).

Inspection severity

Inspection severity indicates how seriously the code issues detected by the inspection impact the project and determines how the detected issues should be highlighted in the editor. By default, each inspection has one of the following severity levels:

- Server problem 
- Typo 
- Info 
- Weak Warning 
- Warning 
- Error 

You can increase or decrease the severity level of each inspection. That is, you can force PyCharm to display some warnings as errors or weak warnings. In a similar way, what is initially considered a weak warning can be displayed as a warning or error, or just as info.

You can also configure the color and font style used to highlight each severity level. Besides, you can create custom severity levels and set them for specific inspections.

If necessary, you can set different severity levels for the same inspection [in different scopes](#).

All modifications to inspections mentioned above are saved in the [inspection profile](#) currently selected in the [inspection settings](#) and apply when this profile is used.

Inspection scope

By default, all enabled code inspections apply to all project files. If necessary, you can configure each code inspection ([enable/disable](#), [change its severity level](#) and options) individually for different [scopes](#). Such configurations, like any other inspection settings, are saved and applied as part of a specific [profile](#).

There may be complicated cases when an inspection has different configurations associated with different scopes. When such inspection is executed in a file belonging to some or all of these scopes, the settings of the highest priority scope-specific configuration are applied. The priorities are defined by the relative position of the inspection's scope-specific configuration in [inspection settings](#): the uppermost configuration has the highest priority. The Everywhere else configuration always has the lowest priority.

For more information and procedural descriptions, see [Configuring Inspection for Different Scopes](#).

Examples of code inspections

In the [Inspections](#) page, all inspections are grouped into categories. The most common tasks covered by code analysis are:

- Finding probable bugs.
- Locating dead code.
- Detecting performance issues.

- Improving code structure and maintainability.
- Conforming to coding guidelines and standards.
- Conforming to specifications.

Locating dead code

PyCharm highlights in the editor pieces of so-called *dead* code. This is the code that is never executed during the application runtime. Perhaps, you don't even need this part of code in your project. Depending on situation, such code may be treated as a bug or as a redundancy. Anyway it decreases the application performance and complicates the maintenance process. Here is an example.

So-called *constant conditions* - conditions that are never met or are always *true*, for example. In this case the responsible code is not reachable and actually is a *dead* code.

```

attribute = parseAttribute(isempty, asp, php);

if (attribute == null) {
    ...
    return;
}
value = parseValue(attribute, false, isempty, delim);

if (attribute != null) {
    ... Condition 'attribute != null' is always true.
}
else {
    sv = new AttVal( null, null, null, null,
                   0, attribute, value );
    Report.attrError(this, this.token, value,
                    Report.BAD_ATTRIBUTE_VALUE);
}

```

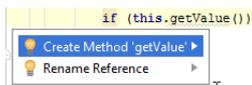
PyCharm highlights the if condition as it's always true. So the part of code surrounded with else is actually a dead code because it is never executed.

Highlighting unused declarations

PyCharm is also able to instantly highlight Java classes, methods and fields which are unused across the entire project via Unused declarations inspection. All sorts of Java EE `@Inject` annotations, test code entry points and other implicit dependencies configured in the Unused declarations inspection are deeply respected.

Unresolved JavaScript function or method

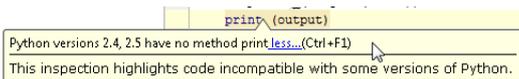
This inspection detects references to undefined JavaScript functions or methods.



Python inspections

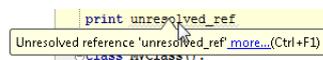
Find descriptions and examples of the various code inspections in the [Inspections](#) page of the Settings dialog. In this section, we'll consider several most prominent examples. In particular, PyCharm reports

- Code incompatible with the current Python interpreter version:

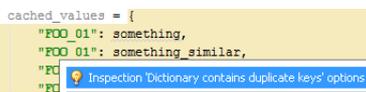


Note that you can change the scope of versions in the [inspection settings](#).

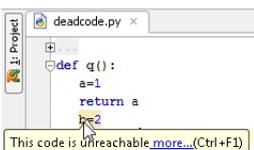
- Unresolved references:



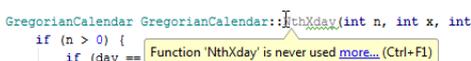
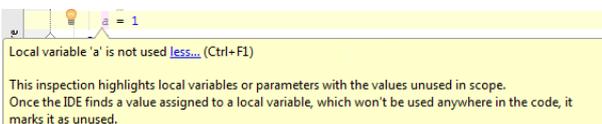
- Duplicate keys in dictionaries:



- Dead code:



- Unused variables :



Accessing Inspection Settings

Inspections and [inspection profiles](#) are editable in the [Inspections](#) settings page. PyCharm provides several ways to gain access to inspection settings.

To access inspections and profiles settings, do one of the following

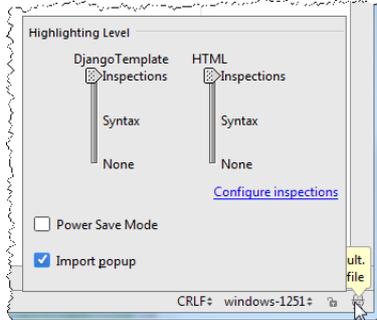
- Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Inspections under Editor.

- On the main toolbar, click , then expand the Editor node, and click [Inspections](#).

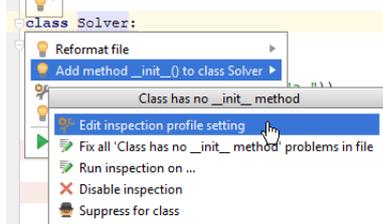
- Click the current profile icon in the Status bar



and then click the Configure inspection link.



- In the editor, open the suggestion list, click the right arrow, and choose Edit inspection profile settings on the submenu.



- In the [Inspection Results Tool Window](#), click Edit Settings  on the toolbar or use the corresponding context menu command.

Customizing Profiles

- [Introduction](#)
- [Customizing profiles](#)
- [Managing profiles](#)

Introduction

PyCharm lets you configure settings for your code validation analysis and save them as inspection profiles. You can customize the existing inspection profiles (including default profiles), and create new ones. You can also share, import and export inspection profiles.

PyCharm distinguishes between IDE and project profiles.

Stored in IDE

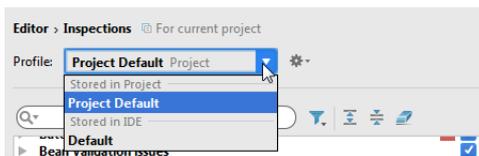
These profiles are saved in an application config directory (for example, `~/PyCharmXXX/config/inspection` on Linux) and are available for any project.

Stored in Project

These profiles are saved in a particular project's `.idea` directory (for example, `$PROJECT_DIR/.idea/inspectionProfiles` on Linux).

Customizing profiles

1. In the [inspection settings](#), select the profile to be changed.



Note that the default profiles are also editable.

2. Customize the desired inspections: [enable or disable](#), [change their severity](#) and the other options, which can be different for the various inspections.
Note that the selectors of severity and scopes, and the options section (if any) are only available for the enabled inspections.
3. To apply an inspection to a restricted set of files, [associate it with the corresponding scopes](#).
By default, all inspections apply to all the sources of the current project.
4. Apply changes.

Managing profiles

1. In the [inspection settings](#), select the profile you want to manage.
2. Click  and from the drop-down list, select one of the following options:
 - Copy to IDE or Copy to Project - to copy the selected profile to either IDE level or the project level.
 - Duplicate - to make a copy of the selected profile. You can change the name of the profile in the Profile field.
 - Rename - to rename the selected profile.
 - Delete - to delete the selected profile.
 - Restore Defaults - to change the selected profile back to its default settings.
 - Add Description or Edit Description - to add a new or edit an existing description for the selected profile.
 - Export - to export the selected profile in a form of XML file. You can select where to export your profile.
 - Import Profile - to import the desired profile (XML file). You can select where to import your profile.
3. Apply changes.

Configuring Inspection for Different Scopes

On this page:

- [Basics](#)
- [Defining the order of scopes](#)

Basics

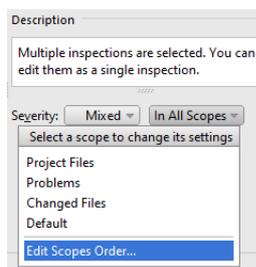
By default, all enabled code inspections apply to all project files. If necessary, you can configure each code inspection ([enable/disable](#), [change its severity level](#) and options) individually for different [scopes](#). Such configurations, like any other inspection settings, are saved and applied as part of a specific [profile](#).

There may be complicated cases when an inspection has different configurations associated with different scopes. When such inspection is executed in a file belonging to some or all of these scopes, the settings of the highest priority scope-specific configuration are applied. The priorities are defined by the relative position of the inspection's scope-specific configuration in [inspection settings](#): the uppermost configuration has the highest priority. The Everywhere else configuration always has the lowest priority.

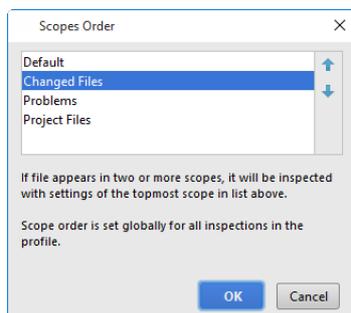
Defining the order of scopes

To define the order of scopes, follow these steps

1. In the [Inspections](#) page of the Settings/Preferences dialog, click the button In All Scopes:



2. Choose Edit Scopes Order... from the scopes drop-down list.
3. In the Scopes Order dialog box that opens, select the desired scope, and click the up and down arrows .



Configuring Inspection Severities

On this page:

- [Basics](#)
- [Changing severity of an inspection](#)
- [Changing severity of an inspection for different scopes](#)
- [Changing the highlighting style for a specific severity level](#)
- [Defining a custom severity level](#)

Basics

Inspection severity indicates how seriously the code issues detected by the inspection impact the project and determines how the detected issues should be highlighted in the editor. By default, each inspection has one of the following severity levels:

- Server problem 
- Typo 
- Info 
- Weak Warning 
- Warning 
- Error 

You can increase or decrease the severity level of each inspection. That is, you can force PyCharm to display some warnings as errors or weak warnings. In a similar way, what is initially considered a weak warning can be displayed as a warning or error, or just as info.

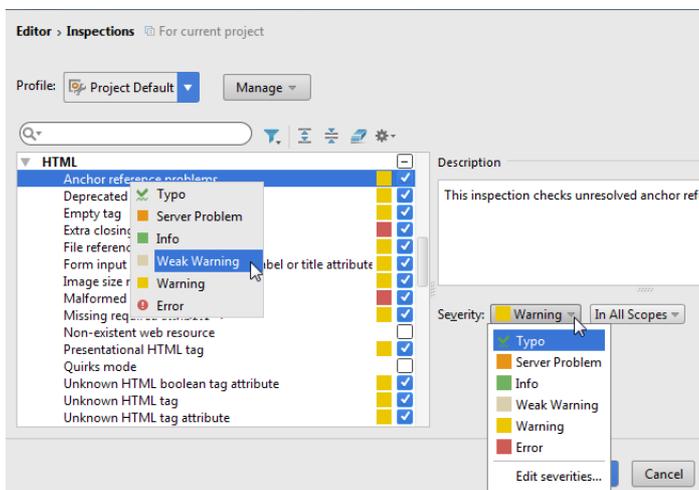
You can also configure the color and font style used to highlight each severity level. Besides, you can create custom severity levels and set them for specific inspections.

If necessary, you can set different severity levels for the same inspection [in different scopes](#).

All modifications to inspections mentioned above are saved in the [inspection profile](#) currently selected in the [inspection settings](#) and apply when this profile is used.

Changing severity of an inspection

1. In the [inspection settings](#), select the desired [profile](#). The inspections associated with the profile are displayed in the tree view.
2. Select the desired inspection. If this inspection is disabled, select the check box next to it.
3. Select the desired severity from the context menu of the inspection or from the Severity selector on the right:

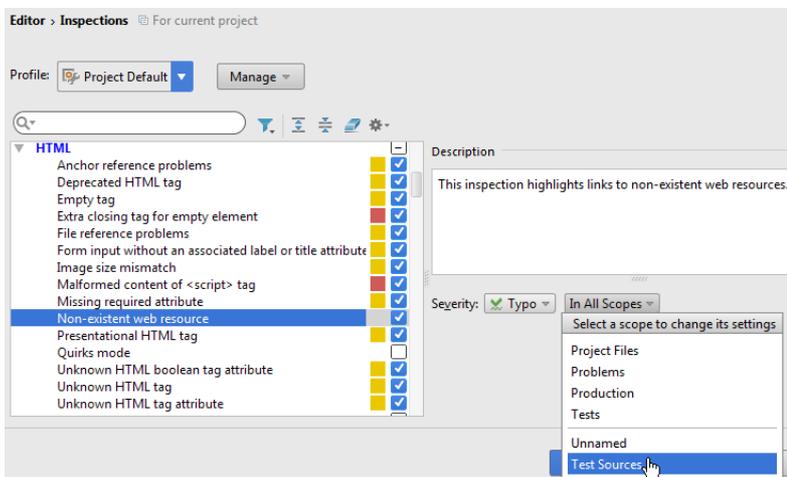


Note that inspections, whose state is changed relative to the default values, and all their grouping nodes are highlighted with blue.

4. Apply the changes. The modified inspection will now have the new severity level when this [profile](#) is used.

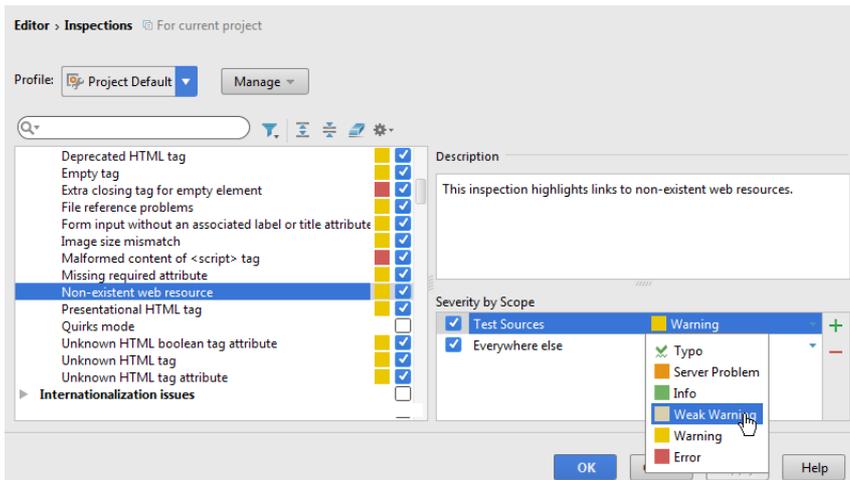
Changing severity of an inspection for different scopes

1. Choose the desired profile and inspection.
2. Click the drop-down list [In All Scopes](#), and select the scope you want to change inspection severity in:



PyCharm shows severities for two scopes: for the selected one and Everywhere else

3. Click severity drop-down list for the selected scope and choose the appropriate severity level from the drop-down list:



Changing the highlighting style for a specific severity level

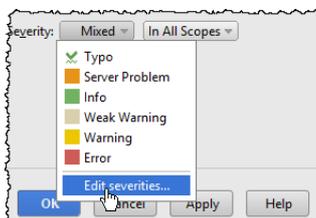
1. Do one of the following:
 - In the [Settings/Preferences](#) dialog, select Editor | Colors & Fonts -> General, and then select the style corresponding to the desired severity level.
 - In the [inspection settings](#), select the desired inspection and choose Edit severities from the Severity selector. Next, in the Severities Editor dialog box that opens, select the desired severity level and click Editor | Colors & Fonts.

Either way you will see the styles associated with this severity in the [Colors and Fonts](#) settings page.

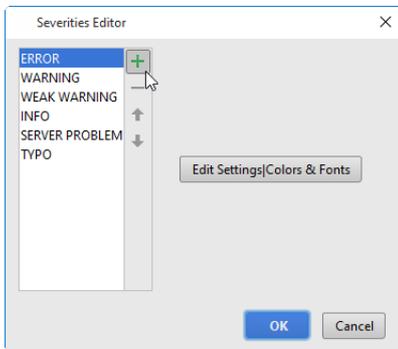
2. Configure the color and font styles as necessary and apply changes. The detected issues with the corresponding severity will now be highlighted in the editor with the modified style when the current [profile](#) is used.

Defining a custom severity level

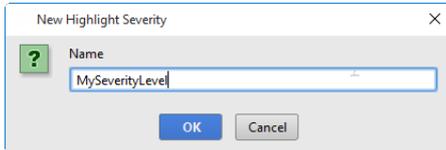
1. In the [inspection settings](#), select the desired inspection and choose Edit severities from the Severity selector.



2. In the Severities Editor dialog box that opens, click +:



3. Type the name for the new severity in the New Highlight Severity dialog box.



The custom severity is added to the list of severities.

4. Specify color and font settings for the new severity using the controls to the right of the list of severities.
5. Use the Up ↑ and Down ↓ buttons to change the priority of the new severity.
6. Apply the changes. The new severity level will now be available for all inspections within the current [profile](#). You can [assign](#) it to specific inspections and get the corresponding code issues highlighted with the specified style in the editor.
If necessary, you can remove the custom severity level later by selecting it in the Severities Editor dialog box and clicking [-](#).

You cannot change priorities of the pre-defined severity levels, or remove them.

Disabling and Enabling Inspections

On this page:

- [Introduction](#)
- [Disabling or enabling inspections](#)

Introduction

If you think that some inspections report about the problems that you are not interested in, you can disable such inspections. Note that when you disable an inspection, it is disabled in the current [inspection profile](#); in all other profiles, it remains enabled.

There are several ways to disable/enable inspections:

- [Using the Inspections page in the Settings/Preferences dialog](#) - this is the main interface for managing inspections; here you can see at once, which inspections are enabled or disabled in all inspection profiles.
- [Using the intention actions](#) - this is the way to disable a highlighted code issues right in the editor.
- [In the Inspection Results tool window](#) - this is a quick way to disable uninteresting issues when [analyzing inspection results](#). Note that here you can only disable inspections.

Note Note the difference between disabling and [suppressing](#) code inspections:

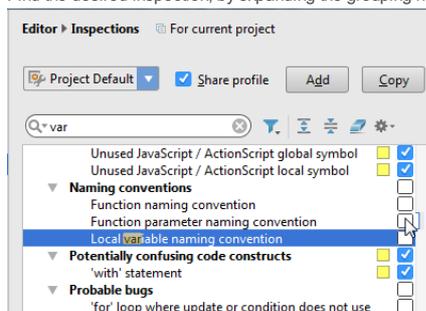
When [suppressing](#) an inspection, PyCharm inserts a special comment that tells the code analysis engine to ignore the issues found by this inspection in the specific piece of code.

When [disabling](#) an inspection, you just turn it off so the code analysis engine just ignores the code issues found by this inspection.

Disabling or enabling inspections

To disable or enable an inspection in the Settings/Preferences dialog

1. Find the desired inspection, by expanding the grouping nodes or using the search field.



2. Use the check box next to the inspection to disable or enable it.
3. Apply the changes and close the dialog box.

To disable an inspection for highlighted issue in the editor

- When you disable inspections this way, they are disabled for the current [inspection profile](#).
- To re-enable inspections disabled this way, use the [main procedure](#) described above.

1. Set the caret at a highlighted issue.
2. Click the bulb icon or press `Alt+Enter` to reveal the inspection alert and suggestion list.
3. Select the inspection to be disabled, then click right arrow button or just press the right arrow key.
4. On the submenu, click Disable <inspection name>.

To disable inspections from the Inspection results report

- When you disable inspections this way, they are disabled for the [inspection profile](#) that was used for [running inspections](#). You can see it in the header of the Inspection Results window's tab.
 - To re-enable inspections disabled this way, use the [main procedure](#) described above.
1. In the [Inspection Results Tool Window](#), right-click the inspection you want to disable.
 2. On the context menu, choose Disable inspection.
 3. Press the filter button  to hide the disabled inspection alerts.

Exporting Inspection Results

After you [perform code analysis](#) or [execute a single inspection](#), you can save the inspection results for further examination or for sharing with colleagues. PyCharm enables you to export inspection results to the HTML or XML format.

To export inspection results

1. On the toolbar of the [Inspection Results Tool Window](#), click the Export button .
2. From the Export To context menu, select the target format. The available options are HTML and XML.
3. In the [dialog that opens](#), specify the target directory to store the inspection results in.

Resolving Problems

On this page:

- [Introduction](#)
- [Fixing problems](#)

Introduction

By default, PyCharm analyses the code in all open files and highlights the detected code issues. You can fix most of the issues immediately by [applying quick-fixes](#).

If you [perform code analysis](#) or [execute a single inspection](#) in a larger scope of source files, PyCharm displays the detected code issues in the [Inspection Results Tool Window](#). When you select a specific issue in this window, its report is shown in the right part of the window.

If there are available fixes to the issue, PyCharm notifies it in the following ways:

- The Apply a Quickfix  button becomes available on the toolbar of the Inspection Results tool window.
- The available fixes are shown in the optional Problem resolution field of the report.
- The available fixes are shown in the context menu of the issue.

If there are no available fixes, the only option is to fix the issue manually.

Please note the following:

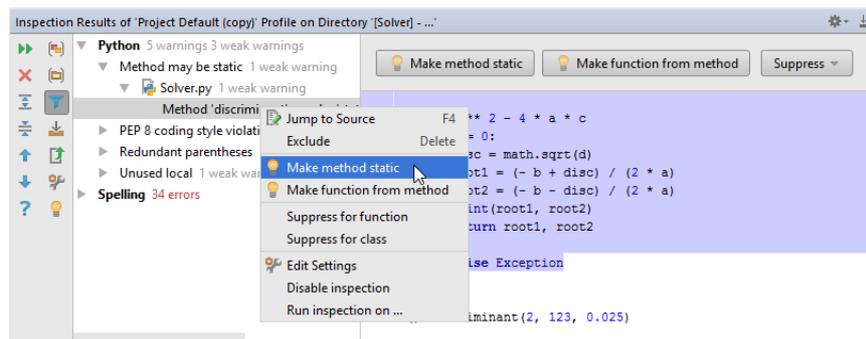
- To display the source code of an issue in the editor, when it is selected in the [Inspection Results Tool Window](#), toggle the Autoscroll to Source  button.
- If you find that some of the reported issues are minor or not helpful to you, you can ignore them either by [disabling](#) the corresponding inspection or by [suppressing](#) it in a specific piece of code.

Fixing problems

To fix a problem reported by code inspection

In the [Inspection Results Tool Window](#), select the code issue you are interested in and do one of the following:

- If PyCharm suggests any fixes to the issue as described above, you can use one of them to fix the problem immediately.



- If no resolutions are suggested, use the Jump to source command in the context menu and fix the problem manually.

Running Inspections

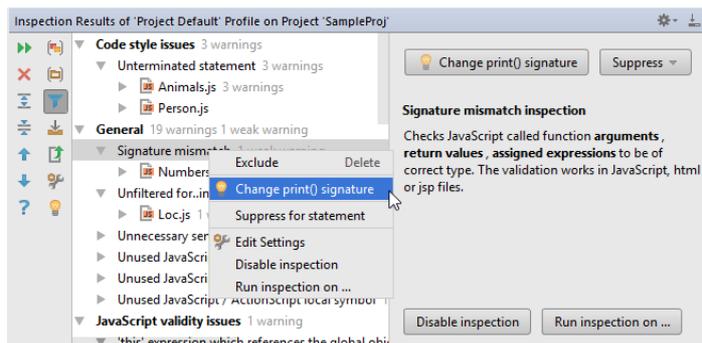
Although code analysis is performed on-the-fly in all open files, you may want to run it for the whole project or a custom scope and examine the results in a friendly view.

Note Inspecting code in a large scope, for example, the whole project, can be time consuming. For large projects, it is recommended to [create](#) a number of smaller [scopes](#) and run analysis in each of them separately.

– If you do not want to have some code issues in the analysis report, you can [disable some inspections](#).

To run a code inspection

1. Open the desired file in the editor. Alternatively select files or directories in the [Project tool window](#). For multiple selection, click the items holding down the [Ctrl](#)/[⌘](#) key. The initial inspection scope will be confined to the opened file or the selection.
2. On the main menu, choose Code | Inspect Code. The [Specify Inspection Scope](#) dialog box opens.
3. In the Inspection scope area, specify which files should be inspected.
 - To have the source code of the entire project inspected, select the Whole Project option.
 - If you are using [version control integration](#), you can choose to only inspect uncommitted files.
 - To run an inspection for the currently opened file, or the file(s)/folder(s) selected in the Project view, select the File/Module <name> option.
 - To apply inspect code in a specific [scope](#), select the Custom scope option, then choose the desired scope from the drop-down list or click the Browse button [\[...\]](#) and configure a new scope in the [Scopes](#) dialog box.
4. To have test source files inspected too, select the Include test sources check box.
5. Specify the [inspection profile](#) to apply. Do one of the following:
 - Select one of the existing profiles from the Inspection Profile drop-down list.
 - Click the Browse button [\[...\]](#) and [configure a new profile](#) in the [Inspections](#) dialog box.
6. Click OK to run code analysis.
7. [Examine results](#) in the [Inspection Results Tool Window](#).



It is also possible to run inspections offline, without starting the IDE. Follow the procedure described in the section [Working with PyCharm Features from Command Line](#).

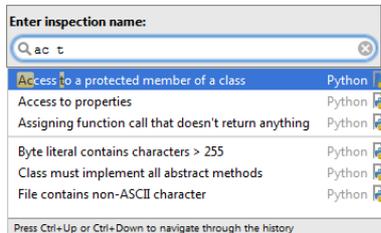
Running Inspection by Name

Rather than running all enabled inspections, PyCharm makes it possible to exactly specify the desired inspection by its name and run it to inspect the code in the specific scope.

To run a code inspection by name

1. Open the desired file in the editor. Alternatively select files or directories in the [Project tool window](#). For multiple selection, click the items holding down the `Ctrl`/`⌘` key. The initial inspection scope will be confined to the opened file or the selection.
2. On the main menu, choose Code | Run Inspection by Name, or press `Ctrl+Shift+Alt+I`.
3. In the pop-up frame that opens, start typing the inspection name. As you type, the suggestion list shrinks to show the matching inspection only.

Tip Use CamelHumps to match camel case words and white spaces to match initial letters of the words.



The [Specify Inspection Scope](#) dialog box opens.

4. In the Inspection scope area, specify which files should be inspected.
 - To have the source code of the entire project inspected, select the Whole Project option.
 - If you are using [version control integration](#), you can choose to only inspect uncommitted files.
 - To run an inspection for the currently opened file, or the file(s)/folder(s) selected in the Project view, select the File/Module <name> option.
 - To apply inspect code in a specific [scope](#), select the Custom scope option, then choose the desired scope from the drop-down list or click the Browse button `⋮` and configure a new scope in the [Scopes](#) dialog box.
5. To have test source files inspected too, select the Include test sources check box.
6. To apply inspection only in files matching the specific mask, select the File Masks(s) and specify the file mask. Use comma to separate multiple file masks.
7. Click OK to run the inspection.
8. [Examine results](#) in the [Inspection Results Tool Window](#).

Running Inspections Offline

On this page:

- [Basics](#)
- [Launching a code inspection from the command line](#)
 - [Examples](#)
 - [Windows](#)
 - [macOS](#)
- [Viewing the results of an offline inspection](#)

Basics

In addition to running code inspections from the main menu, or from the context menus of the [Project tool window](#), you can also launch the inspector from the command line, without actually running PyCharm.

This way you can perform regular code inspections as a part of your development process, which is especially important for large projects. Inspection results are stored in the XML format.

Launching a code inspection from the command line

To launch a code inspection from the command line

- Specify the following command line arguments:
 - Path to the launcher: specify the **full path** to one of the following launchers (which reside under the `bin` directory of your PyCharm installation):
 - For **Windows**: `inspect.bat`
 - For **UNIX** and **macOS**: `inspect.sh`
 - Project file path is the **full path** to the directory that contains the project to be inspected.
 - Inspection profile path is the **full path** to the profile, against which the project should be inspected. The inspection profiles are stored under `USER_HOME\PyCharmXX\config\inspection`
 - Output path is the **full path** to an existing directory where the report will be stored.
 - Options. You can specify:
 - The directory to be inspected `-d <full path to the subdirectory>`
 - The verbosity level of output `-vX`, where X is 0 for quiet, 1 for noisy and 2 for extra noisy.

Warning! Please note that you have to specify full paths. Relative paths are not accepted!

Examples

Windows

```
C:\Program Files (x86)\JetBrains\<PyCharm home>\bin\inspect.bat
E:\SampleProjects\MetersToInchesConverter E:\Work\MyProject\.idea\inspectionProfiles\Project_Default.xml
E:\Work\MyProject\inspection-results-dir -v2 -d E:\SampleProjects\MetersToInchesConverter\subdirectory
```

macOS

```
/Applications/PyCharm.app/Contents/bin/inspect.sh ~/PyCharmProjects/MyTestProject
~/Library/Preferences/pycharmXX/inspection/Default.xml ~/PyCharmProjects/MyTestProject/results-dir -v2
```

Viewing the results of an offline inspection

If you have performed an offline inspection and exported the inspection results to a directory in the XML format you can always download and view these results.

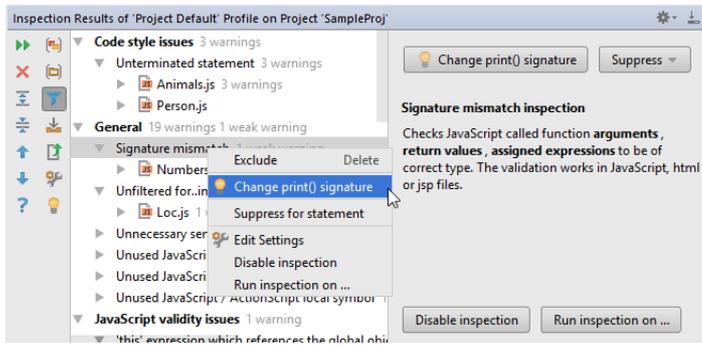
To view the results of an offline inspection, follow these steps

1. Open the project against which the inspection was performed.
2. On the main menu, choose Code | View Offline Inspection Results.
3. In the Select Path dialog box that opens, navigate to the directory that contains inspection results in XML format.
4. Click OK. Inspection results display in the Offline View tab in the [Inspection Results Tool Window](#).

Tip! Alternatively, you can open the relevant XML file in PyCharm or in any other text processor without opening the inspected project.

Analyzing Inspection Results

After you [perform code analysis](#) or [execute a single inspection](#), you can examine the results in the [Inspection Results Tool Window](#).



Each run of the code analysis or a single inspection is displayed in a new tab in the tool window. The left part of the tab displays detected code issues grouped by inspection groups, inspections and files. If necessary, you can change the default grouping. When you select a specific issue, its report appears in the right part.

In the Inspection Results tool window you can:

- [Resolve issues using quick-fixes](#) where available.
- [Jump to the source of the problem](#) to resolve it manually.
- [Export inspection results](#) to an XML or HTML file.
- [Disable inspections](#) to skip all corresponding issues next time you ran code analysis.
- [Suppress inspections](#) for specific code issues.
- [Access inspection settings](#) to configure inspections.

Viewing Offline Inspections Results

If you have exported inspection results to a directory in XML format, or [performed offline inspection](#) outside PyCharm, as a part of the automated build process, you can always download and view the results.

- [Prerequisite](#)
- [Viewing offline inspection results](#)

Prerequisite

The relevant project should be opened in the IDE, and should contain the classes that have been inspected. Otherwise, you have to review inspection results, opening the reports as XML files in the editor.

To view inspection results offline

1. On the main menu, choose Code | View Offline Inspection Results.
2. In the Select Path dialog, navigate to the directory that contains inspection results in XML format.
3. Click OK. Inspection results display in the Offline View tab in the [Inspection Results Tool Window](#).

Suppressing Inspections

On this page:

- [Introduction](#)
- [Suppressing inspections in the editor](#)
- [Suppressing inspections from the Inspection Results tool window](#)

Introduction

For some reasons, you may want to partly disable a specific inspection, i.e. ignore some code issues while continuing to detect the other issues with this inspection.

For example, PyCharm considers some code to be "dead", and you can see that it is true. The inspection is helpful and you do not want to [disable](#) it. However, you may want to use this code later and do not want it to be highlighted in the editor or appear in the issue reports.

To do so, PyCharm allows you to **suppress** certain inspections for a specific statement, function/method, tag or file. You can do it either [in the editor](#), using the suggestion list or [in the Inspection Results tool window](#) when analysing inspection results.

Let's summarize the difference between suppressing and [disabling](#) code inspections:

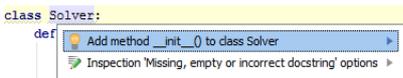
When **suppressing** an inspection, PyCharm inserts a special comment that tells the code analysis engine to ignore the issues found by this inspection in the specific piece of code.

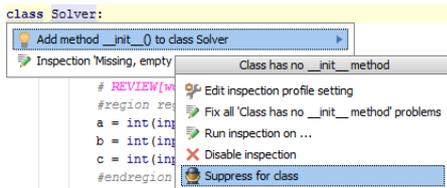
When **disabling** an inspection, you just turn it off so the code analysis engine just ignores the code issues found by this inspection.

Some code inspections (e.g. those detecting errors) cannot be suppressed.

Suppressing inspections in the editor

To suppress an inspection in the editor

1. Set the cursor to the highlighted code issue in the editor.
2. Press `Alt+Enter`, or click the light bulb icon  to expand the suggestion list.

3. Depending on the issue, you will see either quick-fixes related to the inspection or the Inspection "<inspection name>" options item.
4. Use the up/down arrow keys to select this item and then press the right arrow key or just click the right arrow  next to this item. Pressing the left arrow key, or `Escape` hides the suggestion list.
5. In the inspection options list, select the desired suppress action:

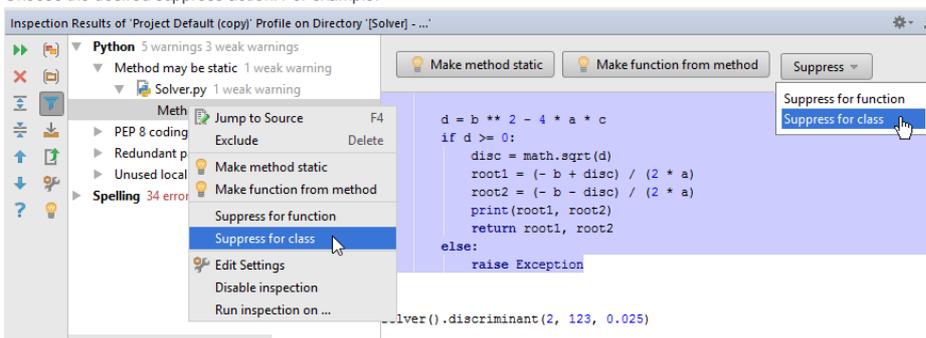


The inspection will be suppressed with special comments in the corresponding piece of code.

Suppressing inspections from the Inspection Results tool window

To suppress inspections from the Inspection Results tool window

1. After [running code analysis](#), select a code issue, for which you want to suppress the inspection, in the [Inspection Results Tool Window](#).
2. Click the button `Suppress` and choose the scope of suspension, or just right-click the selected inspection.
3. Choose the desired suppress action. For example:



The inspection will be suppressed with special comments in the corresponding piece of code.

Changing Highlighting Level for the Current File

Use the [Status bar](#) to quickly re-configure highlighting for the file which is currently opened in the editor. With Hector, you can choose to highlight syntax problems, inspection issues, or turn off highlighting.

To change the highlighting level for the current file

1. Open the Highlighting level pop-up window by doing one of the following:
 - On the main menu, choose Code | Configure Current File Analysis.
 - Press `Ctrl+Shift+Alt+H`.
 - Click the Hector icon  on the Status bar.
 - Right-click the code inspection indicator on top of the scroll bar.
2. Move the slider to one of the three available positions that define the highlighting level:
 - None: turn off problems highlighting in the editor.
 - Syntax: highlight syntax problems only.
 - Inspections: (default) highlight syntax problems and inspection issues.

Intention Actions

In this section:

- Intention Actions
 - [Introduction](#)
 - [Intention action icons](#)
 - [Intention action types](#)
- [Disabling Intention Actions](#)
- [Configuring Intention Actions](#)
- [Applying Intention Actions](#)
- [Examples of Python-Specific Intention Actions](#)

Introduction

PyCharm helps you handle the situations when you use classes that haven't been imported, or methods that haven't been written etc., which can result in errors. When a possible problem is suspected, PyCharm suggests a solution, and in certain cases can implement this solution (properly assign variables, create missing references and more). Besides syntax problems, PyCharm recognizes code constructs that can be optimized or improved, and suggests appropriate intention actions, denoted with the special icons.

Intention action icons

Item icon Description

Intention actions suggested	 A yellow bulb indicates that PyCharm just proposes to alter your code. It covers a range of situations from warning correction to suggestions for code improvement (like micro-refactorings).
Specific intention action	 This sign appears in the suggestion list before each specific intention action. If an intention action alert is disabled, the sign turns to  . Disabled intention action is still available and can be enabled again.
Quickfix suggested	 A red bulb with an exclamation mark indicates that PyCharm suggests a way to fix an error. It is related to Create from usage intentions and Quick fixes.
Disabled	 Alert is disabled, but the intention action is still available and can be enabled again.

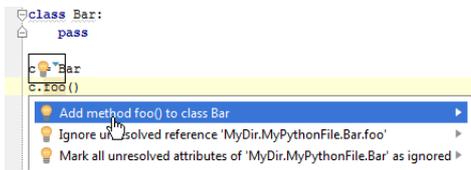
Intention action types

Find descriptions of specific intention actions on the [Intentions page](#) of the editor settings/preferences, where they are grouped according to the areas of their usage. Generally, intention actions can be divided into several categories, for example:

Create from usage

This type of intention action creates new code items: classes, methods, etc. They are smart enough to analyze your code and provide actions suitable for a particular case. The main concept behind this type is that you can begin using new things without declaring them first. You are not taken away from your current task for mundane minutiae like creating declarations, new files, etc. which PyCharm handles while you keep focused.

For example, if you refer to a non-existent symbol, PyCharm suggests you to create one:

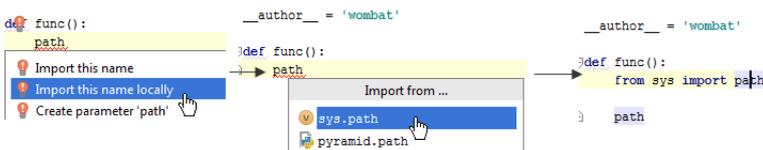


Quick fixes

This type of intention action responds to common coding mistakes: using an improper access modifier, or an expression of the wrong type, or missing resources, etc. PyCharm catches these kinds of problems as you type, and provides a quick way to fix them using Intention Actions with appropriate suggestions for the error.



It is a common practice to delay import of a symbol to the point where it is actually used. For this purpose, PyCharm suggests a dedicated quick fix Import this name locally:



Micro-refactorings

These intention actions appear for code that is syntactically correct, but can be structurally improved by such things as:

- Converting code constructs.
- Splitting declarations and assignments.
- Splitting or merging statements and tags, etc.

Convert lambda to function. A lambda function can be turned into def statements:

Disabling Intention Actions

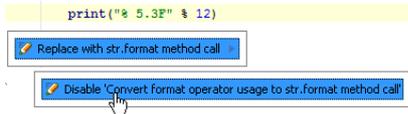
If an intention action is enabled, the alert shows automatically, when the caret rests on the problem-causing piece of code. You can disable alert for any type of intention actions, and show it by explicit invocation only.

Disabled intention actions are marked with the grey light bulb icon . Though disabled, such actions are still available and can be applied to the source code.

You can suppress intention actions related to inspections "on-the-fly", as described in the section [Suppressing Inspections](#).

To disable an intention action alert

1. Click `Alt+Enter`, or click the light bulb icon to show the suggestion list.
2. Select the action to be disabled and click the right arrow button.
3. On the submenu, click `Disable <intention action name>`:



For the disabled intention action, the menu item changes to `Enable <intention action name>`.

Configuring Intention Actions

On this page:

- [Introduction](#)
- [Configuring intention settings using the Settings/Preferences dialog](#)
- [Configuring intentions on-the-fly](#)

Introduction

PyCharm makes it possible to configure intention action settings either in the [Intentions](#) page of the Settings/Preferences dialog, or "on-the-fly".

By default, all available intention actions that ship with PyCharm, are enabled. By changing the Intention settings, you can disable the actions that are not required for your current working environment.

Configuring intention settings using the Settings/Preferences dialog

To configure intention settings using the Settings/Preferences dialog

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Intentions under Editor.
2. In the [Intentions](#) page, clear the check boxes of the intention actions or action categories that you do not currently need. Selecting or clearing a category affects all intention actions in this category.
3. Apply changes and close the dialog.

Configuring intentions on-the-fly

To configure intention settings on-the-fly

1. In the editor, press `Alt+Enter` to reveal the inspection alert and suggestion list.
2. Select the action to be disabled, then click right arrow button or just press the right arrow key.
3. On the submenu, choose Disable <intention action name>.

Applying Intention Actions

PyCharm displays an intention action alert, and it is your task to define which action (if any) should be performed.

To apply an intention action

1. Click the light bulb icon, or use the `Alt+Enter` keyboard shortcut to bring up the suggestion list.
2. Click the desired option, or use the arrow keys to select the option and press `Enter`.

Examples of Python-Specific Intention Actions

In this section:

- [Converting comments](#)
- [Converting formatting style](#)

Converting comments

For comment-based type hints, PyCharm suggests an intention action that allows you to convert comment-based type hint to a variable annotation. This intention has the name Convert to variable annotation, and works as follows:

Warning! This is available for Python 3.6!

BeforeAfter

```
from typing import List, Optional  
  
xs = [] # type: List[Optional[str]]
```

```
from typing import List, Optional  
  
xs: List[Optional[str]] = []
```

Converting formatting style

Warning! This is available for Python 3.6!

PyCharm provides an intention that converts `.format()` calls and `printf`-style formatting to f-string literals. This intention is called Convert to f-string literal, and works as follows:

BeforeAfter

```
var = 'hello %s!' % 'John'
```

```
var = f'hello {"John"}!'
```

```
var = 'hello {}'.format('John')
```

```
var = f'hello {"John"}!'
```

Creating and Optimizing Imports

In this part:

- [Creating Imports](#)
- [Optimizing Imports](#)

Creating Imports

On this page:

- [Introduction](#)
- [Importing packages on the fly](#)
- [Importing TypeScript symbols](#)
- [Importing an XML namespace](#)

Introduction

When you reference a class that has not been imported, PyCharm helps you locate this file and add it to the list of imports. You can import a single class or an entire package, depending on your settings.

The import statement is added to the imports section, but the cursor does not move from the current position, and your current editing session does not suspend. This feature is known as the Import Assistant.

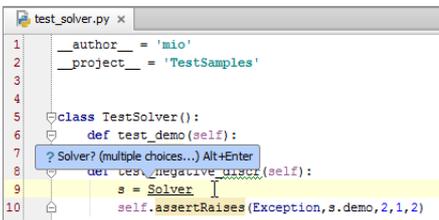
Tip To configure the way auto-import works, [open the PyCharm settings](#), and then go to the page [Auto Import](#).

The same possibility applies to the XML files. When you [type a tag with an unbound namespace](#), import assistant suggests to create a namespace and offers a list of appropriate choices.

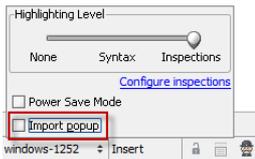
Importing packages on the fly

To import packages on-the-fly, follow these steps:

1. Start typing a name in the editor. If the name references a class that has not been imported, the following prompt appears:

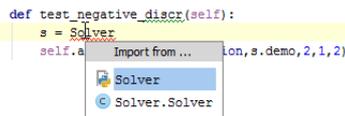


If the pop-up annoys you, change this behavior for the current file. Just click Hector  in the [Status bar](#), and clear the check box [Import Pop-up](#):

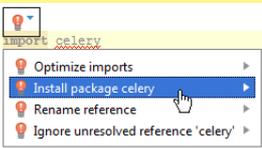


The unresolved references will be underlined, and you will have to [invoke intention action](#) [Add import explicitly](#).

2. Press [Alt+Enter](#). If there are multiple choices, select the desired import from the list.



Tip PyCharm provides a quick fix that automatically installs the package you're trying to import: if, after the keyword `import`, you type a name of a package that is not currently available on your machine, a quick fix suggests you to either ignore the unresolved reference, or download and install the missing package:

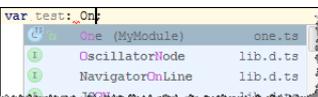


Importing TypeScript symbols

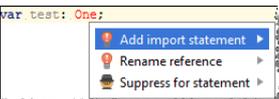
In the [TypeScript](#) context, PyCharm can generate `import` statements for modules, classes, and any other symbol that can be exported and called as a type.

Open the desired file in the editor and do one of the following:

- Start typing the short name of a symbol. From the suggested variants for completion, select the relevant symbol name:



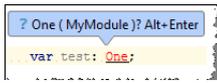
- Position the cursor at the unresolved symbol, which is displayed in red, and press [Alt+Enter](#):



On the context menu, select [Add import statement](#) and press [Enter](#).

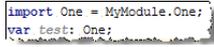
- Configure PyCharm to show a pop-up every time you hover the mouse pointer over an unresolved reference which required import:
 1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing `File | Settings for Windows and Linux or PyCharm | Preferences for macOS`. Expand the Editor node, and then click `Auto Import` under `General`.
 2. On the `Auto Import` page that opens, select the `Show import pop-up` check box in the `TypeScript` area.

Every time you hover the mouse pointer over an unresolved symbol, PyCharm will display the following pop-up message:



Press `Alt+Enter` to have an import statement generated and inserted automatically.

In either case, PyCharm inserts an `import` statement:

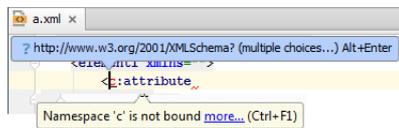


You can configure the quotes style for generated `import` statements on the [Code Style. TypeScript](#) page, tab `Punctuation` (`File | Settings | Editor | Code style | TypeScript | Punctuation for Windows and Linux or PyCharm | Preferences | Editor | Code style | TypeScript | Punctuation for macOS`).

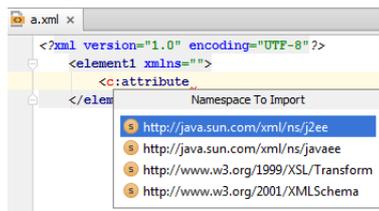
Importing an XML namespace

To import an XML namespace, follow these steps:

1. Open the desired file for editing, and start typing a tag. If a namespace is not bound, the following prompt appears:



2. Press `Alt+Enter`. If there are multiple choices, select the desired namespace from the list.



Optimizing Imports

On this page:

- [Introduction](#)
- [Optimizing imports in project](#)
- [Optimizing imports in the current file](#)

Introduction

Sooner or later, some of the imported classes or packages become redundant to the code.

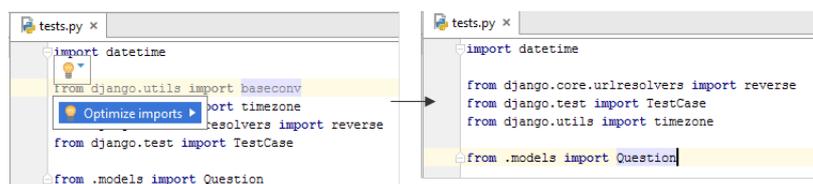
Typically, you have to stop what you are doing, scroll to the head of the file, find the unused imports, and remove them. It is rather easy to forget to remove imports when you remove usages.

PyCharm provides the Optimize Imports feature, which enables you, whenever it is convenient, to remove unused imports from your current file, or from all files in the current directory at once. This helps you avoid unused, excessive and duplicating imports in your project.

One can remove unused import statements in the entire project or in the current file only.

Besides cleaning the code from the unused imports, PyCharm formats the existing import statements according to the [Style Guide for Python Code](#). So doing, PyCharm splits import statements into separate lines, and sorts them into groups (refer to the [Imports](#) section for details).

Also, imports are sorted alphabetically within the respective groups:



Optimizing imports in project

To optimize imports in the entire project, follow these steps:

1. Place the caret at the Project tool window and do one of the following:
 - On the main menu, choose Code | Optimize Imports.
 - Press `Ctrl+Alt+O`.

The **Optimize Imports** dialog box opens.

2. If your project is under version control, the option Only VCS changed files is enabled. Select or clear this option as required.
3. Click Run.

Optimizing imports in the current file

One way of dealing with unused import is to use the quick-fix that appears when you set the caret at the highlighted unused import. However, you can optimize imports in a larger scope as described below.

To optimize imports in the currently opened file, do one of the following:

- On the main menu, choose Code | Optimize Imports.
- Press `Ctrl+Alt+O`.
- Place the caret at the import statements and press `Alt+Enter`, or just click  to show the list of suggested intention actions, and then choose Optimize imports.

Viewing Reference Information

PyCharm facilitates quick and easy access to the API documentation, which you can view immediately in the editor or in an external browser. To gain access to the API documentation from within your project, make sure to attach archives or directories that contain sources of the library classes.

This section describes how to see definitions of symbols, display documentation references, and use the view parameter information feature:

- [Viewing Definition](#)
- [Viewing Inline Documentation](#)
- [Viewing External Documentation](#)
- [Viewing Method Parameter Information](#)

These features are available in all supported language contexts. For information on retrieving reference in JavaScript context, refer to section [Viewing JavaScript Reference](#).

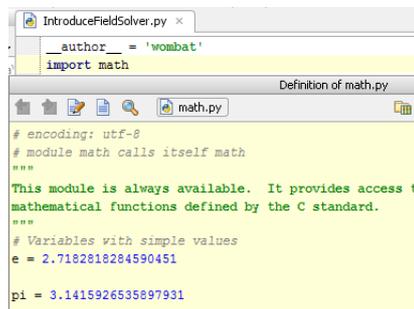
Viewing Definition

On this page:

- Basics
- Viewing the definition of a symbol at caret
- Toolbar of the quick definition lookup

Basics

Quick Definition Lookup makes it possible to view definition of a symbol (tag, class, method/function, field, etc.) in a pop-up window.



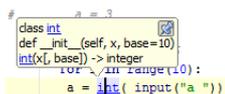
For markup languages, PyCharm retrieves definitions of symbols from the specified DTD or schema. For details, see [Markup Languages and StyleSheets](#).



Viewing the definition of a symbol at caret

Do one of the following:

- On the main menu, choose View | Quick Definition.
 - Press **Ctrl+Shift+I**.
 - Keeping the **Ctrl** key pressed, point with your mouse cursor to the symbol of interest, so that it turns to a hyperlink, with the definition of the symbol displayed in a tooltip. Clicking this hyperlink results in opening the respective definition page in the editor.
- Quick definition tooltip shows hyperlinks to the symbols involved.



When you move your mouse pointer within the tooltip, a pin button  appears. If you pin the tooltip, documentation for the symbol at caret is displayed in the [Documentation Tool Window](#).

Toolbar of the quick definition lookup

Use the icons on the toolbar of the pop-up window to navigate to the source code of the definition and view its usages.

Icon KeyboardAction shortcut

	Shift+Alt+Left Shift+Alt+Right	Navigate to the previous/next screen in the definition pop-up window after using hyperlinks in the definition.
	F4	Open the source code of the definition for editing, and close the quick definition lookup window.
	Ctrl+Enter	Open the source code of the definition, and preserve the quick definition lookup window opened.

Viewing Inline Documentation

On this page:

- [Basics](#)
- [Viewing quick documentation](#)
- [Toolbar of the quick documentation lookup](#)

Basics

Quick Documentation Lookup helps you get quick information for any symbol or just method signature information, provided that this symbol has been supplied with documentation comments in the applicable format.

PyCharm recognizes inline documentation created in accordance with the [docstring](#) format.

Such documentation is rendered in the [Documentation Tool Window](#) according to the docstring format, selected in the [Python Integrated Tools](#) page.

For information on retrieving inline documentation in the JavaScript context, refer to section [Viewing JSDoc Comments](#).

The URLs and e-mail addresses specified in the documentation comments for methods are also properly rendered. Clicking a hyperlink opens the corresponding URL in an external browser; clicking an e-mail address opens the default mail client.



Viewing quick documentation

To view documentation for a symbol at caret, do one of the following

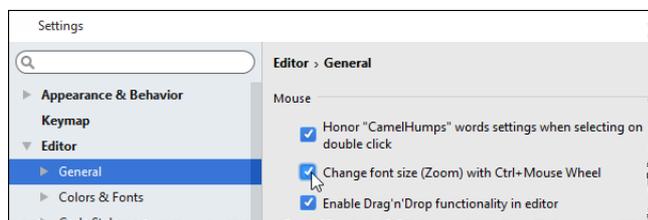
- On the main menu, choose View | Quick Documentation Lookup.
- Press `Ctrl+Q`.
- Provided that the check box [Show quick doc on mouse move](#) in the editor settings is selected, just move your mouse pointer over the desired symbol.

When you explicitly invoke code completion, then quick documentation for an entry selected in the suggestion list can be displayed automatically. The behavior of quick documentation lookup is configured in the [Code Completion](#) page of the Settings/Preferences dialog.

Tip Sequential pressing `Ctrl+Q` toggles the focus of the Quick Documentation pop-up window and [Documentation Tool Window](#).

To change the font size of quick documentation, do one of the following

- Click in the upper-right corner of the quick documentation window, and move the slider.
 - Rotate the mouse wheel while keeping the `Ctrl` key pressed.
- Note that for this feature to work, you have to enable it in the [General](#) page of the editor settings.



Refer to the section [Zooming in the Editor](#) for details.

Toolbar of the quick documentation lookup

The Quick Documentation Lookup window helps navigate to the related symbols via hyperlinks, and provides a toolbar for moving back and forth through the already navigated pages, changing font size, and viewing documentation in an external browser.

When pinned, the Quick Documentation Lookup turns into [Documentation Tool Window](#), with the corresponding sidebar icon, and more controls.

IconShortcutDescription

`Left` or `Right` Switch to the previous or next documentation page (e.g. after using hyperlinks).

Note On an macOS computer, you can also use the three-finger right-to-left and left-to-right swipe gestures.



Shift+F1

View external documentation in the default browser.



F4

Switch to the item (e.g. source) that corresponds to the documentation page currently shown.



Turn the Auto-update from source option on or off. When the option is on, the information in the tool window is synchronized with your navigation in the editor and other places in the UI.



Click this icon to show font size slider. Move the slider to increase or decrease the font size in the quick documentation window as required.

Viewing External Documentation

External documentation makes it possible to get additional information for the symbols at caret. In contrast to the [quick documentation](#), this feature shows the documentation in an external browser, which helps study the symbol in more detail, navigate to related symbols, and retain the information for further reference.

PyCharm shows documentation for [SciPy](#), [NumPy](#), [PyGTK](#), and the other modules you've installed for your Python interpreter. External documentation should be properly configured in the [Python External Documentation](#) page of the Settings/Preferences.

To view documentation for a symbol at caret in an external browser, do one of the following:

- On the main menu, choose View | External documentation.
- Press `Shift+F1`.
- While in the [Quick Documentation Lookup window](#), click .

Viewing Method Parameter Information

On this page:

- [Parameter hints for methods](#)
- [Configuring the behavior of parameter hints](#)

Parameter hints for methods

Place the caret anywhere within the call of the desired method or function and choose View | Parameter Info on the main menu or press `Ctrl+P`.

```
for index in range(len(data)-1, -1, -1):  
    yield data[index] start=None, stop=None, step=None
```

Configuring the behavior of parameter hints

Open the [Code Completion page](#) (Settings | Editor | General | Code Completion for Windows and Linux or PyCharm | Preferences | Editor | General | Code Completion for macOS) and configure the following options in the Parameter info section:

1. To have a complete method or function signature shown rather than a list of required types, select the Show full signatures check box.

Make sure to include the required third-party [libraries](#) in the project source path. Otherwise, names of the parameters will not be displayed.

2. To have the list of parameter types for the called method or function shown automatically after a certain delay, select the Auto pop-up (in ms) check box and specify the time period in milliseconds.

Viewing Pages with Web Contents

With PyCharm, you can perform two opposite operations:

- Preview the output of your Web application in the browser to check whether the pages are rendered correctly. PyCharm can display a page preview in a browser of your choice or you can switch between several browsers. PyCharm currently supports previews in the following browsers:
 - Mozilla Firefox
 - Internet Explorer
 - Safari
 - Chrome
 - Opera
- View the HTML source code of a Web page in the PyCharm editor.

In this section:

- [Configuring Browsers](#)
- [Previewing Pages with Web Contents in a Browser](#)
- [Viewing HTML Source Code of a Web Page in the Editor](#)

Configuring Browsers

On this page:

- [Integrating browser installations with PyCharm](#)
- [Choosing the default PyCharm browser](#)

Integrating browser installations with PyCharm

To make it possible to launch a Web browser from PyCharm, you need to integrate installations of Web browsers with PyCharm, activate or deactivate launching Web browsers from PyCharm and specify whether a browser will be launched by running its executable file or through the default system command .

PyCharm is shipped with a predefined list of most popular browsers which you may like to install and use. The items are added to the list in advanced and are not based on the information on actually installed browsers. PyCharm presumes that you install browsers according to a standard procedure. Based on this assumption, each browser in this predefined list is assigned an **alias** which stands for the path to its executable file, as PyCharm supposes it to be. If in your actual browser installation the path to the executable file is different, you need to specify it explicitly as described below.

In addition to the predefined browsers, you can configure as many custom browser installations as you need using the controls on the toolbar. To create a list of Web browser that can be launched from PyCharm:

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Web Browsers under Tools.
2. The [Web Browsers](#) page that opens shows a **predefined list of browsers**, possibly extended with previously configured **custom browser installations**. Each browser is presented as a separate table row. The fields in each row show the name of the browser, the family to which the browser belongs, and the path to the browser's executable file or the predefined **alias** that stands for this path.
 - To activate an actually installed browser, select the Active check box next to its name. The browser will be added to the context menu of the Open in Browse menu item and its icon will be displayed in the Browsers pop-up toolbar.
 - If the browser was installed according to a standard installation procedure, most likely the **alias** shown in the Path field points at the right location of the executable file. To specify the path explicitly, click in the Path field and choose the actual location of the executable file in the dialog box that opens.
 - To configure a custom browser installation, click **+** on the toolbar. In the new row that is added to list, specify the browser name, family and the path to its executable.
 - To change the order of browsers in the list, use the **↑** and **↓** buttons. The order of browsers in the list affect the order in which they will be shown on the context menu of the Open in Browse menu item.
 - To specify a custom profile for **Firefox** or a browser of the **Chrome** family, select the browser in question, click **🌿** on the toolbar. Depending on the family of the selected browser, the Firefox Settings or Chrome Settings dialog box opens.
 - For **Firefox**, specify the path to the required `profiles.ini` file and choose the profile to use from the drop-down list. Learn more at [Firefox browser profile](#).
 - For **Chrome**, select the Use custom profile directory check box and specify the location of the `chrome-user-data` folder where users' profiles are stored. Learn more about **Chrome** profiles at [Multi-profiles](#).
 - To launch a browser of the **Chrome** with additional options, click **🌿** on the toolbar and type the required keys in the Command Line Options text box of the Chrome Settings dialog box that opens. Learn more about Chrome command line options by opening `chrome://flags` in **Chrome**.
 - To remove a browser from the list, select the browser and click **–** on the toolbar. Note that only custom browser can be removed.

Choosing the default PyCharm browser

When you want to preview your application output in the browser by choosing View | Open in Browser on the main menu or Open in Browser on the context menu of a file, you need to choose the browser to open the preview in. You can use a specific browser from the context menu or choose Default Browser. tell PyCharm which browser you want to be used by default. This browser is called **PyCharm default browser**.

PyCharm also opens the **PyCharm default browser** to render external resources.

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Web Browsers under Tools. The [Web Browsers](#) page opens.
2. From the Default Browser drop-down list, choose the browser to use by default for previewing pages.
 - To use the default operating system browser, choose System default.
 - To use the browser on top of the list, choose First listed. Change the order of browsers using the **↑** and **↓** icons on the toolbar.
 - To use another browser as default, choose Custom path and specify the location of the executable file of the required browser. Type the path manually or use the Browse button , if necessary.
3. Specify the way to have the browsers launched.
 - To have a popup window with the enabled browsers appear in the HTML files, select the Show browser popup in the editor check box.
 - If the Show browser popup in the editor check box is cleared, previewing HTML files is available only through the View | Open in Browser command on the main menu or the Open in Browser command on the context menu of a file.

Previewing Pages with Web Contents in a Browser

You can preview a file with Web contents in a browser. This can be the [PyCharm default browser](#) specified in the [Web Browsers](#) section of the IDE settings or the one of your choice.

To preview an opened file in a Web browser, do one of the following:

- On the main menu, choose View | Open in Browser. The current file opens in the default browser.
- With the editor tab having the focus, choose View | Preview file in on the main menu or press `Alt+F2`. Then select the desired browser from the pop-up menu:



- Hover your mouse pointer over the code to show the browser icons bar, and click the icon that indicates the desired browser:



Tip If you don't want to see the icons toolbar, or would like to see the icons of just those browsers you are interested in, clear the Active check boxes for the unnecessary browsers in the [Web Browsers](#) section of the IDE settings.

Viewing HTML Source Code of a Web Page in the Editor

You can open the HTML source code of any Web page in the PyCharm editor.

No matter which programming language was originally used to develop a page (for example, XML or PHP), PyCharm shows the resulting HTML code.

To open the HTML source code of a Web page in the editor

1. Choose File | Open URL.
2. In the Open URL dialog box that opens, specify the URL address of the desired Web page. Type the URL address manually or choose a previously specified one from the drop-down list.

Navigating Through the Source Code

PyCharm suggests various ways of navigation between the IDE components and within source code. PyCharm's smart editor makes it possible to navigate across the source code using the code structure rather than plain scrolling. You can find your way in the source code using the method calls, declarations, errors, changes etc.

In this part:

- [Navigating with Bookmarks](#)
- [Navigating Between Files and Tool Windows](#)
- [Navigating Between IDE Components](#)
- [Navigating Between Methods and Tags](#)
- [Navigating from Stacktrace to Source Code](#)
- [Navigating Between Test and Test Subject](#)
- [Navigating to Action](#)
- [Navigating to Braces](#)
- [Navigating to Class, File or Symbol by Name](#)
- [Navigating to Custom Folding Regions](#)
- [Navigating to Declaration or Type Declaration of a Symbol](#)
- [Navigating to Super Method or Implementation](#)
- [Navigating to File Path](#)
- [Navigating to Line](#)
- [Navigating to Navigated Items](#)
- [Navigating to Next/Previous Change](#)
- [Navigating to Next/Previous Error](#)
- [Navigating to Recent File](#)
- [Navigating with Breadcrumbs](#)
- [Navigating with Navigation Bar](#)
- [Navigating with Structure Views](#)

Navigating with Bookmarks

On this page:

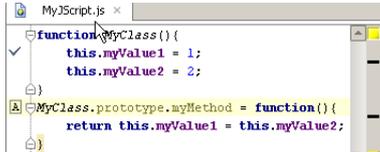
- [Introduction](#)
- [Navigating within the current file](#)
- [Navigating across a project](#)

Introduction

The editor of PyCharm provides two types of bookmarks:

- Anonymous bookmarks, indicated by check signs ✓ in the left gutter. The number of anonymous bookmarks is unlimited.
- Bookmarks with mnemonics indicated by **N** or **S** icons in the left gutter. There can be only 10 numbered and 26 lettered bookmarks within a project.

All bookmarks are indicated with the black streaks in the marker bar:



Once created, the bookmarks enable you to easily jump to the desired location within a file, or across the entire project.

Navigating within the current file

To navigate through the bookmarks within the current file, do one of the following

- On the main menu, choose `Navigate | Bookmarks | Next/Previous Bookmark`. The order of visiting bookmarks depends on their order in the collection of bookmarks in the [Bookmarks dialog](#).
- Click the black streak in the marker bar.

Navigating across a project

To navigate across a project using numbered bookmarks

- Use `Ctrl+Number` where the <number> corresponds to the desired bookmark.

To navigate among all bookmarks in a project, do one of the following

- On the main menu, choose `Navigate | Bookmarks | Show Bookmarks`, or press `Shift+F11`. In the [Bookmarks dialog](#), select the target bookmark, and press `Enter`.

For your convenience, the target code preview is shown in the right pane of the dialog box.

- In the [Favorites tool window](#), select the desired bookmark in the Bookmarks list, and then double-click the bookmark entry, or press `F4`. The corresponding file opens in the editor, with the caret at the beginning of the bookmarked line.

Managing Bookmarks

This section describes how to:

- [Create and delete bookmarks with mnemonics](#)
- [Toggle bookmarks](#)
- [View project bookmarks](#)
- [Delete bookmarks](#)
- [Change the order of bookmarks](#)

To create a bookmark with mnemonics

1. Place the caret at the desired line of code in the editor.
2. Press `Ctrl+F11` (alternatively, choose `Navigate | Bookmarks | Toggle Bookmark With Mnemonic` on the main menu), then press one of the keys 0-9 or A-Z.

To toggle an anonymous bookmark on the current line

- On the main menu, choose `Navigate | Bookmarks | Toggle Bookmark`.
- Press `F11`.

To view all bookmarks in a project

- On the main menu, choose `Navigate | Bookmarks | Show Bookmarks`.
- Press `Shift+F11`.

To delete bookmarks in a project

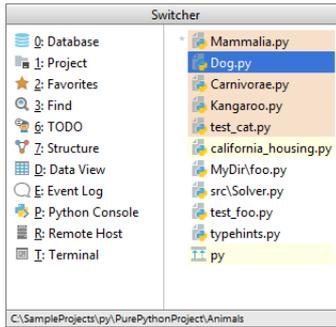
1. Open the Bookmarks dialog (`Navigate | Bookmarks | Show Bookmarks`, or `Shift+F11`).
2. Select bookmarks and press `Delete`.

To change the order of bookmarks

1. Open the Bookmarks dialog (`Navigate | Bookmarks | Show Bookmarks`, or `Shift+F11`).
2. Select the desired bookmark.
3. Use `Move Up` (`↑`, `Ctrl+Up`) and `Move Down` (`↓`, `Ctrl+Down`) buttons to shift the bookmark in the desired direction.

Navigating Between Files and Tool Windows

PyCharm suggests a handy way to switch between files opened in the editor, split editor tabs, and tool windows (docked or floating). This is similar to the application switchers in the various operating systems. The switcher consists of two columns: the left one displays the list of tool windows, while the right one displays the list of files currently opened in the editor.



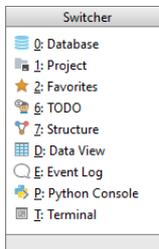
To switch between open files and tool windows

1. Press `Ctrl+Tab`.

2. Keeping `Ctrl` pressed, use the following keys:

- `Up` and `Down` arrow keys, `Tab` or `Shift+Tab` to go up and down the list in both panes.
- `Delete` or `Backspace` to close the editor tab where the selected file is opened and delete the selected file from the list.

When all the files are deleted, the only column left shows the list of the tool windows:



3. Having selected the desired file or tool window, release the `Ctrl` key. The corresponding file or tool window gets the focus, and the switcher pop-up window disappears.

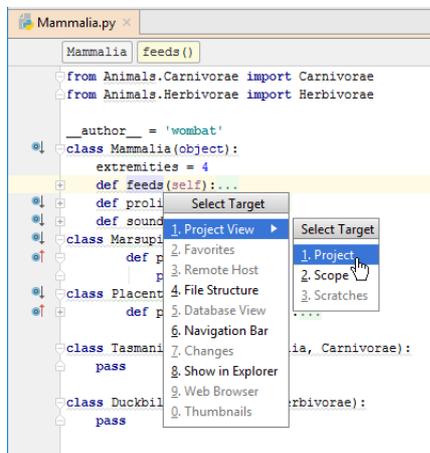
Navigating Between IDE Components

Suppose you have selected a file or member in one of the tool windows, and would like to quickly find it in the Project view. The Select Target pop-up menu moves the focus to the selected component of PyCharm, the file system, etc.

- [Project View](#)
- [Favorites](#)
- [Remote Host](#)
- [Framework Views](#)
- [File Structure](#)
- [Database View](#)
- [Coverage](#)
- [Navigation Bar](#)
- [Version Control tool window](#)
- Show in Explorer
- Web Browser
- Thumbnails

To navigate to the desired component

1. On the main menu select Navigate and click Select In, or press `Alt+F1`.
The Select Target pop-up menu shows up.
2. Use the arrow keys or the mouse pointer to select the desired component. If your target is the Project tool window, you can select the desired view: Project, Scratches etc.



Tip Double-click selected file or member in one of the tool windows, or press `Enter` to open it in the editor.

Navigating Between Methods and Tags

Since your source code can contain numerous methods, it is convenient to navigate to the beginning of the next or previous method. In the Web contents, this feature enables navigating between tags.

To navigate to the next or previous method or tag

- On the main menu, choose `Navigate | Next Method / Previous Method` respectively.
- Use `Alt+Up` / `Alt+Down` keyboard shortcuts.

For JavaScript code inside HTML files this behavior depends on the caret location. If the caret rests inside a JavaScript block, this means of navigation enables jumping between JavaScript functions. If the caret rests on a `<script>` tag, then navigation is performed between the tags.

To improve the visibility of your source code by having a line added automatically between adjacent methods, choose the `Show Method Separators` option in the [Appearance](#) page of the editor settings.

Navigating from Stacktrace to Source Code

You can easily navigate from the stack trace in the Run tool window to the source code that causes problems.

To navigate from the stack trace to a line of code

- In the Run tool window scroll to the desired stack trace line and click the link to the source file in question. The source file opens in the editor.

Navigating Between Test and Test Subject

On this page:

- [Overview](#)
- [Jumping from a test to its test subject](#)
- [Jumping from a class or file to its test](#)

Overview

Testing support in PyCharm provides the ability to navigate between a test and the test subject.

For information on common testing procedures, see [Testing](#).

For language- and framework-specific guidelines, see , [Testing Frameworks](#), [Testing JavaScript](#), [Running Nodeunit Tests](#), and [Language and Framework Specific Guidelines](#).

Jumping from a test to its test subject

1. Open the desired test class in the editor.
2. On the main menu or on the context menu of the editor, choose `Navigate | Test Subject`. Alternatively, press `Ctrl+Shift+T`.
The test subject for the current test class opens in the dedicated tab of the editor and gets the focus.

Jumping from a class or file to its test

1. Open the desired class in the editor.
2. On the main menu or on the context menu of the editor, choose `Navigate | Test`. Alternatively, press `Ctrl+Shift+T`.
If more than one test is associated with the test subject, select the desired one from the pop-up list. The test for the current class opens in the dedicated tab of the editor and gets the focus.

Note If a test class doesn't exist, you will be prompted to create one as described in the section [Creating Tests](#) .

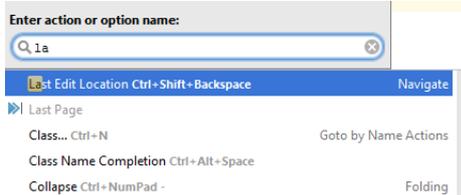
Navigating to Action

PyCharm helps you quickly find the desired action, without digging through the menus and toolbars. The concept action covers the commands of the main menu and various context menus, commands performed through the toolbar buttons of the main toolbar and tool windows.

To find an action

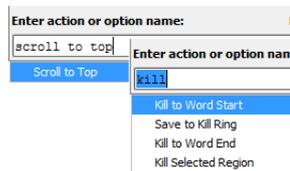
1. Choose Help | Find Action on the main menu or press `Ctrl+Shift+A`. The pop-up window that opens, shows the suggestion list of matching names. By default, this list includes the menu commands only. If you want to include the non-menu commands in the suggestion list, press `Ctrl+Shift+A` once more.

2. Start typing the desired action name. As you type, the suggestion list displays the matching names of actions. The actions that are not valid in the current context are displayed gray.



3. Double-click the desired entry in the suggestion list, or select it using the arrow keys and press `Enter`.

This way you can invoke the actions that are not mapped to keyboard shortcuts in certain schemes (for example, scroll to top, scroll to bottom, or Emacs actions, like kill rings, sticky selection, or hungry backspace).



These actions are not mapped to certain keyboard shortcuts, neither they appear in the menus. If necessary, configure keyboard shortcuts for these actions as described [here](#).

Navigating to Braces

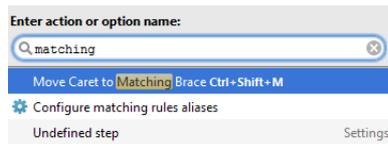
In this section:

- [Navigating to the borders of a code block](#)
- [Navigating to the borders of the closest higher code block](#)

Navigating to the borders of a code block

To navigate to the borders of a code block, do one of the following:

- To navigate to the code block start, press `Ctrl+Open Bracket`, with the caret anywhere inside the code block.
In the Python code, the caret rests before the `def` keyword.
- To navigate to the code block end, press `Ctrl+Close Bracket`, with the caret anywhere inside the code block.
In the Python code, the caret rests after the end of code block.
- To toggle between code block start or end, press `Ctrl+Shift+M`.
You can also invoke this action (Move Caret to Matching Brace) with a [Search Everywhere](#) or [Go to Action](#) functionality:



Navigating to the borders of the closest higher code block

To navigate to the borders of the closest higher code block, do one of the following:

- To jump to the higher code block start, press `Ctrl+Open Bracket`, with the caret at the [current code block opening brace](#).
- To jump to the higher code block end, press `Ctrl+Close Bracket`, with the caret at the [current code block closing brace](#).

Tip Practically, you can just press `Ctrl+Open Bracket` or `Ctrl+Close Bracket` as many times as you need, until the caret is positioned at the start or end of the desired code block.

Navigating to Class, File or Symbol by Name

In this section:

- [Overview](#)
- [Navigating by name](#)
- [Tips and tricks](#)

Overview

Navigate commands enable you to quickly jump to the desired classes, files, or symbols specified by names. PyCharm suggests a look-up list of matching names, from which you can select the desired one, and open it in the editor. This navigation honors CamelCase and snake_case capitalization. Refer to the [tips](#) for detailed list of available techniques.

Navigating by name

To navigate to a class, file, or symbol with the specified name:

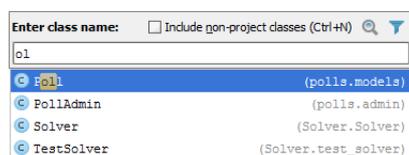
1. On the main menu, point to Navigate, and then choose Class, File, or Symbol respectively, or use the following shortcuts:

- Class: `Ctrl+N`
- File (directory): `Ctrl+Shift+N`
- Symbol: `Ctrl+Shift+Alt+N`

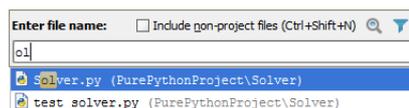
2. In the pop-up window, start typing the desired name.

So doing, you can enter characters located anywhere inside the desired name. As you type, the suggestion list shrinks, displaying the matching names only.

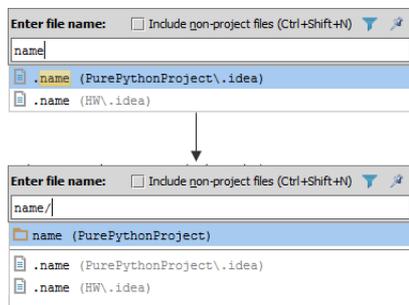
- Class:



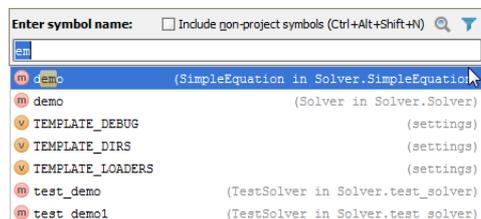
- File:



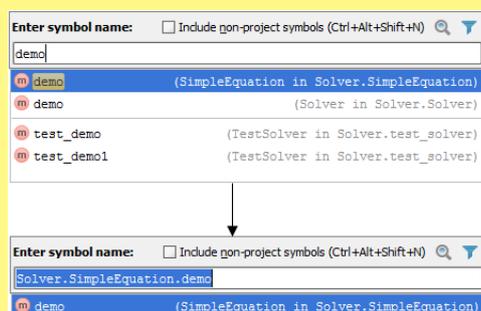
- Directory: use the same `Ctrl+Shift+N` shortcut as for file navigation, and type the name of the directory you are looking for, the pattern name ending with / or \:



- Symbol:



Note If there are several symbols with the same name, you can make your search more precise by specifying the hierarchy path. For example, if you want to find a certain function that belongs to different classes, you can type the function name, and see all possible locations in the look-up list. On the other hand, it is possible to specify the class or file name, followed by a dot, and then the desired function:

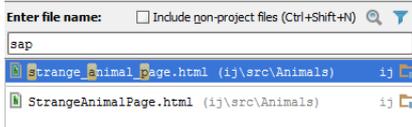


3. Click the desired entry in the suggestion list, or select it using the arrow keys, and press `Enter`.

Tips and tricks

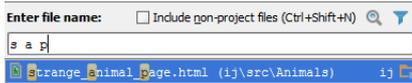
While working in the navigation pop-up window, use the following helpful techniques:

- Narrow down the search scope by selecting the file types to search in. Just click the filter , and clear the check boxes next to the file types you are not interested in.
- Include non-project files in the look-up list and thus make available matching files from SDKs and libraries.
- If the look-up list is too long, type more characters to shrink it, or click the ellipsis sign at the end of the list, to reveal its next portion.
- Type the initial letters of the **CamelHumps** names, for example:

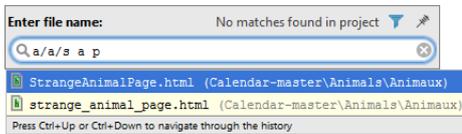


Note that PyCharm automatically recognizes **CamelHumps** and matches them to the lower case letters.

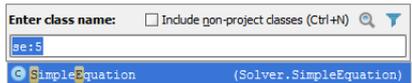
- Type any letters separated with spaces for **snake_case** names, for example:



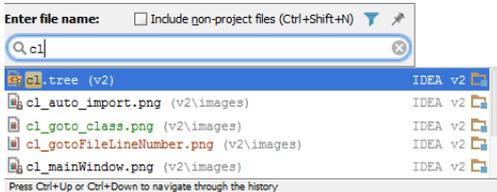
- In the navigation to file pop-up window, type letters delimited with slashes to denote nested directories:



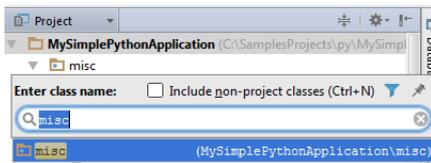
- Type line number after a file name, delimited with a colon, to navigate to the specified line:



- Use `*` wildcard to represent any number of characters, though it is quite enough to type characters located in the middle of the desired name.
- If while typing in one of the Navigate to Class/File/Symbol pop-up windows you notice that you need another one, just invoke the necessary dialog box. The text you have already entered will not disappear.
- Press `Alt+F1` to invoke the Select Target pop-up window, and choose the desired IDE component.
- Note that for the projects under version control, the entries in the look-up list are color coded according to their status:



- When there is a **detached editor frame** with a certain file, you can opt to open this file in the main PyCharm frame by pressing `Enter`, or activate the detached frame by pressing `Shift+Enter`.
- Navigate|Class or Navigate|Symbol allows navigating to Python modules (`*.py` files) and packages (`'__init__.py'` files):



Navigating to Custom Folding Regions

If there are code folding comments in your file (see [Using code folding comments](#)), you can navigate to corresponding folding regions like this:

1. Select Navigate | Custom Folding or press `Ctrl+Alt+Period`.
2. Select the target folding region. (The regions in the list are identified by their descriptions.)

Navigating to Declaration or Type Declaration of a Symbol

On this page:

- [Introduction](#)
- [Important note](#)
- [Navigating to the declaration of a symbol](#)
- [Navigating to the type declaration of a symbol](#)

Introduction

While editing your source code, you might need to navigate to the location where a particular named code reference (a symbol) has been first declared.

Navigate | Declaration command enables you to navigate back to the initial declaration of a symbol from any place in the source code, even if it is from inside another class, or comment.

Important note

Navigate | Declaration/Type Declaration

- Applies to the symbols of source code, CSS, HTML or XML tags and attributes, DTD and schema elements and attributes, and references in comments.
- Does not apply to the primitive types.

Navigating to the declaration of a symbol

1. Place the caret at the desired symbol in the editor.
2. Do one of the following:
 - On the main menu, choose Navigate | Declaration.
 - Press `Ctrl+B`.
 - Click the middle mouse button.
 - Keeping `Ctrl` for Windows or Linux users or `⌘` for macOS users pressed, point to the symbol, and click, when it turns to a hyperlink. You can also see declaration at the tooltip while keeping `Ctrl` for Windows or Linux users or `⌘` for macOS users pressed.

```
def index(request):  
    poll_list = render_to_response(*args, **kwargs)  
    # latest_poll = render_to_response('index.html', {'poll_list': poll_list})  
    return render_to_response('index.html', {'poll_list': poll_list})
```

Navigating to the type declaration of a symbol

This navigation is available in the JavaScript context only.

1. Place the caret at the desired symbol in the editor.
2. Do one of the following:
 - On the main menu, choose Navigate | Type Declaration.
 - Press `Ctrl+Shift+B`.
 - Press the `Ctrl+Shift` for Windows and Linux users, or `⌘+⇧` for macOS users keys and hover your mouse pointer over the symbol. When the symbol turns to a hyperlink, click it without releasing `Ctrl+Shift` for Windows and Linux users, or `⌘+⇧` for macOS users keys. The type declaration of the symbol opens in the editor. You can also see the declaration at the tooltip while keeping `Ctrl+Shift` for Windows and Linux users, or `⌘+⇧` for macOS users pressed.

```
};  
public class String extends Object  
{  
    public String replace(/\\/s/gi, "+").replace("Control", "Ctrl");  
}
```

Navigating to Super Method or Implementation

PyCharm provides an easy way to navigate up and down through the hierarchy of methods. If a method is overridden by a certain method, or overrides some method itself, it is marked with an icon in the gutter area of the editor. When the mouse cursor hovers over such icon, the method information is displayed as the tooltip:

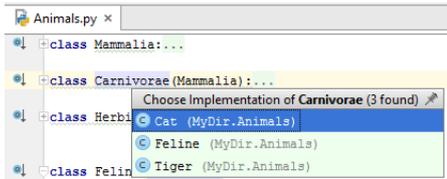
- : This method overrides a method defined by its superclass.
- : This method or field is overridden by one or more subclasses.

Use these icons, shortcuts, or menu commands to navigate to the corresponding points of origin.

Navigating through the hierarchy of methods

To navigate up and down through the method hierarchy, do one of the following:

- Click the gutter icon and select the desired ascendant or descendant class from the list.
- On the main Navigate menu, choose Super Method, or Implementation(s) respectively.
- Press `Ctrl+U` or `Ctrl+Alt+B` for the super method or implementation respectively.



Navigating to File Path

On this page:

- [Overview](#)
- [Navigating to a file path](#)
- [Viewing file path](#)

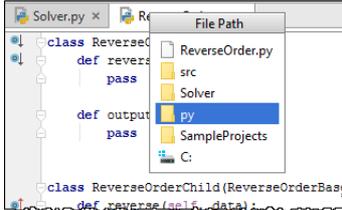
Overview

It may be helpful to open a file in a file manager (for example, in the Windows Explorer, if Windows is your OS), or the Finder. PyCharm allows you to easily navigate to any part of a file path, right from the editor.

Navigating to a file path

To navigate to a file path from the editor

1. Hold `Ctrl` / `⌘` and click the relevant tab in the editor.
2. In the drop-down list, select the element you need.



The file manager shows the selected file or directory.

Viewing file path

To just view the path to a file, place the mouse pointer over the tab where the file is opened.



Navigating to Line

Navigate | Line command is the most basic way to navigate to the line with the specified number.

To navigate to a line in the editor

1. On the main menu, choose Navigate | Line, or press `Ctrl+G`.
2. In the Go To Line dialog box that appears, PyCharm shows the current line and offset delimited by a colon. Type the target line number and, optionally, an offset, and then click OK.

The current caret position (line number and offset) is displayed in the [Status Bar](#). You can click it to open the Go To Line dialog box.

Navigating to Navigated Items

You can go back and forth along the items that have already been navigated to. This feature applies to the items reached using all navigation commands except for the simplest ones, such as arrow keys, Page Up, Page Down, Home and End.

To navigate to the navigated items

Do one of the following:

- On the main menu, choose Navigate | Back / Forward.
- Use keyboard shortcuts `Ctrl+Alt+Left`, or `Ctrl+Alt+Right`.
- On the main toolbar, click  or .

Note On an macOS computer, you can also use the three-finger right-to-left and left-to-right swipe gestures.

Navigating to Next/Previous Change

If you edit a file that is under version control, PyCharm provides several ways to move back and forth with the updates. In particular, you can use the navigation commands, keyboard shortcuts, and the change markers.

To navigate to the next/previous change in the editor, do one of the following:

- On the main menu, choose `Navigate | Next / Previous Change`.
- Use keyboard shortcuts `Ctrl+Shift+Alt+Down` or `Ctrl+Shift+Alt+Up`.
- Point to a [change marker](#), and click the arrow up  or arrow down  buttons.

To navigate to the place of your last edit, do one of the following:

- On the main menu, choose `Navigate | Last Edit Location`
- Use keyboard shortcut `Ctrl+Shift+Backspace`.

Navigating to Next/Previous Error

Another method of code navigation is to **move between found errors and warnings**. The caret is positioned immediately before the code issue.

You can configure the way PyCharm navigates between code issues: it can either jump between all code issues or skip minor issues and only navigate between detected errors.

In this section:

- [Configuring the error navigation](#)
- [Navigating between errors or warnings](#)

To configure the error navigation

1. Right click the [Validation Side Bar](#).
2. On the context menu, choose one of the available navigation modes:
 - To have PyCharm skip warnings, infos, and other minor issues, choose Go to high priority problems only.
 - To have PyCharm jump between all detected code issues, choose Go to next problem.

To navigate between errors or warnings, do one of the following

- On the main menu, choose Navigate | Next / Previous Highlighted Error.
- Use keyboard shortcuts `F2` and `Shift+F2` respectively.

Navigating to Recent File

In this section:

- [Navigating to a recently opened file](#)
- [Navigating to a recently edited file](#)
- [Navigating to the latest edit location](#)
- [Navigating to the next edit location](#)
- [Using multi-selection in the lists of recent files](#)

To navigate to a recently opened file

1. On the main menu, choose View | Recent Files or press `Ctrl+E`.
2. From the Recent Files pop-up window that opens select the desired file.

To navigate to a recently edited file

1. On the main menu, choose View | Recently Changed Files or press `Ctrl+Shift+E`.
2. From the Recently Edited Files pop-up window that opens select the desired file.

Tip The recently opened or recently modified files are selected from the history list. The number of entries in the history list is configurable in the Recent file limit field in the [Editor](#) settings page.

- Navigating to recent files applies to the search results as well. By pressing `Ctrl+E` in the [Find](#) tool window, you can have the list of recent search results shown.

To jump to the latest edit location

- Do one of the following:
 - On the main menu, choose Navigate | Last Edit Location.
 - Press `Ctrl+Shift+Backspace`.

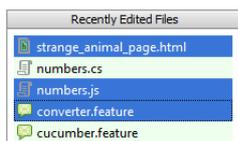
To jump to the next edit location

- On the main menu, choose Navigate | Next Edit Location.

This action is not bound to any shortcut. An interested user can do it as described in the section [Configuring Keyboard Shortcuts](#).

Using multi-selection in the lists of recent files

- To select non-adjacent files, use `Ctrl`/`⌘` + mouse click.
- To select adjacent files, use `Shift`/`⇧` + mouse click.



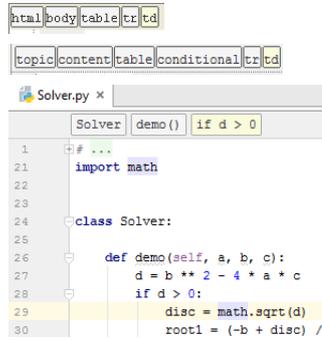
Navigating with Breadcrumbs

On this page:

- [Overview](#)
- [Enabling breadcrumbs](#)
- [Navigating between a breadcrumb and the source code](#)

Overview

In the Python, HTML and XML contexts, you can navigate through the source code using breadcrumbs that are displayed above the main editor window. The breadcrumbs show the elements of a Python, XML or HTML file opened in the [active editor tab](#):



To have breadcrumbs shown, you have to [enable](#) them.

Enabling breadcrumbs

To have breadcrumbs displayed

1. Open the PyCharm Settings/Preferences dialog.
2. Under the Editor node, click General | Appearance.
3. In the [Appearance](#) page, select the Show breadcrumbs check box.

Navigating between a breadcrumb and the source code

To navigate with breadcrumbs

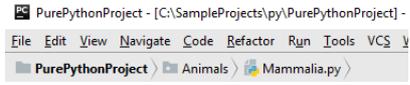
- Click the breadcrumb with the relevant tag. The cursor jumps to the corresponding element in the source code.

Navigating with Navigation Bar

Use the [Navigation Bar](#) as a handy tool to find your way across the project.

To navigate to a file using the Navigation bar

1. Press `Alt+Home` to activate the Navigation bar.
2. Use the arrow keys or the mouse pointer to locate the desired file.
3. Double-click the selected file, or press `Enter` to open it in the editor.



Tip To jump to Python methods/functions, use [breadcrumbs](#).

Navigating with Structure Views

Use the Structure pop-up window or the Structure tool window to quickly jump to the desired member of a file in the editor. The Structure views provide quick navigation for all supported file types.

Besides that, navigation with the Structure pop-up window is also available for [diagrams](#).

To navigate to a member in the editor

1. Choose Navigate | File Structure on the main menu or press `Ctrl+F12`.
2. In the File Structure pop-up window that opens select the Narrow down the list on typing check box and start typing the desired member name. You can include the members, inherited from the parent classes, by checking the option Show inherited members or pressing `Ctrl+F12` again.



3. Use the navigation keys to select the desired node. Then do one of the following:
 - If the cursor rests on a top or intermediate node (for example, class `Ⓢ` or element `<>`), double-click this node or press `Enter` to expand it in the Structure pop-up, or press `F4` to jump to its declaration in the editor.
 - If the cursor rests on a leaf node (for example, a member `Ⓜ`, `ⓘ` or lowest-level element `<>`), double-click this node or press `Enter` to jump to its declaration in the editor.

In the case of an inherited member, the respective parent class opens in the editor.

You can also use the [Structure](#) tool window (`Alt+7`). This view is flexibly configurable and useful for many tasks, apart from navigation. However, the File Structure pop-up window is the easiest way for quick navigation.

Searching Through the Source Code

PyCharm provides extensive search and replace capabilities, which includes basic search and replace, search and replace in paths, finding usages, and more. Some of the replacement facilities (like renaming classes and members) are performed by means of [refactorings](#).

All the search commands can be found under the Find node of the Edit menu.

In this part you will learn how to:

- [Find and replace](#) text in the current file.
- [Find specific word](#).
- [Search and replace across the project](#).
- [Find usages](#).
- [Highlight usages](#).
- [View usages of a symbol across the project, and jump to the desired usage](#).
- [Work with the search results](#).

Finding and Replacing Text in File

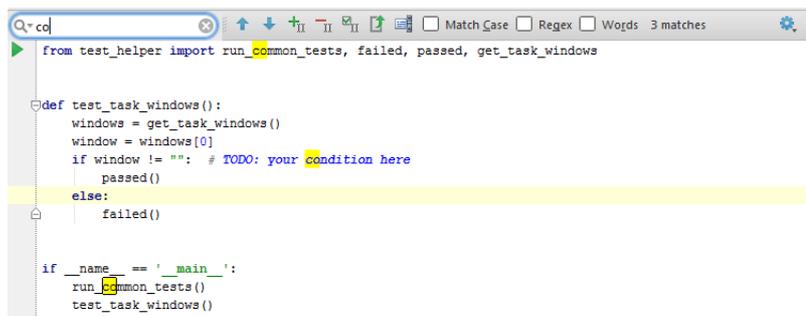
The standard facility helps you find and replace text strings in the active editor.

On this page:

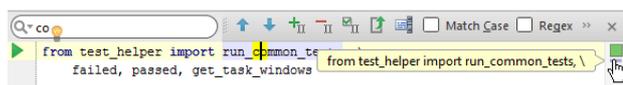
- [Search through the current file](#)
- [Replace in the current file](#)
- [Working with search results](#)
- [Search and replace options](#)

Search through the current file

1. From the main menu, choose Edit | Find | Find, or press `Ctrl+F`. The search pane appears on top of the active editor.
2. If necessary, specify the [search options](#).
3. In the search field, start typing the search string:



As you type, the first occurrence of the search string after the current cursor position is selected; the other occurrences are highlighted in the editor. In addition, the matching occurrences are marked in the right gutter with stripes.



4. [Explore the search results](#).

TIP To turn on the multiline mode, press `Alt+Enter`.

To return to a single-line mode, press `Delete`.

Replace in the current file

1. From the main menu, choose Edit | Find | Replace, or press `Ctrl+R`. The search and replace pane appears on top of the active editor.
2. If necessary, specify the [search and replace options](#).
3. In the search field, start typing the search string. As you type, the matching occurrences are highlighted in the editor, and a Replace pop-up dialog box opens at the first occurrence, suggesting you to replace the current occurrence, or all of them, with an empty string.
4. Start typing the replacing string.
5. [Explore the search results](#), and, using the buttons of the replace dialog box, replace occurrences as required. See [Search and replace options](#) below.

Working with search results

- To initiate a new search, do one of the following (depending on the current focus):

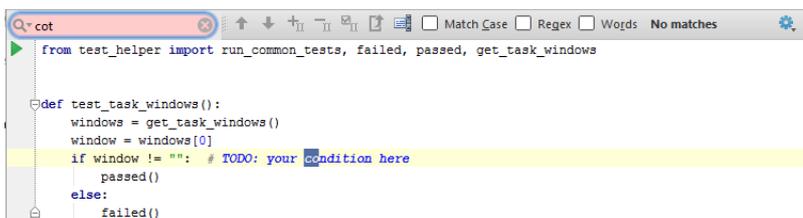
- If the editor has the focus, press `Ctrl+F`.
- If the search field has the focus, press `Ctrl+A`

In both cases, the existing search string will be selected, and you can start typing a new one.

- To jump between occurrences, do one of the following:

- Press `Shift+F3` (jump to previous selection) or `F3` (jump to next selection).
- Use the `↑` or `↓` buttons in the Search pane.
- Click the gutter stripes.

- The search pane shows the number of found occurrences. If no matches are found, the search pane becomes red:



- Use the recent search history: with the search pane already open, click `Q` to show the list of recent entries.

- Use Code completion in the Find and Replace panes. Start typing a search string, press `Ctrl+Space`, and select the appropriate word from the suggestion list.

- With the Find and Replace pane already opened, use `Ctrl+R` or `Ctrl+F` to toggle between panes. Thus, the search and replace strings are

preserved.

- To cancel the operation and close the pane, press `Escape`.
- Use **multiple selection (multiselection)**. For example, if a certain string has been highlighted as a search result, it is possible to add an occurrence of this string to multiple selection by clicking `⌘` (`Alt+J`), delete an occurrence from multiple selection using `⇧` (`Shift+Alt+J`), or add all found occurrences to multiple selection using `⌘` (`Ctrl+Shift+Alt+J`).

Search and replace options

Item	Description	Search/Replace
	Click this button to show the history of the recent entries.	Search, replace
	Click this button to clear the search field.	Search, replace
	Click these buttons to navigate through the occurrences of the search string.	Search, replace
	Click this button to add the next found occurrence to a multiple selection.	Search
	Click this button to remove the found occurrence from a multiple selection.	Search
	Click this button to create a selection that contains all the found occurrences.	Search
	Click this button to show search results in the Find tool window .	Search, replace
Match Case	If this check box is selected, PyCharm will distinguish between upper and lowercase letters while searching.	Search, replace
Regex	If this check box is selected, the search string will be perceived as a regular expression , and the replacement preview is shown in a tooltip (Refer to the section Example).	Search, replace
Words	if this check box is selected, PyCharm will search for whole words only, that is, for character strings separated with spaces, tabs, punctuation, or special characters. This check box is disabled, if the Regular expressions check box is selected.	Search, replace
Preserve Case	If this check box is selected, PyCharm retains the case of the first letter and the case of the initial string in general. For example, <i>MyTest</i> will be replaced with <i>Yourtest</i> if you specify <i>yourtest</i> as the replacement. This check box is disabled, if Regular expressions check box is selected.	Replace
In Selection	If this check box is selected, search and replacement will be confined to the selected text only.	Replace
Replace	Click this button to replace the current occurrence and proceed to the next one.	Replace
Replace all	Click this button the replace all found occurrences in the current file, or in the selection.	Replace
Exclude/Include	Click Exclude button to skip the current occurrence and exclude it from the Replace all operation. The button for this occurrence changes to Include.	Replace
	Click this button to invoke the list of additional options. Checking the corresponding option confines the search to the specified scope, while the other occurrences are ignored.	Search, replace

Finding and Replacing Text in Project

Introduction

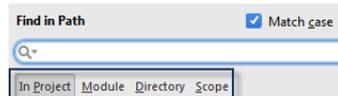
PyCharm extends search and replace capability to the entire project, or any directory with its nested hierarchy. Explore search results in the preview tab or in the [Find tool window](#).

Finding a piece of text in all the files within the specified path

1. On the main menu, choose Edit | Find | Find in Path, or press `Ctrl+Shift+F`.
2. In the Find In Path dialog, specify the following options:
 - Start entering the text. Type the text explicitly, or specify a pattern using a regular expression, or select a previously used piece of text or a pattern from the recent entries' drop-down list.

If you specify the search pattern through a regular expression, use the `$n` format in back references (to refer to a previously found and saved pattern).

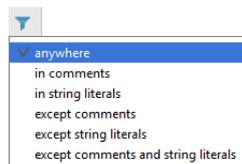
- Search scope (project, directory, or custom scope).



- Search options (case sensitivity, whole words, and regular expressions).



- Context search options.



The results are displayed in the preview area.

3. Edit the selected result right in the preview editor as it is a functional editor where actions such as *Find* (`Ctrl+F`) or *Find Next* (`F3`) are available without leaving the Find in Path window.

Search Result	File Name
class Mammalia(object):	Mammalia.py 4
class Marsupialia(Mammalia):	Mammalia.py 17
class Placentalia(Mammalia):	Mammalia.py 22
import Mammalia	Carnivorae.py 1
class Carnivorae(Mammalia):	Carnivorae.py 3
import Mammalia	Herbivorae.py 1

```
Animals/Mammalia.py
2 import Herbivorae
3
4 class Mammalia(object):
5     extremities = 4
6
7     def feeds(self):
8         print ("milk")
9
10    def proliferates(self):
11        pass
12
13    def sound(self):
14        pass
15
16
17 class Marsupialia(Mammalia):
18     def proliferates(self):
19         print ("poach")
```

Press Enter to open the selected result in the editor.

Click Open in Find Window (`Ctrl+Enter`) to see all of the results in the Find tool window.

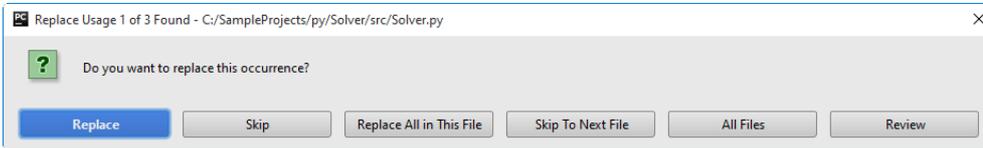
If the search takes too much time, click [Background](#) in the search progress window. In this case the search progress is indicated in the Status bar.

TIP When invoked for the second (and subsequent) time, the dialog opens with the scope that has been selected previously. For example, if the scope has been Directory, the next time you invoke the dialog, the scope again will be Directory.

Replacing a piece of text in all the files within the specified path

1. Do one of the following:
 - On the main menu, choose Edit | Find | Replace in Path.

- Press `Ctrl+Shift+R`
 - Being in the Find In Path dialog box, press `Ctrl+Shift+R` to switch to Replace In Path dialog box.
2. In the Replace In Path dialog, specify the search and replace strings, the search options, and the scope. Type the search and replacement text explicitly, or specify patterns using regular expression, or select a previously used piece of text or a pattern from the recent history drop-down list.
- If you specify the search and/or replacement text through a regular expression, use the `$n` format in back references (to refer to a previously found and saved pattern).
 - To use a backslash character `\` in a regular expression, escape the meaningful backslash by inserting **three extra backslashes** in preposition: `\\`.
3. Click Replace in Find Window. PyCharm displays the encountered occurrences of the search string in the [Find tool window](#), selects the first occurrence and opens the file with this occurrence in the editor and moves the focus to it.
- At the same time, PyCharm opens the Replace Usage dialog box, with the full path to the encountered occurrence in the title bar:



Do one of the following:

- To have the selected occurrence replaced, click Replace.
- To preserve the selected occurrence and move to the next one, click Skip.
- To have all the occurrences of the search string in the currently active tab replaced, click Replace All in This File.
- To preserve the occurrences of the search string in the currently active tab (any) and move to the next file, click Skip to Next File.
- To have all the detected occurrences replaced, click All Files.
- To switch to the manual mode, click Preview. The Replace Usage dialog box closes and the focus moves to the Find tool window. Do one of the following:
 - Browse through the list of detected occurrences, select the ones you want to replace and then click Replace Selected.
 - To have all the occurrences changed click Replace All.

Tip To learn about changing case of the literals, refer to the [example](#), and to [Regular Expression Syntax Reference](#).

Toggleing between the Find and Replace

- To switch from the Find In Path to Replace In Path window, press `Ctrl+Shift+R`.
- To switch from the Replace In Path to Find In Path window, press `Ctrl+Shift+F`.

Finding Usages

Search for usages is an important part of the code analysis, which enables you to clarify dependencies. PyCharm suggests several types of search for usages:

- [Finding Usages in Project](#)
- [Finding Usages in the Current File](#)
- [Highlighting Usages](#)
- [Viewing Recent Find Usages](#)
- [Viewing Usages of a Symbol](#)

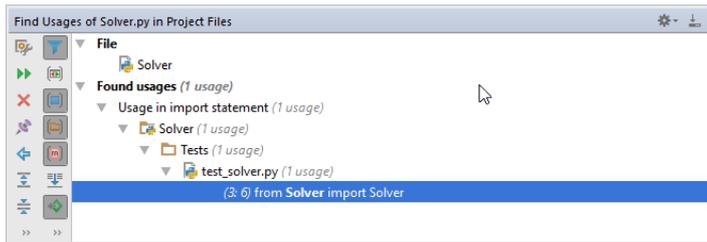
Finding Usages in Project

PyCharm provides different search options depending on whether you are searching for usages of a class, method, field, parameter, or throw statements, and extends search for usages to the files in supported languages. For example, in CSS, XML and HTML files you can search for the usages of styles, classes, tags and attributes.

Explore search results in the [Find tool window](#).

Finding usages of a symbol in a project

1. Select a symbol to find usages for. To do that, place the caret within the desired symbol in the editor, or click the symbol in the [Project tool window](#). You can also select symbol in the Model Dependency diagram.
2. Do one of the following:
 - On the main menu, choose Edit | Find | Find Usages
 - Choose Find Usages on the context menu
 - Press `Alt+F7`.
3. In the [Find tool window](#), explore search results. Use the  button to represent search results in meaningful groups by type of usage.



While analyzing the search results, you can at any time open the [search options dialog box](#) by clicking click  in the [Find tool window](#) or by pressing

`Ctrl+Shift+Alt+F7`.

To find usages of a symbol in the current file

1. Click the desired symbol in the editor, or in the Structure view.
2. On the main menu, choose Edit | Find | Find Usages in File, or press `Ctrl+F7`. The encountered usage is highlighted in the editor.

Highlighting Usages

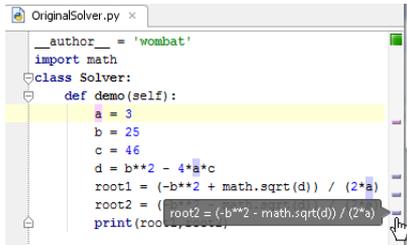
On this page:

- [Introduction](#)
- [Activating automatic highlighting of usages](#)
- [Highlighting usages of a symbol in the current file](#)
- [Navigating among usages](#)
- [Removing highlighting](#)

Introduction

The search command Highlight Usages in File ([Ctrl+Shift+F7](#)) makes it possible to visualize usages of a symbol in the current file.

All found usages of a symbol in the current file are highlighted and color-coded, as defined in the [Colors and Fonts](#) settings page, to represent read or write access to the symbol. In addition to the highlights of occurrences in text, the stripes of the same colors appear in the marker bar, accompanied with tooltips.



The behavior of usage highlighting is configurable: you can make PyCharm show usages of a symbol at caret automatically, or invoke it with a command.

Activating automatic highlighting of usages

1. Open the Settings dialog box (File | Settings for Windows and Linux or PyCharm | Preferences for macOS), and click General under the Editor node.
2. On the [General](#) page that opens, select the Highlight usages of element at caret check box in the Highlight on Caret Movement area.

Highlighting usages of a symbol in the current file

1. Place the caret at the selected symbol in the editor. If automatic usages highlighting is enabled, see all its occurrences in the current file highlighted. Otherwise, proceed to the next step.
2. On the main menu, choose Edit | Find | Highlight Usages in File, or press [Ctrl+Shift+F7](#).

Navigating among usages

To navigate among usages, do one of the following:

- Click on a stripe in the marker bar to navigate to the respective usage location.
- Use the [F3](#) and [Shift+F3](#) keyboard shortcuts to navigate to the next and previous usages respectively.

Removing highlighting

To remove highlighting of usages, press [Escape](#).

With PyCharm you can easily navigate to the recent search results if any find usages action took place during the PyCharm session.

To view recent find usages

1. Select Edit | Find | Recent Find Usages on the main menu.
2. Select the required search item from the submenu of the recent search results:



The selected search shows in the Find tool window.

Viewing Usages of a Symbol

Using the Show Usages function, you can bring up a list of the usages of a symbol across the whole project. So doing, the pop-up window with the list of usages of a symbol features a toolbar with the following buttons:

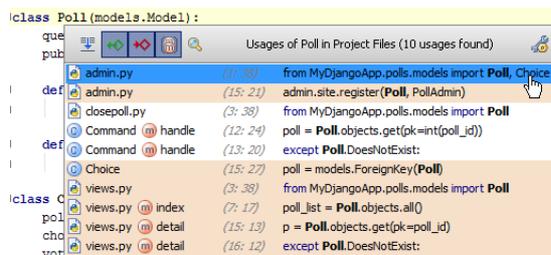
Icon Shortcut Description

Icon	Shortcut	Description
	Ctrl+F	Merge usages of the symbol from the same line.
	Ctrl+R	Show read access to the symbol.
	Ctrl+W	Show write access to the symbol.
	Ctrl+I	Show usages in the import statements.
	Ctrl+M	If this button is pressed, the found usages show under the corresponding method nodes.
	Alt+F7	Show search results in the Find Usages tool window.
	Ctrl+Shift+Alt+F7	Open the Find Usages dialog box for the selected symbol where you can change the search options.

Tip In addition to the ability of viewing usages, you can use this function as a quick means of navigation.

To view the usages of a symbol across the project

1. Place the caret at the desired symbol in the editor.
2. On the main menu, choose Edit | Find | Show Usages, or press `Ctrl+Alt+F7`.
3. Examine and analyze the detected occurrences of the symbol:
 - Use the toolbar buttons to present search results in the desired way.
 - To jump from search results to a line of source code, click the desired entry.
 - To close the list, press `Escape`.



4. If necessary, customize the search options in the [search options dialog box](#). To invoke the dialog box, do one of the following:
 - In the Show Usages pop-up window, click .
 - Press `Ctrl+Shift+Alt+F7`.

Structural Search and Replace

Structural Search and Replace (SSR) performs search and replace in the supported languages code across the whole project, taking advantage of the PyCharm's awareness of the syntax and code structure of the supported languages.

PyCharm finds, and if required, replaces fragments of source code, using the [search templates](#).

Structural Search and Replace is helpful when you need to browse through or modify an extensive code base, find changes in libraries, explore the source code for specific constructs, or refactor the source code.

Tip To see the list of supported languages, open the dialog box [Structural Search and Replace Dialogs](#) and click the drop-down list File type.

In this section:

- [Search Templates](#)
- [Structural Search and Replace - General Procedure](#)
- [Creating and Editing Search Templates](#)
- [Structural Search and Replace Examples](#)

Search Templates

Search templates are the essential part of [Structural Search and Replace](#) feature. Like [Live templates](#), the search templates consist of plain text and one or more **template variables**.

A valid search or replacement template represents one of the following supported languages constructs:

- Expression, for example `"John" + " " + "Doe"`
- Statement, or sequence of statements, for example `document.getElementById("demo").innerHTML = "Hello Dolly.";`
- Class designator, for example `class Engine implements IEngine`
- Line or block comments, for example `/** Created in PyCharm */`.

PyCharm provides a collection of predefined search templates, that match the various statements, expressions, classes and their members, XML and HTML constructs, and more. You can use these templates for structural search and replace, and also as a basis for creating your own search templates.

The search templates make use of the variables, which are the strings surrounded with `$` characters, for example `$expression$`. Symbols in the source code, `String` literals, and comments can be referred to by means of variables.

Variables in a template are subjected to certain constraints that help you refine your search and limit it to the desired matches:

- **Text constraints** are text patterns to match against. These constraints can be plain text, or regular expressions, and can contain references to symbols.
- **Number of occurrences** defines how many sequential elements (in a parameter, declaration or statement list) a variable can include and whether a variable is required to be present in a pattern or not. If the number of occurrences is 1, only one symbol can match the variable. If the number of occurrences is null, it means that an element could be missing.
- **Expression constraints** apply semantic conditions to the search, for example locate the symbols that are read or written to.
- **Script constraints** are used when items to search for are more than a plain match. If you are looking for certain language constructs (for example, constructors with the specified number of parameters, or members with the specified visibility modifiers), apply constraints described as Groovy scripts.

In search templates, the following simplifications can be used:

- Method body can be omitted.
- Short class names (instead of fully qualified names) are used in the templates and constraint fields.
- Using `class $Class$` as a template, results in finding anonymous classes as well.
- Templates for comments and documentation comments should contain variables and constructs with correct comment and JSDoc syntax.

This section outlines the general SSR procedure. Refer to the section [Structural Search and Replace Examples](#) for typical use cases.

To find and replace source code structurally, follow these general steps:

1. On the main menu, choose Edit | Find | Search Structurally, or Edit | Find | Replace Structurally.
2. In the dialog box that opens, define the [search template](#). In brief, defining a search template involves the following steps:
 - Type the desired construct in the Search template text area, or use one of the pre-defined search templates by clicking the Copy existing template button.
 - Specify the constraints to be imposed on the variables within the search template. To do that, click the Edit variables button. All the variables contained in the search template are listed in the Variables pane of the [Edit Variable](#) dialog box.

Refer to the section [Creating and Editing Search Templates](#) for the detailed description of procedure.

3. In case of the structural replace, specify the replacement pattern, and define variable constraints as required.
4. Specify the search and replace options, in particular, the number of occurrences to be matched, and the type of files to be analyzed.
5. Specify the scope to perform the structural search and replace in. To do that, click the down arrow in the Scope list, and select one of the pre-defined scopes, or click the ellipsis button, and configure the desired scope in the [Scopes](#) dialog box.
6. Click Find. The detected occurrences are displayed in the [Find tool window](#).
Please note that in case of replacement, you can select the desired matches in the search results and click the Preview Replacement button. The corresponding occurrence is highlighted in the source code.

Creating and Editing Search Templates

You can create [search and replace templates](#) from scratch, just typing the code in the text area of the [Structural Search / Replace](#) dialog box. However, there is a collection of the predefined search templates that you can use as prototypes for your own templates. All custom templates appear in the list of existing search templates, under the node User defined.

To create a search template, follow these general steps:

- On the main menu, choose Edit | Find | Search Structurally.
 - Do one of the following:
 - Type the code of your template in the Search template text area.
 - Click the Copy existing template button, and in the Existing Templates dialog box, select the desired template as a prototype. The source code of the selected template appears in the Search template text area, where you can change it as required.
 - If you need to configure the template variables, click the Edit variables button. [Edit Variables](#) dialog box appears.
In the Variables column, select a variable you want to configure, and specify the constraints that will apply to this variable. See the dialog [reference page](#) for the detailed description of constraints.
- Repeat the process for the other variables, as required, apply changes and close the dialog box.
- Click the Save Template button.
 - In the Save Template dialog, type the name of the new template, and click OK.

Tip You can create a new template in the Existing Templates dialog. To do that, click the  button on the toolbar. This opens the Structural Search dialog, with the empty template field. To define the custom template, follow the steps described above.

On this page:

- [Method call](#)
- [Searching for XML and HTML tags, attributes, and their values](#)

Method call

```
$Instance$.MethodCall($Arguments$)
```

This template matches method call expressions. If the number of occurrences is zero, it means that a method call can be omitted.

Searching for XML and HTML tags, attributes, and their values

The simplest template to search for a tag is `<a />`

By placing constraints on the variable `a`, you can specify which tags you want to find. For example, if you specify the text/regexp constraint `app.+`, you'll find the tags whose names start with `app`.

A more versatile template for searching in XML and HTML is `<tag $attribute="$value$"/>`. By using this template with property specified search settings and constraints, you can find practically anything that may occur in XML or HTML. For example, if you specify the text/regexp constraint `width` for the variable `$attribute$`, you'll find all the tags that have the `width` attribute.

Finding Word at Caret

This command lets you find occurrences of the current word independently on its structural meaning - it can be anything in your document: an identifier or a keyword in the code, a word in a string literal or a comment, an XML tag or attribute, or even a number. If any matches are found, you can quickly navigate between them.

Note that this command is case-sensitive.

To find the word at caret, do one of the following

- On the main menu, choose Edit | Find | Find Word At Caret.
- Use `Ctrl+F3` keyboard shortcut.

To navigate between the occurrences of the word at caret

1. Press `F3` to go to the next occurrence.
2. Press `Shift+F3` to go to the previous occurrence.

Working With Search Results

The search results display in the [Find tool window](#). The results of each search display in a separate tab.

Using the controls and the context menu commands of this tool window as well as the main menu, you can:

- Navigate [to source code](#).
- [Exclude and include](#) search results in refactoring.
- Add selected search results [to favorites](#).
- Show the results of the [recent find usages](#).
- Perform [version control](#) operations using the specific VCS, associated with the parent directory of the usage.
- [Hide excluded search results](#) from showing them in the Find tool window ([Alt+Delete](#)).

Searching Everywhere

On this page:

- [Introduction](#)
- [Searching everywhere](#)
- [Configuring search everywhere scopes](#)

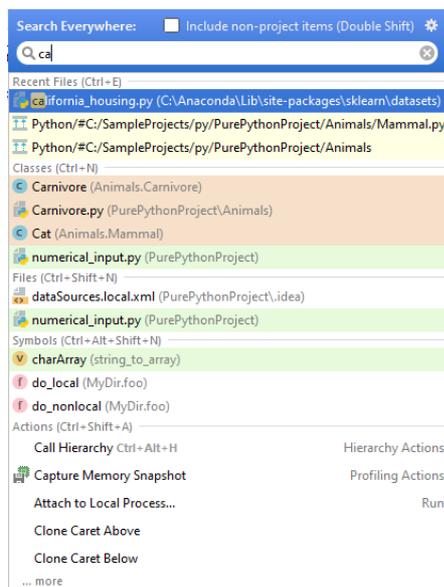
Introduction

PyCharm makes it possible to look for any item of the source code, databases, actions, elements of the user interface, etc. in a single action.

Searching everywhere

To search everywhere, follow these steps

1. Do one of the following:
 - Click  in the upper-right corner of the PyCharm window.
 - Double-press Shift
2. In the pop-up window that opens, start typing the search string, to narrow down the suggestion list:



The search string can be a name of a symbol or action, an option, or an [abbreviation](#) of an action.

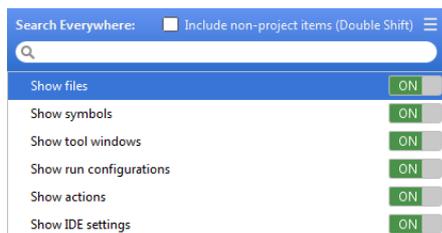
3. Select the desired target from the suggestion list. The further behavior depends on the search target. PyCharm may silently navigate to the desired source code, show the desired setting, or a pop-up list of Find Action.
You can also just press the desired shortcut, to switch from the Search everywhere pop-up window to any other navigation and search action: for example, press Ctrl+N to jump to the selected class, or Ctrl+Alt+S to show the corresponding option in the Settings dialog.

Tip PyCharm allows viewing the history of the previous searches with the help of the **left arrow** key. Search Everywhere remembers last 50 queries finalized with Enter key.

Configuring search everywhere scopes

To configure search everywhere scopes

1. In the Search Everywhere dialog, click .
2. Turn on or off the desired search scopes:



Refactoring Source Code

PyCharm offers a wide variety of code refactorings, which track down and correct the affected code references automatically.

In this part you will find:

- [General refactoring procedure](#)
- [Procedures and examples of the available refactorings](#)

To perform refactoring, follow these general steps

1. Select (or hover caret on) a symbol or code fragment to refactor. The set of available refactorings depends on your selection. You can select symbols in the following PyCharm components:
 - Project view
 - Structure tool window
 - Editor
 - UML Class diagram
2. Do one of the following:
 - On the main Refactor menu or on the context menu of the selection, choose the desired refactoring or press the corresponding keyboard shortcut (if any).
 - On the main menu, choose Refactor | Refactor This, or press `Ctrl+Shift+Alt+T`, and then select the desired refactoring from the pop-up window.
3. In the dialog box that opens, specify the refactoring options.
4. To apply the changes immediately, depending on the refactoring type, click Refactor or OK.
5. For certain refactorings, there is an option of previewing the changes prior to actually performing the refactoring. In such cases the Preview button is available in the corresponding dialog.
To preview the potential changes and make the necessary adjustments, click Preview. PyCharm displays the changes that are going to be made on a dedicated tab of the [Find tool window](#).

One of the possible actions at this step is to exclude certain entries from the refactoring. To do so, select the desired entry in the list and press `Delete`.

If conflicts are expected after the refactoring, PyCharm displays a dialog with a brief description of the encountered problems. If this is the case, do one of the following:

- Ignore the conflicts by clicking the Continue button. As a result, the refactoring will be performed, however, this may lead to erroneous results.
 - Preview the conflicts by clicking the Show in View button. PyCharm shows all conflicting entries on the Conflicts tab in the [Find tool window](#), enabling you to navigate to the problematic lines of code and to make the necessary fixes.
 - Cancel the refactoring and return to the editor.
6. When you are satisfied with the proposed results, click Do Refactor to apply the changes.

PyCharm provides the following common refactorings:

- [Change Signature](#)
- [Copy](#)
- [Extract Refactorings](#)
- [Inline](#)
- [Invert Boolean](#)
- [Move Refactorings](#)
- [Push Members Down](#)
- [Pull Members Up](#)
- [Rename Refactorings](#)
- [Safe Delete](#)

The following language-specific refactorings are available:

- [Change Signature in JavaScript](#)
- [Extract Parameter in JavaScript](#)
- [Extract Variable in JavaScript](#)

Change Signature

In this section:

- [Basics](#)
- [Examples](#)
- [Default value and propagation of new parameters](#)
- [Changing a function signature](#)

Basics

The Change Signature refactoring combines several different modifications that can be applied to a function signature. You can use this refactoring for the following purposes:

- To change the function name.
- To add new parameters and remove the existing ones.
- To assign default values to the parameters.
- To reorder parameters.

When changing a function signature, PyCharm searches for all usages of the function and updates all the calls, implementations, and override replacements of the function that can be safely modified to reflect the change.

Examples

Before/After

<pre># This function will be renamed: def fibonacci(n): a, b = 0, 1 while b < n: print(b) a, b = b, a+b n = int(input("n = ")) fibonacci(n)</pre>	<pre># Function with the new name: def fibonacci_numbers(n): a, b = 0, 1 while b < n: print(b) a, b = b, a+b n = int(input("n = ")) fibonacci_numbers(n)</pre>
<pre># New parameters will be added: def fibonacci(n): a, b = 0, 1 while b < n: print(b) a, b = b, a+b n = int(input("n = ")) fibonacci(n)</pre>	<pre># Function with the new parameters. # Do not forget to specify the default values of the parameters, which will be used in the function call. def fibonacci(n,a,b): a, b = 0, 1 # this should be done manually! while b < n: print(b) a, b = b, a+b n = int(input("n = ")) fibonacci(n,0,1)</pre>

Default value and propagation of new parameters

For each new parameter added to a function, you can specify:

- A default value (or an expression) (the Default value field).

You can also propagate the parameters you have introduced to the functions that call the function whose signature you are changing.

The refactoring result depends on whether you specify the default value and whether you use propagation.

Propagation. New parameters can be propagated to any function that calls the function whose signature you are changing. In such case, generally, the signatures of the calling functions change accordingly.

These changes, however, also depend on the default values set for the new parameters .

Default value. Generally, this is the value to be added to the function calls.

If the new parameter is not propagated to a calling function, the calls within such function will also use this value.

If the propagation is used, this value won't matter for the function calls within the calling functions.

Changing a function signature

1. In the editor, place the cursor within the name of the function whose signature you want to change.
2. Do one of the following:
 - Press `Ctrl+F6`.
 - Choose Refactor | Change Signature on the main menu or .
 - Choose Refactor | Change Signature on the context menu.
3. In the Change Signature dialog, make the necessary changes to the function signature and specify what other, related, changes are required. You can:

Change the function name. To do that, edit the text in the Name field

_ Change the function name. To do that, edit the text in the Name field.

– Manage the function parameters using the table and the buttons in the Parameters area:

- To add a new parameter, click **+** and specify the properties of the new parameter in the corresponding table row.
- To remove a parameter, click any of the cells in the corresponding row and click **-**.
- To reorder the parameters, use the **↑** and **↓** buttons. For example, if you wanted to put a certain parameter first in the list, click any of the cells in the row corresponding to that parameter, and then click **↑** the required number of times.
- To change the name or the default value of a parameter, make the necessary updates in the table of parameters (in the fields Name and Default value respectively).

4. To perform the refactoring right away, click Refactor.

To [see the expected changes](#) and make the necessary adjustments prior to actually performing the refactoring, click Preview.

Note Code completion is available in the Default value field of the table in the Parameters area.

Copy

– [Basics](#)

– [Performing Copy refactoring](#)

Basics

The Copy refactoring lets you create a copy of a class, file or directory in a different or the same directory.

Performing Copy refactoring

To perform the Copy refactoring

1. Select the item of interest in a tool window (e.g. the Project tool window). Alternatively, open the necessary class or file in the editor.
2. Do one of the following:
 - Choose Refactor | Copy on the main menu or the context menu.
 - Press **F5**.
 - In the Project tool window, press and hold the **Ctrl** key, and drag the item to the target location.
3. In the [Copy dialog](#) that opens, specify the name and location for the copy that you are creating, and click OK.

Extract Refactorings

In this section:

- [Extract Constant](#)
 - [Extract Field](#)
 - [Extract Method](#)
 - [Extract Parameter](#)
 - [Extract Superclass](#)
 - [Extract Variable](#)
- The way to extract constant, field, and variable refactorings are performed, depends on the setting of the Enable in-place refactoring check box in the [Editor](#) page of the Settings/Preferences dialog.
- Extract refactorings are performed for the various expressions and blocks of code, including strings and substrings. Extract refactoring on substrings is supported for:
- parts of strings with no formatting
 - parts of strings with concatenation via "+"
 - parts of strings with "%"-style formatting
 - parts of strings with "%()"-style formatting
 - new-style `str.format()` formatted strings

Extract Constant

The **Extract Constant** refactoring makes your source code easier to read and maintain. It also helps you avoid using hard-coded constants without any explanations about their values or purpose.

On this page:

- [Example](#)
- [Extracting a Python constant in-place](#)
- [Extracting a constant using the dialog box](#)

Example

Before

```
import math

class Solver:
    def demo(self):
        a = 3
        b = 25
        c = 46
        root1 = (-b + math.sqrt(b**2 - 4*c)) / (2*a)
        root2 = (-b - math.sqrt(b**2 - 4*c)) / (2*a)
        print (root1, root2)

Solver().demo()
```

```
def print_hello(self):
    s = "Hello, World!"
    print(s)
```

```
import math
RETURN_TYPE_OF_SQRT = math.sqrt(b**2 - 4*a*c)

class Solver:
    def demo(self):
        a = 3
        b = 25
        c = 52
        root1 = (-b + RETURN_TYPE_OF_SQRT) / (2*a)
        root2 = (-b - RETURN_TYPE_OF_SQRT) / (2*a)
        print(root1, root2)

Solver().demo()
```

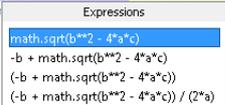
```
def print_hello(self):
    s = "s, World!"
    print(s)
```

To extract a Python constant in-place

The **in-place refactorings** are enabled in PyCharm by default. So, if you haven't changed this setting, the Extract Constant refactoring for Python is performed in-place, right in the editor.

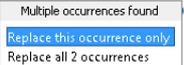
1. Place the cursor within the expression or declaration of a variable to be replaced by a constant.
2. Do one of the following:
 - Press **Ctrl+Alt+C**.
 - Choose Refactor | Extract | Constant on the main menu or on the context menu.
3. If more than one expression is detected for the current cursor position, the Expressions list appears. If this is the case, select the required expression. To do that, click the expression. Alternatively, use the **Up** and **Down** arrow keys to navigate to the expression of interest, and then press **Enter** to select it.

```
c = 46
root1 = (-b + math.sqrt(b**2 - 4*a*c)) / (2*a)
root2 = (-b - math.s
print(root1,root2)
```



4. If more than one occurrence of the expression is found within the class, specify whether you wish to replace only the selected occurrence, or all the found occurrences with the new constant.

```
c = 46
root1 = (-b + math.sqrt(b**2 - 4*a*c)) / (2*a)
root2 = (-b - math.sq
print(root1,root2)
```



5. Specify the name of the constant. Select the name from the list or type the name in the box with a red border.

```
RETURN_TYPE_OF_SQRT = math.sqrt(b ** 2 - 4 * a * c)
__author__ = 'wombat'
import math
class SolverEquation:
    def demo(self):
        a = 3
        b = 25
        c = 46
        root1 = (-b + RETURN_TYPE_OF_SQRT) / (2*a)
        root2 = (-b - RETURN_TYPE_OF_SQRT) / (2*a)
        print(root1, r
```



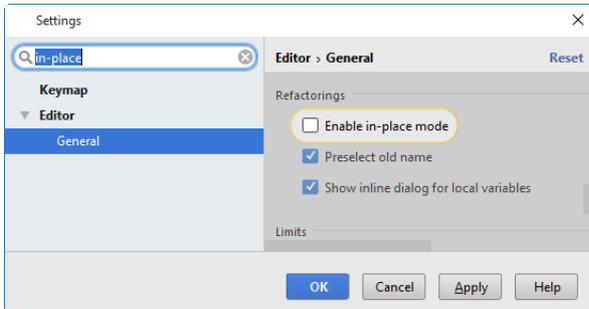
6. To complete the refactoring, press **Tab** or **Enter**.

If you haven't completed the refactoring and want to cancel the changes you have made, press **Escape**.

To extract a constant using the dialog box

If the Enable in-place refactorings check box is cleared on the Editor settings, the Extract Constant refactoring is performed by means of the

Extract Constant Dialog dialog box.



1. In the [Extract Constant Dialog](#) dialog that opens, specify the name of the new constant.
2. To automatically replace all occurrences of the selected expression (if it is found more than once), select the option `Replace all occurrences`.
3. Click OK to create the constant.

Extract Field

Extract Field refactoring declares a new field and initializes it with the selected expression. The original expression is replaced with the usage of the field.

- [Example](#)
- [Extracting a field in-place](#)
- [Extracting a field using the dialog box](#)

Example

Before/After

```
import math

class Solver:
    def roots(self):
        a = 3
        b = 25
        c = 46
        root1 = (-b + math.sqrt(b**2 - 4*c)) / (2*a)
        root2 = (-b - math.sqrt(b**2 - 4*c)) / (2*a)
        print (root1, root2)

Solver().demo()
```

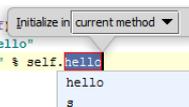
```
import math

class Solver:
    def demo(self):
        a = 3
        b = 25
        c = 46
        self.return_type_of_sqrt = math.sqrt(b ** 2 - 4 * a * c)
        root1 = (-b + self.return_type_of_sqrt) / (2*a)
        root2 = (-b - self.return_type_of_sqrt) / (2*a)
        print(root1,root2)

Solver().demo()
```

```
def print_hello(self):
    s = "Hello, World!"
    print(s)
```

```
def print_hello(self):
    self.hello = "Hello"
    s = "%s, World!" % self.hello
    print(s)
```

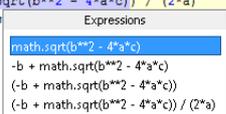


To extract a field in-place

The [in-place refactorings](#) are enabled in PyCharm by default. So, if you haven't changed this setting, the Introduce Field refactorings are performed in-place, right in the editor:

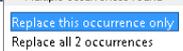
1. Place the cursor within the expression or declaration of a variable to be replaced by a field.
2. Do one of the following:
 - Press `Ctrl+Alt+F`.
 - Choose Refactor | Introduce Field on the main menu, or on the context menu.
3. If more than one expression is detected for the current cursor position, the Expressions list appears. If this is the case, select the required expression. To do that, click the expression. Alternatively, use the `Up` and `Down` arrow keys to navigate to the expression of interest, and then press `Enter` to select it.

```
c = 46
root1 = (-b + math.sqrt(b**2 - 4*a*c)) / (2*a)
root2 = (-b - math.s
print(root1,root2)
```



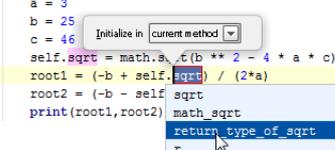
4. If more than one occurrence of the expression is found within the class, specify whether you wish to replace only the selected occurrence, or all the found occurrences with the new constant:

```
c = 46
root1 = (-b + math.sqrt(b**2 - 4*a*c)) / (2*a)
root2 = (-b - math.sq
print(root1,root2)
```



5. If relevant, specify where the new field will be initialized - in the current method, or in a class constructor.
6. Specify the name of the field. Select the name from the list or type the name in the box with a red border.

```
class SolverEquation:
    def demo(self):
        a = 3
        b = 25
        c = 46
        self.sqrt = math.a_f(b ** 2 - 4 * a * c)
        root1 = (-b + self.sqrt) / (2*a)
        root2 = (-b - self.sqrt
        print (root1,root2)
```



7. To complete the refactoring, press `Tab` or `Enter`.
- If you haven't completed the refactoring and want to cancel the changes you have made, press `Escape`.

Note that sometimes you may need to press the corresponding key more than once.

```

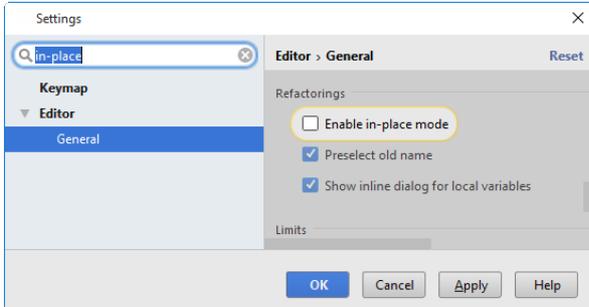
import math

__author__ = 'wombat'
class SolverEquation:
) def demo(self):
    a = 3
    b = 25
    c = 46
    self.return_type_of_sqrt = math.sqrt(b ** 2 - 4 * a * c)
    root1 = (-b + self.return_type_of_sqrt) / (2*a)
    root2 = (-b - self.return_type_of_sqrt) / (2*a)
    print(root1,root2)
)

```

To extract a field using the dialog box

If the Enable in place refactorings check box is cleared on the Editor settings, the Extract Field refactoring is performed by means of the [Extract Field Dialog](#).



1. In the editor, select the expression or variable to be replaced with a field, or just place the cursor within such an expression or variable declaration.
2. In the main menu, or the context menu of the selection, choose Refactor | Extract | Field, or press `Ctrl+Alt+F`.
3. In the Expressions pop-up menu, select the expression to be replaced. Note that PyCharm highlights the selected expression in the editor.
4. In the dialog that opens, specify the type and name of the new field.
5. In the Initialize in section, specify where the new field will be initialized.
6. To automatically replace all occurrences of the selected expression (if it is found more than once), select the option Replace all occurrences.
7. Click OK to create the field.

Extract Method

- Basics
- Examples
 - Python Extract Method example
- Extracting a method
- Processing duplicates

Basics

When the **Extract Method** refactoring is invoked, PyCharm analyses the selected block of code and detects variables that are the input for the selected code fragment and the variables that are output for it.

The detected output variable is used as a return value for the extracted function.

BeforeAfter

<pre>function MyFunction(a,b) { c = a + b; d = c * c; return d; }</pre>	<pre>function Sum(a,b) { return a + b; } function MyFunction(a,b) { c = sum(a,b); d = c * c; return d; }</pre>
---	---

Examples

Python Extract Method example

BeforeAfter

<pre>import math class Solver: def demo(self): a = 3 b = 25 c = 46 root1 = (-b + math.sqrt(b**2 - 4*a*c)) / (2*a) root2 = (-b - math.sqrt(b**2 - 4*a*c)) / (2*a) print(root1, root2) Solver().demo()</pre>	<pre>import math class Solver: def eval_roots(self, a, b, c): d = b**2 - 4*a*c root1 = (-b + math.sqrt(d)) / (2*a) root2 = (-b - math.sqrt(d)) / (2*a) return root1, root2 def demo(self): a = 3 b = 25 c = 52 root1, root2 = self.eval_roots(a, b, c) print(root1, root2) Solver().demo()</pre>
--	---

<pre>def print_hello(self): s = "Hello, World!" print(s)</pre>	<pre>def method_name(self): return "Hello, World!" def print_hello(self): s = self.method_name() print(s)</pre>
--	--

Extracting a method

To extract a method, follow these steps

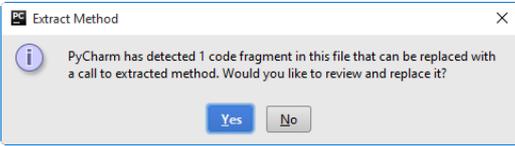
1. In the editor, select a block of code to be transformed into a method or a function.

Tip The code fragment to form the method does not necessarily have to be a set of statements. It may also be an expression used somewhere in the code.

2. On the main menu or on the context menu of the selection, choose Refactor | Extract | Method or press `Ctrl+Alt+M`.
3. In the Extract Method dialog box that opens, specify the name of the new function.
4. In the Parameters area, do the following:
 - Specify the variables to be passed as method parameters, by selecting/clearing the corresponding check boxes.
 - Rename the desired parameters, by double-clicking the corresponding parameter lines and entering new names.
5. Check the result in the Signature Preview pane and click OK to create the required function.
The selected code fragment will be replaced with a function call.

Processing duplicates

If **duplicate code fragments** are encountered, PyCharm suggests to replace them with the calls to the extracted method:



Extract Parameter

The Extract Parameter refactoring is used to add a new parameter to a function declaration and to update the function calls accordingly.

- [Example](#)
- [Extracting a parameter in Python in-place](#)
- [Extracting a parameter in Python using the Extract Parameter dialog](#)

See also [Change Signature](#).

Example

Before/After

<pre>def print_test(self): print "test"</pre>	<pre>def print_test(self, test): print test</pre>
<pre>def print_hello(self): s = "Hello, World!" print(s)</pre>	<pre>def print_hello(self, hello="Hello"): s = "%s, World!" % hello print(s) hello</pre>

Extracting a parameter in Python in-place

The [in-place refactorings](#) are enabled in PyCharm by default. So, if you haven't changed this setting, the Extract Parameter refactorings for Python are performed in-place, right in the editor:

1. In the editor, place the cursor within the expression to be replaced by a parameter.
2. Do one of the following:
 - Press `Ctrl+Alt+P`.
 - Choose Refactor | Extract | Parameter on the main menu.
 - Choose Refactor | Extract | Parameter from the context menu.
3. If more than one expression is detected for the current cursor position, the Expressions list appears. If this is the case, select the required expression. To do that, click the expression. Alternatively, use the `Up` and `Down` arrow keys to navigate to the expression of interest, and then press `Enter` to select it.
4. Type the parameter name in the box with a red border.
5. To complete the refactoring, press `Tab` or `Enter`.
If you haven't completed the refactoring and want to cancel the changes you have made, press `Escape`.

Note that sometimes you may need to press the corresponding key more than once.

Extracting a parameter in Python using the Extract Parameter dialog

To be able to use the Extract Parameter dialog (instead of performing the refactoring in-place), make sure that the [Enable in place refactorings](#) option is off in the editor settings.

Once this is the case, you perform the Extract Parameter refactoring as follows:

1. In the editor, place the cursor within the expression to be replaced by a parameter.
2. Do one of the following:
 - Press `Ctrl+Alt+P`.
 - Choose Refactor | Extract | Parameter on the main menu.
 - Choose Refactor | Extract | Parameter from the context menu.
3. If more than one expression is detected for the current cursor position, the Expressions list appears. If this is the case, select the required expression. To do that, click the expression. Alternatively, use the `Up` and `Down` arrow keys to navigate to the expression of interest, and then press `Enter` to select it.
4. In the [Extract Parameter](#) dialog that opens:
 1. Specify the parameter name in the Name field.
 2. If more than one occurrence of the expression is found within the function body, you can choose to replace only the selected occurrence or all the found occurrences with the references to the new parameter. Use the Replace all occurrences check box to specify your intention.
5. [Preview and apply changes](#).

Extract Superclass

- Basics
- Example
- Extracting a superclass

Basics

The Extract Superclass refactoring enables extracting certain methods of a class into a superclass.

Example

Before/After

```
class Solver1:
    def __init__(self, *args, **kwargs):
        self.sqrtd = math.sqrt(d)

    def eval_roots(self, a, b, c):
        d = b**2 - 4*a*c
        root1 = (-b + self.sqrtd) / (2*a)
        root2 = (-b - math.sqrt(d)) / (2*a)
        return root1, root2

class eval:
    def eval_roots(self, a, b, c):
        d = b**2 - 4*a*c
        root1 = (-b + self.sqrtd) / (2*a)
        root2 = (-b - math.sqrt(d)) / (2*a)
        return root1, root2

class Solver1(eval):
    def __init__(self, *args, **kwargs):
        self.sqrtd = math.sqrt(d)

    def demo(self):
        b = 25
        c = 52
        a = 3
        root1, root2 = self.eval_roots(a, b, c)
        print(root1, root2)
```

To extract a superclass

1. On the on the main menu or on the context menu, choose Refactor | Extract | Superclass.
2. In the [Extract Superclass](#) dialog box that appears, specify the following information:
 - Name of the new superclass.
 - Target directory where it will be stored.
 - Methods to be included in the superclass.
3. Proceed with the refactoring.

Extract Variable

- [Basics](#)
- [Python Example](#)
- [Extracting variable in-place](#)
- [Extracting variable with a dialog](#)

Basics

The Extract Variable refactoring puts the result of the selected expression into a variable. It declares a new variable and uses the expression as an initializer. The original expression is replaced with the new variable ([see the examples below](#)).

To perform this refactoring, you can use:

- [In-place refactoring](#). In this case you specify the new name right in the editor.
- [Refactoring dialog](#), where you specify all the required information. To make such a dialog accessible, you have to clear the check box [Enable in-place mode](#) in the editor settings.

You can select the expression to be replaced with a variable yourself. You can as well use [smart expression selection](#). In this case PyCharm will help you select the desired expression.

This refactoring is also available for [JavaScript](#) and [Sass](#).

Python Example

Before/After

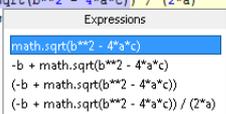
<pre>import math class Solver: def demo(self): a = 3 b = 25 c = 46 root1 = (-b + math.sqrt(b**2 - 4*a*c)) / (2*a) root2 = (-b - math.sqrt(b**2 - 4*a*c)) / (2*a) return root1, root2 Solver().demo()</pre>	<pre>import math class Solver: def demo(self): a = 3 b = 25 c = 46 return_type_of_sqrt = math.sqrt(b ** 2 - 4 * a * c) root1 = (-b + return_type_of_sqrt) / (2*a) root2 = (-b - return_type_of_sqrt) / (2*a) print(root1,root2) Solver().demo()</pre>
<pre>def print_hello(self): s = "Hello, World!" print(s)</pre>	<pre>def print_hello(self): hello = "Hello" s = "%s, World!" % hello print(s)</pre>

Extracting variable in-place

To extract a variable using in-place refactoring, follow these steps

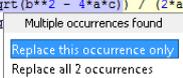
1. In the editor, select the expression to be replaced with a variable. You can do that yourself or use the [smart expression selection](#) feature to let PyCharm help you. So, do one of the following:
 - Highlight the expression. Then choose Refactor | Extract | Variable on the main menu or on the context menu. Alternatively, press `Ctrl+Alt+V`.
 - Place the cursor before or within the expression. Choose Refactor | Extract Variable on the main menu or on the context menu. or press `Ctrl+Alt+V`.

```
c = 46
root1 = (-b + math.sqrt(b**2 - 4*a*c)) / (2*a)
root2 = (-b - math.s
print(root1,root2)
```



2. If more than one occurrence of the selected expression is found, select Replace this occurrence only or Replace all occurrences in the Multiple occurrences found pop-up menu. To select the required option, just click it. Alternatively, use the Up and Down arrow keys to navigate to the option of interest, and press `Enter` to select it.

```
c = 46
root1 = (-b + math.sqrt(b**2 - 4*a*c)) / (2*a)
root2 = (-b - math.sq
print(root1,root2)
```



3. Specify the name of the variable. Do one of the following:
 - Select one of the suggested names from the pop-up list. To do that, double-click the suitable name. Alternatively, use the Up and Down arrow keys to navigate to the name of interest, and `Enter` to select it.

```

a = 3
b = 25
c = 46
sqrt = math.sqrt(b ** 2 - 4 * a * c)
root1 = (-b + sqrt) / (2*a)
root2 = (-b - sqrt) / (2*a)
print(root1, root2)

```

– Edit the name by typing. The name is shown in the box with red borders and changes as you type. When finished, press **Enter**.

```

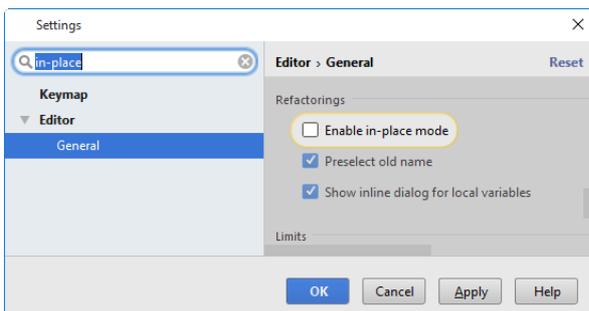
a = 3
b = 25
c = 46
myVar = math.sqrt(b ** 2 - 4 * a * c)
root1 = (-b + myVar) / (2*a)
root2 = (-b - myVar) / (2*a)
print(root1, root2)

```

Extracting variable with a dialog

To extract a variable using the dialog box

If the **Enable in place refactorings** check box is cleared in the Editor settings, the Extract Variable refactoring is performed by means of the dialog box.



1. In the editor, select the expression to be replaced with a variable. You can do that yourself or use the **smart expression selection** feature to let PyCharm help you. So, do one of the following:

– Highlight the expression. Then choose Refactor | Extract | Variable on the main menu or on the context menu.

Alternatively, press **Ctrl+Alt+V**.

– Place the cursor before or within the expression. Choose Refactor | Extract Variable on the main menu or on the context menu. or press

Ctrl+Alt+V.

In the Expressions pop-up menu, select the expression. To do that, click the required expression. Alternatively, use the Up and Down arrow keys to navigate to the expression of interest, and then press **Enter** to select it.

Note The Expressions pop-up menu contains all the expressions appropriate for the current cursor position in the editor.

When you navigate through the suggested expressions in the pop-up, the code highlighting in the editor changes accordingly.

2. In the **Extract Variable Dialog** dialog:

1. Specify the variable name next to Name. You can select one of the suggested names from the list or type the name in the Name box.

2. If more than one occurrence of the selected expression is found, you can select to replace all the found occurrences by selecting the corresponding check box. If you want to replace only the current occurrence, clear the Replace all occurrences check box.

3. Click **OK**.

Refer to the following sections for the other language- and framework-specific refactorings:

– For the JavaScript procedure, refer to the section [Extract Variable in JavaScript](#).

– For the Sass procedure, refer to the section [Extract Variable for Sass](#).

Inline

PyCharm provides the following inline refactorings:

- The [Inline Variable](#) refactoring replaces redundant variable usage with its initializer. This refactoring is opposite to [Extract Variable](#).

Example

BeforeAfter

```
import math

class Solver:
    def demo(self):
        a = 3
        b = 25
        c = 46
        return_type_of_sqrt = math.sqrt(b ** 2 - 4 * a * c)
        root1 = (-b + return_type_of_sqrt) / (2*a)
        root2 = (-b - return_type_of_sqrt) / (2*a)
        print(root1,root2)

Solver().demo()
```

```
import math

class Solver:
    def demo(self):
        a = 3
        b = 25
        c = 46
        root1 = (-b + math.sqrt(b**2 - 4*a*c)) / (2*a)
        root2 = (-b - math.sqrt(b**2 - 4*a*c)) / (2*a)
        print(root1,root2)

Solver().demo()
```

To perform the inline refactoring

1. Place the caret in the editor at the desired symbol to be inlined.
2. Do one of the following:
 - On the main menu or on the context menu, choose Refactor | Inline.
 - Press `Ctrl+Alt+N`.
3. In the Inline dialog box that corresponds to the selected symbol, confirm the inline refactoring.

Invert Boolean

This refactoring aims to flip the value of a Boolean variable and all its usages from True to False and vice versa.

Example

BeforeAfter

```
def foo():  
  abc<caret here> = True  
  return abc
```

```
def foo():  
  abc = False  
  return not abc
```

To invert a Boolean variable, follow these steps

1. Place the caret at the name of a Boolean variable.
2. Do one of the following:
 - On the main menu, choose Refactor | Invert Boolean.
 - On the context menu at caret location, choose Refactor | Invert Boolean.
3. In the dialog box that opens, click Preview to view the suggested changes in the Find tool window, or Refactor to invert the detected occurrences silently.

Move Refactorings

In this section:

- [Move refactorings: basics](#)
- [Performing Move refactoring](#)
- [Moving top-level symbols](#)
- [Moving function/method to the top-level](#)
 - [Example](#)

Move refactorings: basics

Tip The **Move** refactoring is also available from [UML Class diagram](#).

The Move refactorings allow you to move classes, functions, modules, files and directories within a project. So doing, PyCharm tracks these movements and automatically corrects all references to the moved symbols in the source code.

The following Move refactorings are available:

- The Move File refactoring moves a file to another directory.
- The Move Directory refactoring moves a directory to another directory.
- The Move Module Members refactoring moves top-level symbols of a Python module.
- The Make local function/method top-level refactoring converts a method or a local function to a top-level function and moves it to the specified file.

Performing Move refactoring

To perform a Move refactoring, follow these general steps:

1. Select the symbol to be moved and do one of the following:
 - On the main menu, or on the context menu, point to Refactor, and then choose Move.
 - Press **F6**.
 - In the [Project tool window](#), drag the symbol to the new location.

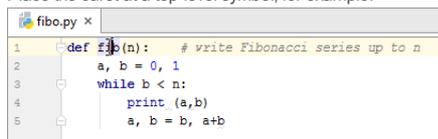
The dialog that opens depends on the type of the selected symbol.

2. Specify the move options according to the type of the item to be moved. See option descriptions in the [Move](#) dialog box reference.
3. [Preview and apply the changes](#).

Moving top-level symbols

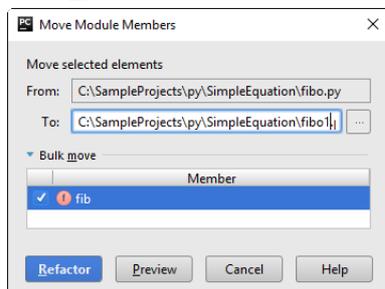
To move a member, follow these steps:

1. Place the caret at a top-level symbol, for example:



```
1 def fib(n): # write Fibonacci series up to n
2     a, b = 0, 1
3     while b < n:
4         print (a,b)
5         a, b = b, a+b
```

2. Press **F6**. The dialog box [Move Module Members](#) opens:



Refer to the [dialog reference](#) for the detailed description of controls.

3. In this dialog box, select the members to be moved, and specify the target file.
4. [Preview and apply the changes](#).

Moving function/method to the top-level

This refactoring moves local functions or methods to the top-level by converting references to instance attributes or variables from enclosing scopes to parameters and updating existing usages accordingly.

To move a function or a method to top-level, follow these steps:

1. Place the caret at the local function or method name.
2. On the main menu, or on the context menu of the editor, choose Refactor | Move, or press `F6`.
3. In the Make Method Top-Level dialog box that opens, specify the destination of move. You can type it manually, or click the browse button  and locate the destination file in the [Choose Destination File](#) dialog.
4. Click Refactor to perform the refactoring, or Preview, to shows the preview in the Find tool window. If satisfied with the preview results, confirm move by clicking Do Refactor.

Example

BeforeAfter

```
import math

class Solver(object):
    def __init__(self,a,b,c):
        self.a = a
        self.b = b
        self.c = c

    def demo(self, a, b, c):
        d = self.b ** 2 - 4 * self.a * self.c
        if d >= 0:
            disc = math.sqrt(d)
            root1 = (- self.b + disc) / (2 * self.a)
            root2 = (- self.b - disc) / (2 * self.a)
            print(root1, root2)
            return root1, root2
        else:
            raise Exception

Solver(2, 123, 0.025).demo()
```

```
import math

class Solver(object):
    def __init__(self, a, b, c):
        self.a = a
        self.b = b
        self.c = c

    def demo(b, a, c):
        d = b ** 2 - 4 * a * c
        if d >= 0:
            disc = math.sqrt(d)
            root1 = (- b + disc) / (2 * a)
            root2 = (- b - disc) / (2 * a)
            print(root1, root2)
            return root1, root2
        else:
            raise Exception

s = Solver(2, 123, 0.025)
demo(s.b, s.a, s.c)
```

Push Members Down

The Push Members Down refactoring helps clean up the class hierarchy by moving class members to a subclass. The members are then relocated into the direct subclasses only.

Example

BeforeAfter

```
class SuperClass:
    def super_method(self):
        pass

class SubClassOne(SuperClass):
    def my_method(self):
        pass

class SubClassTwo(SuperClass):
    def my_method(self):
        pass
```

```
class SuperClass:
    pass

class SubClassOne(SuperClass):
    def my_method(self):
        pass

    def super_method(self):
        pass

class SubClassTwo(SuperClass):
    def my_method(self):
        pass

    def super_method(self):
        pass
```

Pushing members down

1. In the editor, open the class whose members you need to push down.
2. On the main menu or on the context menu, choose Refactor | Push Members Down. [Push Members Down dialog box](#) displays the list of members to be pushed down.
3. In the Members to be pushed down area, select the members you want to move. Note that the member at caret is already selected.
If pushing a member might cause problems, you will be notified with red highlighting. It means that, if the situation is unattended, an error will emerge after refactoring. PyCharm prompts you with a Problems Detected dialog, where you can opt to ignore or fix the problem.
4. Preview and apply changes.

Pull Members Up

- [Basics](#)
- [Example](#)
- [Pulling members up](#)

Basics

The Pull Members Up refactoring allows you to move class members to a superclass.

Pull Members Up refactoring can create abstract methods. If a project makes use of the interpreter Python 2.x, then only the instance methods can be abstracted. If a project uses Python 3.x, then any method can be abstracted.

Note that PyCharm automatically adds import statements, required for abstract methods.

Example

Before/After

<pre>class SuperClass: def super_method(self): pass class SubClassOne(SuperClass): def my_method(self): pass</pre>	<pre>class SuperClass: def super_method(self): pass def my_method(self): pass class SubClassOne(SuperClass): pass</pre>
<pre>class Bar(object): pass class SomeClass (Bar): def foo(self): pass</pre>	<pre>from abc import abstractmethod from abc import ABCMeta class Bar(object, metaclass=ABCMeta): @abstractmethod def foo(self): pass class SomeClass (Bar): def foo(self): pass</pre>

Pulling members up

1. Select the class to be moved to a superclass.
2. On the main menu or on the context menu, choose Refactor | Pull Members Up. The [Pull Members Up](#) dialog box appears.
3. Select the destination object (superclass).
4. In the Members section, select the members you want to move.
5. To move a method as abstract, select the check box in the column Make abstract next to the method in question.
6. Click Refactor to pull the selected members to their destination.

Rename Refactorings

On this page:

- [Basics](#)
- [Examples](#)
 - [Renaming a method](#)
 - [Renaming a template](#)
- [Renaming a symbol](#)
- [Renaming a file or directory](#)
- [Important notes](#)

Basics

Rename refactorings allow you to rename symbols with all the references to them in the code corrected automatically.

The following rename refactorings are available in PyCharm:

- **Rename Class.** The following usages are renamed:
 - Import statements
 - Qualified names of classes
- **Rename Method.** The following usages are renamed:
 - All calls of the method.
 - All overridden/implemented methods in subclasses.
- **Rename Field.**
- **Rename Function.**
- **Rename Variable.**
- **Rename Parameter.** The following usages are renamed:
 - All usages of the parameter.
 - The corresponding `param` tag in documentation comment.
- **Rename CSS color value.**
- **Rename File.**
- **Rename Directory.**

Examples

Renaming a method

This feature is supported only when the Python plugin is installed.

Before/After

Renaming method

```
def was_published_today(self):  
    return self.pub_date.date () == datetime.date.today()
```

```
def published_today(self):  
    return self.pub_date.date () == datetime.date.today()
```

Renaming a template

This feature is supported only when the Python plugin is installed.

Rename a template:

```
urlpatterns = patterns('MyDjangoApp.polls.views',  
    (r'^polls/$', 'index'),  
    (r'^polls/(?P<poll_id>\d+)/$', 'detail'),  
    (r'^polls/(?P<poll_id>\d+)/results/$', 'results'),  
    (r'^polls/(?P<poll_id>\d+)/vote/$', 'vote'),
```

So doing, the following usages will be renamed:

The screenshot shows the 'Refactoring preview' window with the following content:

- Function to be renamed to **poll_details**
 - detail()
- References in code to function detail (2 references in 2 files) (2 usages)
 - Unclassified usage (2 usages)
 - MyDjangoApp (2 usages)
 - index.html (1 usage)
 - (5:37)
{{ poll.id }}. {{ poll.question }}
 - urls.py (1 usage)
 - (9:37) (r'^polls/(?P<poll_id>\d+)/\$', 'detail').

Renaming a symbol

To rename a symbol, follow these general steps

1. Select the item to be renamed.

- To select a file, click the desired file in the [Project tool window](#).
- To select a symbol in the editor, place the caret at the name of the symbol to be renamed.
- To select a symbol in the [Project tool window](#), make sure that the members are shown, and then click the desired symbol.
- To select a symbol in the [Structure view](#), click the desired symbol in the Structure tool window.

2. Choose Refactor | Rename on the main menu or on the context menu, or press `Shift+F6`.

3. The subsequent behavior depends on the check box [Enable in-place mode](#) (Settings/Preferences dialog, [Editor](#)).

- If this check box is selected, the suggested name appears right below the symbol in question. You can either accept suggestion, or type a new name. However, if you press `Shift+F6` once more, PyCharm will display the [Rename](#) dialog box with more options.
- If this check box is not selected, the [Rename](#) dialog box opens immediately.

The set of controls and their names depend on the type of the symbol to be renamed.

4. [Preview and apply changes](#).

Renaming a file or directory

To rename a file or directory

1. Select a desired file in the [Project tool window](#).

2. Choose Refactor | Rename on the main or context menu or press `Shift+F6`.

3. In Rename File dialog box that opens, specify the new file name. Select Search in comments and strings checkbox to let PyCharm apply changes to comments and strings.

4. Press Preview to observe possible changes in [Find Tool Window](#). Press Refactor to proceed. PyCharm finds all the occurrences of the file name and changes them respectively.

Important notes

- Local variables are renamed in-place.

```
for poll_id in args:
    try:
        poll = Poll.objects.get(pk=int(poll_id))
    except Poll.DoesNotExist:
        raise CommandError('Poll "%s" does not exist' % poll_id)
```

To be able to use the Rename dialog when renaming local variables, you should [disable in-place refactoring in the editor settings](#).

- When renaming Gherkin steps, mind the following limitations:

- Step definitions should not contain regular expressions
- Step names should contain alphanumeric characters only.
- A step definition should be only one in various frameworks.
- There should be a "one-to-one" mapping between a step and a step definition.

Safe Delete

On this page:

- [Introduction](#)
- [Performing the refactoring](#)

Introduction

The Safe Delete refactoring lets you safely remove files from the source code.

To make sure that deletion is safe, PyCharm looks for the usages of the file being deleted. If such usages are found, you can explore them and make the necessary corrections in your code before the symbol is actually deleted.

Performing the refactoring

1. In the [Project](#) tool window, select a file to be deleted.
2. Do one of the following:
 - Press `Alt+Delete`.
 - Select Refactor | Safe Delete from the main or the context menu.
 - Select Refactor | Refactor This from the main menu (`Ctrl+Shift+Alt+T`), and select Safe Delete.
3. In the [Safe Delete dialog](#), select the necessary options and click OK.
4. If the refactoring is potentially unsafe, the Usages Detected dialog opens.
 - View Usages. Click this button to see where in your code the item you are about to delete is used. As a result, the [Find tool window](#) opens. Analyze your code and make the necessary corrections. Then click Do Refactor. (If you want to rerun the refactoring from its start, click Rerun Safe Delete. PyCharm will check if the refactoring is safe once more.)
 - Delete Anyway. Click this button to delete the item without looking at its usages.

Working with Run/Debug Configurations

In this section:

- [Run/Debug Configuration](#)
- [Creating and Editing Run/Debug Configurations](#)
- [Changing Run/Debug Configuration Defaults](#)
- [Creating and Saving Temporary Run/Debug Configurations](#)
- [Creating Folders and Grouping Run/Debug Configurations](#)

Run/Debug Configuration

On this page:

- [Basics](#)
- [Temporary configuration](#)
- [Permanent configuration](#)
- [Default run/debug configuration settings](#)

Basics

PyCharm enables using numerous run/debug configurations. Each run/debug configuration represents a named set of run/debug startup properties. When you perform run, debug, or test operations with PyCharm, you always start a process based on one of the existing configurations using its parameters.

PyCharm comes with a number of run/debug configuration types for the various running, debugging and testing issues. The user can create his/her own run/debug configurations of specific types.

Each run/debug configuration type has its own default settings. Whenever a new run/debug configuration of the respective type is created, it is based on these default settings.

Temporary configuration

A temporary run/debug configuration is automatically created every time you choose Run <item_name> or Debug <item_name> for an item without a permanent configuration. Temporary configurations can be saved as permanent.

Temporary configurations are marked with semi-transparent icons and are managed same way as the permanent configurations.

By default, 5 temporary configurations are allowed per project. You can change this limit via the [Edit Configurations](#) dialog.

Permanent configuration

A configuration of this type [is explicitly created](#) for a particular class or method. If there is no permanent configuration for an item, PyCharm automatically creates a temporary configuration for it, when you choose Run <item_name> or Debug <item_name> on the context menu of this class or method.

Default run/debug configuration settings

The default run/debug configuration settings appear in the Run/Debug configurations dialog box under the node Defaults. They denote the settings that are used when a new run/debug configurations is created.

You can set the default settings for a specific configuration type that will become applicable to any run/debug configuration of this type created later. Changing defaults does not affect the existing run/debug configurations.

The process of editing the per-type default configuration settings is described [here](#). The process of creating or editing the custom run-debug configurations is described on the page [Creating or Editing Run/Debug configurations](#).

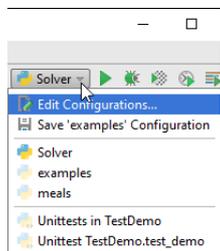
Creating and Editing Run/Debug Configurations

On this page:

- [Overview](#)
- [Creating a run/debug configuration](#)
- [Editing an existing run/debug configuration](#)
- [Important note for multi-project usage](#)

Overview

With the Navigation bar visible (View | Navigation Bar), the available [run/debug configurations](#) are displayed in the run/debug configuration selector in the Run area:



PyCharm provides the [Run/Debug Configuration](#) dialog box as a tool for handling run/debug configurations: create configuration profiles or change the default ones.

PyCharm suggests the following ways to create a run/debug configuration:

- [Create a run configuration manually](#) on the base of the default one, using the [Run/Debug Configuration](#) dialog box.
- [Save a temporary run configuration](#).

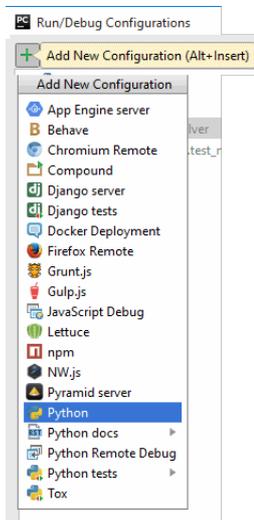
Also, the Run/Debug configuration can be deleted automatically for the deleted/obsolete targets, if this capability is enabled. See details [here](#).

Note, that this capability is applicable only to those configurations that had being created automatically by CLion.

Creating a run/debug configuration

To create a run/debug configuration, follow these steps:

1. In the [Run/Debug Configuration](#) dialog box, click **+** on the toolbar or press `Alt+Insert`. The drop-down list shows the default run/debug configurations. Select the desired configuration type.



The fields that appear in the right pane, display the default settings for the selected configuration type.

Note If you want to change the settings of the default run/debug configuration, expand the Defaults node, select the desired configuration type, and modify it as required.

2. For the new run/debug configuration:
 - Specify its name in the Name text box. This name will be shown in the list of the available run/debug configurations.
 - Specify whether you want to make PyCharm check execution status of the instances of the same run/debug configuration. If you want to make sure that only one instance of the run/debug configuration is currently executed, select the check box **Single instance only**. In this case, a confirmation dialog box will show up every time you try to launch run/debug configuration, when one instance of the same type is still running. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. If this check box is not selected, you can launch as many instances of the runner as required. As the result, each runner will start in its own tab of the Run tool window.

Tip To use an existing configuration as a pattern, create its copy by clicking the Copy button  on the toolbar, then change it as required.

3. In the Before launch section, define whether you want to execute some tools or scripts prior to launching the run/debug configuration.

4. Specify additional parameters depending on the configuration type. Refer to the descriptions of run/debug configuration parameters below the [Run/Debug Configurations](#) section.
5. Apply the changes and close the dialog box.

Editing an existing run/debug configuration

To change an existing run/debug configuration:

- On the main menu, choose Run | Edit Configurations.
- With the Navigation Bar visible (View | Navigation Bar), choose Edit Configurations from the run/debug configurations selector.
- Press `Shift+Alt+F10`, then press `0` to display the Edit Configuration dialog box or select the configuration from the pop-up window and press `F4`.

In the corresponding run/debug configuration dialog box, change parameters as required.

Important note for multi-project usage

If a project has been created in an earlier version of PyCharm, its run/debug configurations can be lost, when such a project is [added to another project](#), already opened in the same window.

To avoid the loss of run/debug configurations, it is recommended to open such a project once in the latest version, and only after being added to another project.

Changing Run/Debug Configuration Defaults

On this page:

- [Introduction](#)
- [Configuring defaults](#)

Introduction

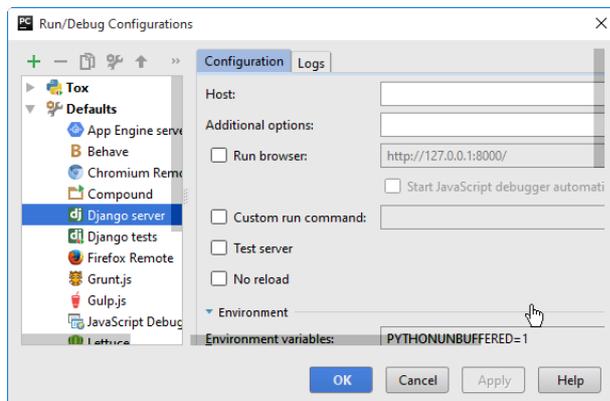
For a run/debug configuration of a particular type, you can set up the default values for one or more parameters and save them as template for further usages. In this case, next time when you create a new configuration of that type, the corresponding fields of the dialog will already contain the specified values.

Note that changing the default values does not affect the already existing run/debug configurations.

Configuring defaults

To set up the default values for a run/debug configuration, follow these steps:

1. In the left-hand pane of the run/debug configuration dialog, expand the Defaults node.
2. From the drop down list under Defaults node, select the desired configuration type. The corresponding configuration template appears in the right-hand pane.
3. Specify the desired parameters of the right pane and click Apply to save the template.



Tip The fields Name, Share and Single instance only are not available for the default run/debug configurations.

Creating and Saving Temporary Run/Debug Configurations

On this page:

- [Introduction](#)
- [Creating a temporary run/debug configuration](#)
- [Saving a temporary run/debug configuration](#)

Introduction

Sometimes you might need to run or debug a script, an application, or a test, without creating a dedicated run configuration. In this case, PyCharm provides a [temporary run configuration](#).

Temporary run/debug configuration is added to the list of available configurations and works same way as the [permanent run/debug configurations](#). You can change its settings using the [Run/Debug Configuration](#) dialog box and optionally save it as permanent.

Creating a temporary run/debug configuration

To create a temporary run/debug configuration

1. Select the desired script in the Project tool window or open it in the editor.
2. Do one of the following:
 - On the context menu, choose Run <name> or Debug <name>.
 - Press `Ctrl+Shift+F10`.

PyCharm creates a temporary configuration, which appears in the Run/Debug Configuration selector, when the run or debug session is over.

Saving a temporary run/debug configuration

To save a temporary configuration, do one of the following

- In the Run/Debug Configuration selector, choose Save <configuration name>.
- In the Run/Debug Configuration dialog box, click .
- On the context menu of the editor or Project view, choose Save <configuration name>.

Creating Folders and Grouping Run/Debug Configurations

On this page:

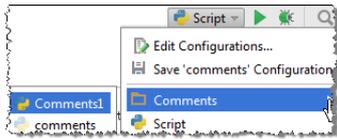
- [Introduction](#)
- [Creating folders of run/debug configurations](#)
- [Deleting folders](#)
- [Changing order of folders](#)

Introduction

When there are too many run/debug configurations of the same type, you can put them into directories, and thus make them visually distinguishable.

Folders area used to organize run/debug configurations. When not needed, they can be deleted, and the run/debug configurations under those folders just pass under the root of the corresponding type.

Once grouped, the run/debug configurations appear in the drop-down list under the corresponding folders:



Creating folders of run/debug configurations

To create a folder of run/debug configurations

1. Open [Run/Debug Configuration](#) dialog box.
2. In the [Run/Debug Configuration dialog box](#), click  on the toolbar. A new empty folder is created.
3. Specify the folder name in the text field to the right, or accept default.
4. Select the desired run/debug configuration of a certain type, and move under the target folder. This can be done in one of the following ways:
 - Use drag-and-drop.
 - Use the toolbar buttons .
 - Press `Alt+Up` or `Alt+Down`.
5. Apply changes. Note that the folders of run/debug configurations should not be empty. The empty folders are not saved.

Deleting folders

To delete a folder

1. In the [Run/Debug Configuration dialog box](#), select a folder to be deleted.
2. On the toolbar, click . The selected folder is deleted silently. Any run/debug configurations grouped under this folder, are moved under the root of the corresponding type.
3. Apply changes.

Changing order of folders

To move folders up or down

1. In the [Run/Debug Configuration dialog box](#), select one of the folders within a category of run/debug configurations.
2. Do one of the following:
 - On the toolbar, click  or .
 - Press `Alt+Up` or `Alt+Down`.

The folder in question moves one position up or down.
3. Apply changes.

After closing the dialog box, groups of run/debug configurations in the run/debug configurations selector on the main toolbar appear in the order achieved by moving folders up or down within a category.

Running

This section describes the procedures that are common for the various types of applications:

- [Running Applications](#)
- [Rerunning Applications](#)
- [Reviewing Results](#)
- [Stopping and Pausing Applications](#)
- [Setting Log Options](#)
- [Viewing Running Processes](#)

For the details related to running applications in the supported frameworks, refer to [Language and Framework Specific Guidelines](#)

Running Applications

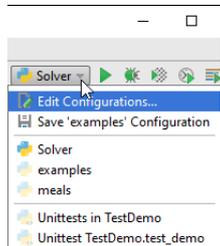
On this page:

- [Introduction](#)
- [Running a script](#)
- [Using the Run pop-up menu](#)

Introduction

PyCharm enables running entire applications as well as particular scripts.

PyCharm makes use of the settings defined in a [Run/Debug Configuration](#). All the run configurations that exist in a project, are available in the Select Run/Debug Configuration drop-down list.



If you want to see a list of all currently running applications, select Run | Show Running List from the main menu. Refer to the section [Viewing Running Processes](#) for details.

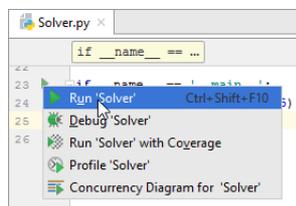
Note that after you've started a run session, the  icon that marks the [Run tool window](#) and in the Run/Debug Configuration Selector toggles to  to indicate that the run process is active.

Note If the options that launch tools before running were enabled in a [Run/Debug configuration](#), PyCharm runs the tools, and after success will run the application. Otherwise, the program will start immediately.

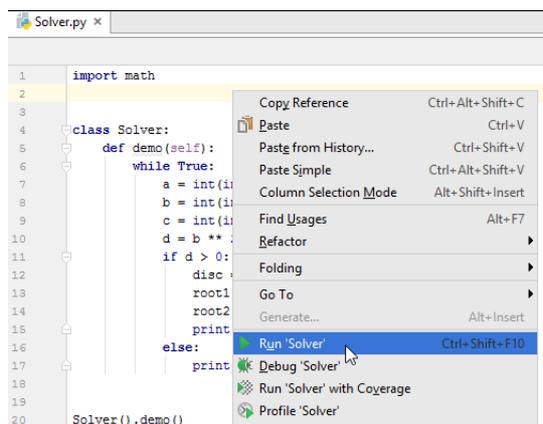
Running a script

To run a script, do one of the following:

- Choose Run | Run on the main menu or press `Shift+Alt+F10`, and then select the desired run configuration.
- For the main clause: In the left gutter, click the icon , and choose the desired command.

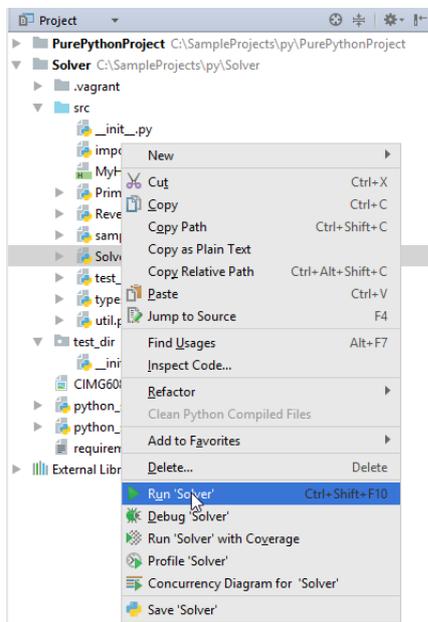


- Choose Run <name> on the context menu:



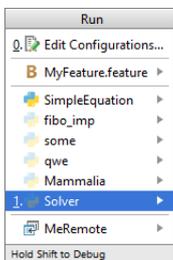
- Press `Ctrl+Shift+F10`.

- Select the desired module in the Project tool window and choose Run <name > on the context menu of the selection:



Using the Run pop-up menu

Invoke the Run pop-up menu either by choosing the Run | Run command on the main menu, or by pressing `Shift+Alt+F10`.



From this pop-up menu that you can:

- Invoke the Edit Configuration dialog.
- Edit the selected configuration before launch (`F4`).
- Instantly delete a configuration (`Delete`).
- Switch from run to debug and vice versa (hold `Shift`).
- Access a previously selected configuration (`1`).

This pop-up menu can also be quickly accessed by pressing `F9`, when you're not running any debug session.

Rerunning Applications

You can re-run an application if its tab is still opened in the [Run](#) window. The program re-runs with the initial settings.

To re-run an application

1. In the Run window, select the tab where the desired application is opened.
2. In the toolbar of the Run window, click the Rerun button , or press `Ctrl+F5`.

Tip If you want to re-run without losing focus in the editor tab you are working in, press `Shift+F10`.

Reviewing Results

You can review any output from your running applications in the Run window console. The output from each application is displayed in its own tab of the [Run](#) tool window, named after the corresponding [run/debug configuration](#).

If you re-run an application, the new output overwrites the contents of the tab. To preserve the output of an application, even if you re-run it, [pin](#) the output tab.

Stopping and Pausing Applications

In the [Run](#) tool window, you can stop a program, or pause its output. If a program is stopped, its process is interrupted and exits immediately. When program output is paused, the program continues running in the background, but its output is suspended.

To stop a program, do one of the following

- In the Run tool window, click the Stop button  on the toolbar, or press `Ctrl+F2`.
- To close the active tab, click the Close button , or press `Ctrl+Shift+F4`.

To suspend program output

- In the Run tool window, click the Pause button  on the toolbar.

Note that the button is not available for [Run/Debug Configuration: Node JS](#), [Run/Debug Configuration: Node JS Remote Debug](#), and [Run/Debug Configuration: NodeUnit](#).

Setting Log Options

Use Logs tab in the Run/Debug Configuration dialogs to configure the way log files, generated by an application or server, are displayed in the console.

If your application or server generates log files, the default entries will be automatically added to the log file list in the Run/Debug Configuration dialog.

To configure Logs options

1. In the [Run/Debug Configuration](#) dialog box, click Logs tab. The table Log files to be shown in console displays the list of log files (if any).
2. Click [+](#). [Edit Log Files Aliases Dialog](#) dialog is displayed.
3. In the Alias field, type the alias name to show in the list of log entries. In the Log File Location field, type the fully qualified name of the log file, or specify its location by pressing the ellipsis button. Select whether you want to show all or last file coverable by pattern. Click OK to close the dialog.
4. Activate the log entry. To do that, select the check box in the Is Active column.
5. To skip the previous content, select the check box in the Skip Content column.

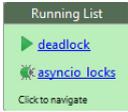
Viewing Running Processes

PyCharm makes it possible to view all the running applications. The command Show Running List of the menu Run is only enabled if there are active applications. If no applications are active, the command is greyed out.

To view the list of running applications

- On the main menu, choose Run | Show Running List.

A popup listing all active applications is displayed in the top-right corner of the editor.



Tip Same is also valid for debugging.

Debugging

- Debugging
 - [Overview](#)
 - [General debugging steps](#)
- [Breakpoints](#)
- [Configuring Debugger Options](#)
- [Starting the Debugger Session](#)
- [Pausing and Resuming the Debugger Session](#)
- [Monitoring the Debug Information](#)
- [Examining Suspended Program](#)
- [Exploring Frames](#)
- [Finding the Current Execution Point](#)
- [Stepping Through the Program](#)
- [Remote Debugging](#)
- [Inline Debugging](#)
- [Using Debug Console](#)
- [Attaching to Local Process](#)
- [Thread Concurrency Visualization](#)
- [Viewing as Array or DataFrame](#)

Overview

This section describes the procedures that are common for various types of applications.

For details on debugging applications in the supported frameworks, refer to [Language and Framework Specific Guidelines](#).

PyCharm provides a full range of facilities for debugging your source code:

- [Breakpoints in Python](#).
- [Breakpoints in JavaScript](#).
- Customizable breakpoint properties: conditions, pass count, etc.
- [Frames](#), [variables](#), and [watches](#) views in the debugger UI.
- Runtime [evaluation of expressions](#).

If you want to see a list of all currently debugging applications, select Run | Show Running List from the main menu. Refer to the section [Viewing Running Processes](#) for details.

General debugging steps

1. [Configure the debugger options](#).
2. To debug [CoffeeScript](#), [TypeScript](#), and [Dart](#) code, you need **source maps** generated in addition to the JavaScript code. [Source maps](#) set the correspondence between lines in your original code and in the generated JavaScript code, otherwise your breakpoints will not be recognised and processed correctly.

JavaScript and source maps are generated by compiling the original code either manually using the **File Watcher** of the type [CoffeeScript](#), or by the built-in compiler (for [TypeScript](#)), or through integration with the **Pub Serve** tool (for [Dart](#)). After that, you can debug the output JavaScript code. See [Compiling CoffeeScript to JavaScript](#), [Compiling TypeScript to JavaScript](#), and [Using Integration with the Pub Tool](#) for details.
3. [Define a run/debug configuration](#) for the application to be debugged.
4. [Create breakpoints](#) in the source code.
5. [Launch](#) a debugging session.
6. [Pause or resume](#) the debugging session as required.
7. During the debugger session, [step through the breakpoints](#), [evaluate expressions](#), change values on-the-fly, [examine suspended program](#), [explore frames](#), and [set watches](#).

After you've started a debug session, the  icon that marks the [Debug tool window](#) toggles to  to indicate that the debug process is active.

Breakpoints

Breakpoints are source code markers used to trigger actions during a debugging session.

In this part:

- [Types of Breakpoints](#)
- [Breakpoints Icons and Statuses](#)
- [Using Breakpoints](#)

Types of Breakpoints

On this page:

- [Introduction](#)
- [Python Line breakpoint](#)
- [Temporary Line breakpoint](#)
- [Django Line breakpoints](#)
- [Exception breakpoint](#)
- [JavaScript breakpoints](#)

Introduction

PyCharm lets you create breakpoints of several types. Each breakpoint type supported by PyCharm addresses different debugging needs and has its own individual settings.

Breakpoints are triggered when the program reaches the specified line of source code, before it is executed. The line of code that contains a set breakpoint, is marked with a red stripe; once such line of code is reached, the marking stripe changes to blue.

```
print("%s: %s" % (thread_name, time.ctime(time.time())))  
  
# Create two threads as follows  
t1 = threading.Thread(target=print_time, args=(1,))  
t2 = threading.Thread(target=print_time, args=(2,))
```

Once set, a breakpoint remains in project until removed. Breakpoints can only be set on executable lines of code. Comments, declarations of methods, and empty lines are not valid locations for breakpoints.

Tip If a file with breakpoints has been modified externally, for example, updated from a version control repository, or changed in an external editor, so that line numbers are changed, then the breakpoints will be moved accordingly. Note that PyCharm should be running at the moment of such modification; otherwise, such changes will pass unnoticed.

Python Line breakpoint

These breakpoints are assigned to lines of source code and are used to target a particular section for debugging.

Temporary Line breakpoint

These breakpoints are assigned to lines of source code and are used to target a particular section for debugging. When hit, such breakpoints are immediately removed.

Django Line breakpoints

These breakpoints are assigned to the lines of the Django templates, and can only be set on the lines that contain Django tags. Plain HTML lines in Django templates are not valid location for the Django line breakpoints.

```
{% if poll_list %}  
    <ul>  
        {% for poll in poll_list %}  
        <li><a href="{% url polls.  
        {% endfor %}  
    </ul>  
    {% else %}  
    <p>No polls available</p>  
{% endif %}
```

Django line breakpoints are triggered when the corresponding tags are rendered by the Django template engine. Refer to the section [Debugging Django Templates](#) for details.

Exception breakpoint

PyCharm provides exception breakpoints for Python, Django, and JavaScript.

Exception breakpoints are triggered when the specified exception is thrown. Unlike the line breakpoints, which require specific source references, exception breakpoints apply globally to the exception condition, rather than to a particular code reference.

Depending on the type of processing an exception, the debugging can break when a process terminates with an exception, or as soon as an exception occurs.

Django exception breakpoints enable suspending the program execution when a Django template error occurs (exceptions `TemplateDoesNotExist` or `VariableDoesNotExist`), if such a breakpoint is enabled in the Django exception breakpoints tab of the [Breakpoints](#) dialog.

JavaScript breakpoints

JavaScript breakpoints are the line breakpoints assigned to particular lines of JavaScript source code. They can be set in `*.html` and in `*.js` files, and are used to target a particular section of code for [debugging](#).

Breakpoints Icons and Statuses

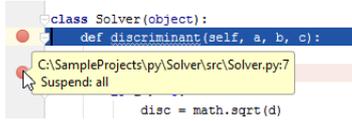
On this page:

- [Basics](#)
- [Breakpoint states and icons](#)

Basics

When a breakpoint is set, the editor displays a breakpoint icon in the gutter area to the left of the affected source code. A breakpoint icon denotes status of a breakpoint, and provides useful information about its type, location, and action.

The icons serve as convenient shortcuts for managing breakpoints. Clicking an icon removes the breakpoint. Successive use of **Alt** - click on an icon toggles its state between enabled and disabled. The settings of a breakpoint are shown in a tooltip when a mouse pointer hovers over a breakpoint icon in the gutter area of the editor.



Breakpoint states and icons

The table below summarizes the possible breakpoint states:

Status	Line	Temporary Line	Exception	Description
	Python / Django/ JavaScript		Python / Django / JavaScript	
Enabled				 Shown at design-time or during the debugging session.
Disabled				 Indicates that nothing happens when the breakpoint is hit.
Conditionally disabled				 This state is assigned to breakpoints when they depend on another breakpoint to be activated.

When the button  is pressed in the toolbar of the [Debug](#) tool window, all the breakpoints in a project are muted, and their icons become grey: .

Using Breakpoints

In this section:

- [Accessing Breakpoint Properties](#)
- [Creating Line Breakpoints](#)
- [Creating Exception Breakpoints](#)
- [Configuring Breakpoints](#)
- [Enabling, Disabling and Removing Breakpoints](#)
- [Moving Breakpoints](#)
- [Named Breakpoints](#)
- [Navigating Back to Source](#)
- [Working with Groups of Breakpoints](#)

Accessing Breakpoint Properties

On this page:

- [Introduction](#)
- [Viewing all breakpoints](#)
- [Viewing properties of a breakpoint](#)

Introduction

To view the whole list of the breakpoints in the current project, use the [Breakpoints](#) dialog box. For each individual breakpoint in the list, you can view and change its properties as required.

Viewing all breakpoints

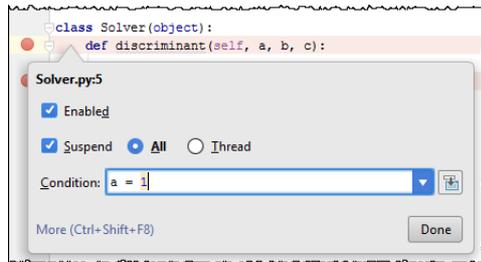
To view the list of all breakpoints and their properties, do one of the following:

- On the main menu, choose Run | View Breakpoints.
- Press `Ctrl+Shift+F8`.
- In the toolbar of the [Debug tool window](#), click .
- Breakpoints are visible in the [Favorites](#) tool window.

Viewing properties of a breakpoint

To view properties of a single breakpoint

- Right-click a breakpoint icon in the left gutter of the editor.



Creating Line Breakpoints

On this page:

- [Basics](#)
- [Creating line breakpoints in the editor](#)
- [Creating temporary line breakpoints](#)
- [Deleting line breakpoints](#)

Basics

A **line breakpoint**  is a breakpoint assigned to a specific line in the source code.

Line breakpoints can be set on executable lines. Comments, declarations and empty lines are not valid locations for the line breakpoints.

PyCharm also allows placing line breakpoints on the lines of Django templates.

Creating line breakpoints in the editor

1. Place the caret on the desired line of the source code.
2. Do one of the following:
 - Click the left gutter area at a line where you want to toggle a breakpoint.
 - On the main menu, choose Run | Toggle Line Breakpoint.
 - Press `Ctrl+F8`.

Creating temporary line breakpoints

1. Place the caret on the desired line of the source code.
2. Do one of the following:
 - On the main menu, choose Run | Toggle Temporary Line Breakpoint.
 - Press `Ctrl+Shift+Alt+F8`.

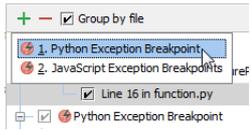
Deleting line breakpoints

Do one of the following:

- In the [Breakpoints](#) dialog box, select the desired line breakpoint, and click .
- In the editor, locate the line with the line breakpoint to be deleted, and click its icon in the left gutter.
- Place caret on the desired line and press `Ctrl+F8`.

Creating Exception Breakpoints

1. On the main menu, choose Run | View Breakpoints, or press `Ctrl+Shift+F8`.
2. In the **Breakpoints** dialog box that opens, click **+**.
3. Select Python Exception Breakpoint or JavaScript Exception Breakpoint from the drop-down list.



4. In the Choose Exception Class dialog box, specify the desired exception class from the library, or from the project, and click OK.
PyCharm returns you to the Breakpoints dialog box.
5. Configure the new exception breakpoint as described in [Configuring Breakpoints](#).

- [Basics](#)
- [Configuring breakpoints](#)

Basics

For a breakpoint, you can configure the following properties:

- Actions to be performed upon hitting a certain breakpoint.
- Suspend policy, which defines whether the application should be suspended upon hitting the breakpoint.
- Dependencies on other breakpoints.
- Conditions defining when a breakpoint is hit.

PyCharm suggests the following way to change the breakpoints properties:

- Using the [Breakpoints](#) dialog box, for a breakpoint selected in the list.
- Using breakpoint icon in the left gutter

Configuring breakpoints

To configure actions, suspend policy and dependencies of a breakpoint

1. Do one of the following:

- Right-click a breakpoint in the left gutter, and then click the link More or press `Ctrl+Shift+F8`.
- Open the Breakpoints dialog box as described on page [Accessing Breakpoint Properties](#) and select the desired breakpoint in the list.
- In the [Favorites](#) tool window, select the desired breakpoint, and click .

Note that the pop-up window shows less options than the [Breakpoints](#) dialog box. To show hidden options, click More.

2. Define the actions to be performed by PyCharm on hitting breakpoint:

- To notify about the reaching of a breakpoint with a text message in the debugging console, select the Log message to console check box.
To evaluate an expression in the context of a breakpoint and display its value in the debugging console, check the option Evaluate and log, and enter a valid expression in the option field.

This feature lets you obtain information about your running application without having to suspend its execution.

- To set a breakpoint the current one depends on, select it from the Disabled until selected breakpoint hit drop-down list. Once dependency has been set, the current breakpoint is disabled until selected one is hit.
 - Choose Disable again radio button to disable the current breakpoint after selected breakpoint was hit.
 - Choose Leave enable radio button to keep the current breakpoint enabled after selected breakpoint was hit.
- Enable suspending an application upon reaching a breakpoint by selecting the Suspend check box, and then select one of the option buttons to specify the way a running program will be paused. For more information on the Suspend options, refer to [Breakpoints](#) dialog reference.
- To set the break condition, enable condition by selecting the appropriate check box, and enter the desired expression in the Condition field. If the expression evaluates to true, the user-selected actions are performed. If the evaluation result is false, the breakpoint does not produce any effect.

Enabling, Disabling and Removing Breakpoints

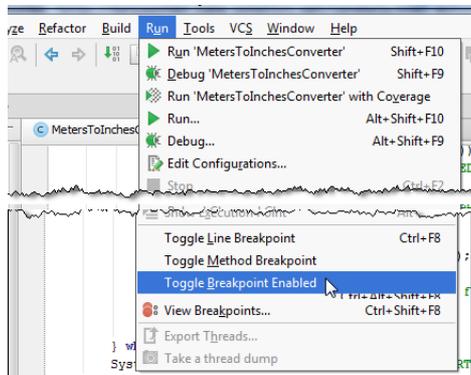
On this page:

- [Toggling between the enabled and disabled state of a breakpoint](#)
- [Disabling a breakpoint temporarily in the editor](#)

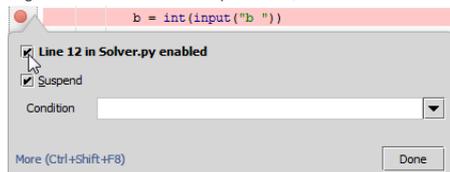
Toggling between the enabled and disabled state of a breakpoint

When you temporarily disable or enable a breakpoint, its icon changes from  to  and vice versa.

1. Place the caret at the desired line with a breakpoint.
2. Do one of the following:
 - On the main menu, choose Run | Toggle Breakpoint Enabled:



- Right-click the desired breakpoint icon, select or deselect the <breakpoint name> enabled check box, and then click Done.



-  -click on the breakpoint icon.

Tip Alternatively, [open the Breakpoints dialog box](#), select the desired breakpoint, and select or clear the check box to its left.

Disabling a breakpoint temporarily in the editor

When you temporarily disable a breakpoint, its icon changes from  to .

Alternatively, open the Breakpoints dialog box, as described on page [Accessing Breakpoint Properties](#), select the desired breakpoint, and clear the check box next to it or the Line <line number> in <file name> check box in the right-hand pane.

Enabling a temporarily disabled breakpoint in the editor

When you enable a temporarily disabled breakpoint, its icon changes from  to .

1. Place the caret at the desired line with a breakpoint.
2. Do one of the following:
 - Right-click the desired breakpoint icon, select the Line <line number> in <file name> check box in the pop-up dialog box that opens, and then click Done.
 - With the  key pressed, click the breakpoint icon.

Alternatively, open the Breakpoints dialog box, as described on page [Accessing Breakpoint Properties](#), select the desired breakpoint, and select the check box next to it or the Line <line number> in <file name> check box in the right-hand pane.

Removing a breakpoint

To remove a breakpoint permanently, do one of the following:

- [Open the Breakpoints dialog box](#), select the desired breakpoint, and click .
- Click the breakpoint icon in the left gutter of the editor.

Removing all breakpoints of a certain type

1. On the main menu, choose Run | View Breakpoints, or press .
2. In the [Breakpoints](#) dialog, press the left arrow key to select the desired category.
3. Press .

All breakpoints of a certain type will be deleted.

Moving Breakpoints

To move a breakpoint, drag-and-drop it to the target line.

Named Breakpoints

On this page:

- [Introduction](#)
- [Editing breakpoint description](#)
- [Searching for a breakpoint using its name](#)

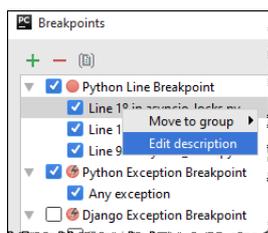
Introduction

PyCharm makes it possible to add a name or a short description to a breakpoint to facilitate search.

Editing breakpoint description

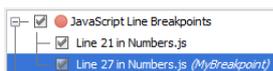
To edit a breakpoint description

1. [Open the Breakpoints dialog](#).
2. Right-click a breakpoint you are interested in.
3. On the context menu, choose `Edit description`.



4. In the Edit Description dialog box, type the desired description.

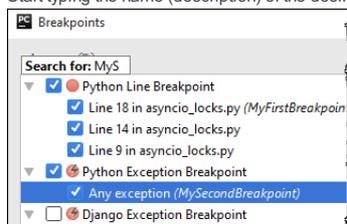
The specified description shows in italic next to the address of a breakpoint in the [Breakpoints dialog](#):



Searching for a breakpoint using its name

To search a breakpoint by name

1. [Open the Breakpoints dialog](#).
2. Start typing the name (description) of the desired breakpoint.



The breakpoint with the matching description gets the focus.

To navigate to the breakpoints source from the Breakpoints dialog box

- To view the selected breakpoint without closing the dialog box, use the preview pane.
- To open the file with the selected breakpoint for editing, double-click the desired breakpoint. To close Breakpoints dialog, press **F4**. The caret will be placed at the line marked with the breakpoint in question.

On this page:

- [Introduction](#)
- [Creating groups of breakpoints](#)
- [Moving breakpoints to another group, or out of a group](#)
- [Toggling a group of breakpoints](#)

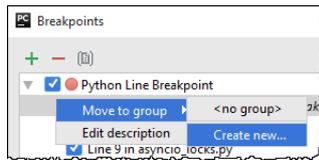
Introduction

PyCharm makes it possible to organize breakpoints in groups, for example, to mark out breakpoints for a specific problem. This is done in the [Breakpoints dialog](#).

Creating groups of breakpoints

To create a group of breakpoints

1. In the [Breakpoints dialog](#), right-click one or more breakpoints you are interested in.
2. On the context menu, point to the command Move to group, and then on the submenu, choose Create new...:

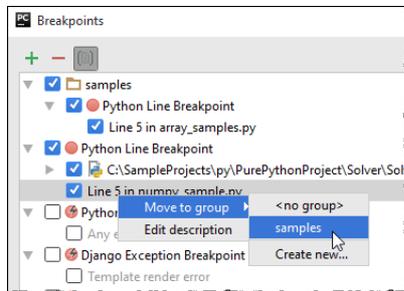


3. In the New Group dialog box, type the name of the new group. The selected breakpoint moves to the newly created group.
4. Optionally, you can right-click a group of breakpoints and select Set as default from the popup menu. All newly created breakpoints will be automatically added to this group.

Moving breakpoints to another group, or out of a group

To move breakpoints to another group

1. In the [Breakpoints dialog](#), right-click one or more breakpoints you are interested in.
2. On the context menu, point to the command Move to group, and then on the submenu, choose the desired group name:



The breakpoints in question move to the selected group.

To move breakpoints out of a group

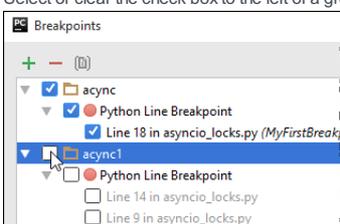
1. In the [Breakpoints dialog](#), right-click one or more breakpoints within a group.
2. On the context menu, point to the command Move to group, and then on the submenu, choose <no group>.
3. The selected breakpoints move to a node according to their [type](#).

Toggling a group of breakpoints

Using groups of breakpoints, it is possible to toggle all breakpoints within a group in a single click.

To toggle all breakpoints belonging to a certain group

- Select or clear the check box to the left of a group name:



Configuring Debugger Options

On this page:

- [Introduction](#)
- [Configuring debugger settings](#)

Introduction

PyCharm supports debugging for Python and Django applications, classes, and files. The debugging functionality is incorporated in PyCharm, you only need to configure its settings.

Depending on the plugins enabled, PyCharm can also support debugging for other languages, for example, JavaScript .

The JavaScript debugging functionality is incorporated in PyCharm, you only need to [configure its settings](#).

PyCharm supports debugging applications running on the built-in or an external web server only in [Google Chrome](#) and other browsers of the **Chrome** family.

Configuring debugger settings

To configure settings required for debugging, perform the following general steps

1. In the [Project Structure](#), configure the roots, dependencies and libraries to be passed to the interpreter.
2. In the Settings/Preferences dialog box, configure the debugger options:
 - Under the Build, Execution and Deployment section, click [Python Debugger](#), and configure the Python debugger options.
 - Under the Build, Execution and Deployment, click [Debugger](#), and define the debugger options as required.
In this section, configure the JavaScript debugger options.

Starting the Debugger Session

On this page:

- [Before debugging](#)
- [Debugging an application](#)

Before debugging

- [Set breakpoints](#) in the source code.
- If necessary, create or modify the corresponding [Run/Debug configuration](#).

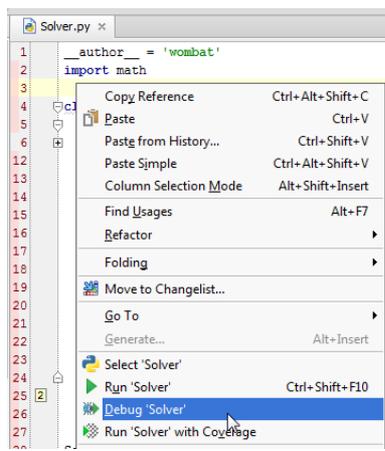
The debug session starts with the selected run/debug configuration. Note that several debug processes can be launched simultaneously.

For example, debugging session for a Python script will start with the default temporary run/debug configuration, unless you select a permanent one.

Debugging an application

To start debugging a Python script

1. Open the desired Python script in the editor, or select it in the Project tool window.
2. On the context menu, choose Debug <script name>:



Note that after you've launched a debug session, the  icon that marks the [Debug Tool Window](#) toggles to  to indicate that the debugging process is active.

Pausing and Resuming the Debugger Session

On this page:

- [Introduction](#)
- [Pausing the debugger session](#)
- [Resuming the debugger session](#)

Introduction

When a breakpoint is hit, or when a running thread or an application is paused manually, the debugging session is suspended.

Pausing the debugger session

To pause the debugger session, do one of the following

- On the main menu, choose Run | Pause Program.
- Click  on the Debug toolbar.
Note that the button is not available for [Run/Debug Configuration: Node JS](#), [Run/Debug Configuration: Node JS Remote Debug](#), and [Run/Debug Configuration: NUnit](#).

Resuming the debugger session

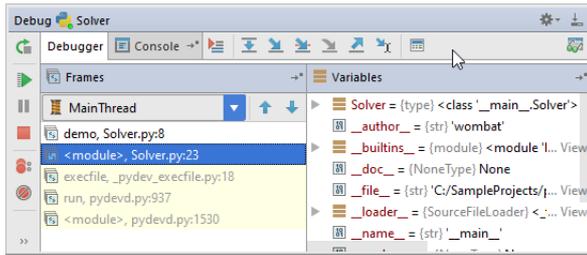
To resume the debugger session, do one of the following

- On the main menu, choose Run | Resume Program.
- Click  on the Debug toolbar.
- Press .

Monitoring the Debug Information

Information of a debugging session is displayed in the dedicated tabs of the [Debug](#) tool window, named after the selected run/debug configuration.

For each session, use the [Console](#) tab to view the debugger messages and application output, and the Debug tab to monitor threads and frames.



Examining Suspended Program

On this page:

- [Basics](#)
- [Examining a suspended thread](#)
- [Navigating between frames](#)

Basics

When a breakpoint is hit, or a program execution is manually [suspended](#), you can [examine](#) your application by analyzing frames.

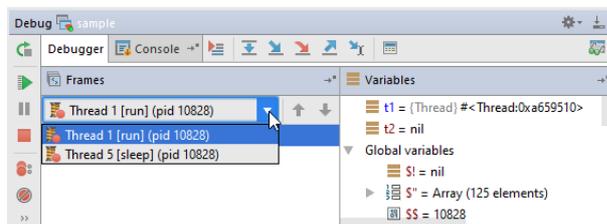
A frame corresponds to an active method or function call. A frame stores the local variables of the called method or function, the arguments to it, and the code context that enables expression evaluation.

All currently active frames are displayed on the Frames pane of the [Debug](#) tool window, where you can [switch](#) between them and analyze the information stored therein.

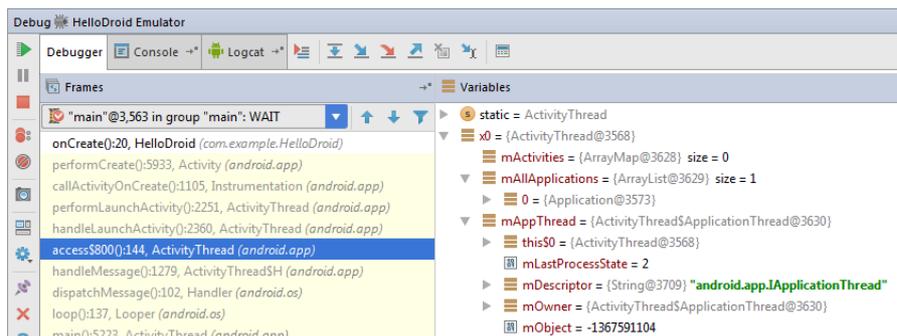
Examining a suspended thread

To examine frames of a suspended thread

1. Select a thread from the thread selector drop-down list on top of the Frames pane. The list of frames is displayed:



2. Select a frame from the Frames list. The Variables pane shows all the variables available to the method call in this frame, so you can further explore them.



Navigating between frames

Do one of the following:

- Use up and down arrow buttons on the toolbar.
- Use [Up](#) and [Down](#) shortcuts.

You do not need to perform any actions to navigate to the frame's source code. PyCharm automatically jumps to the source code of the selected frame in the editor.

Exploring Frames

When a frame is selected in the list, all the values available to this frame's method call are displayed in the Variables pane of the Debug tool window, so you can further explore them. This section describes the ways to simplify examining these values:

- [Adding, Editing and Removing Watches](#)
- [Evaluating Expressions](#)
- [Inspecting Watched Items](#)
- [Navigating to Source Code from the Debug Tool Window](#)

Adding, Editing and Removing Watches

On this page:

- [Introduction](#)
- [Accessing the Watches pane](#)
- [Creating watches](#)
- [Editing watches](#)
- [Deleting watches](#)

Introduction

If you want to evaluate a number of variables or expressions in the context of the current frame, and view all of them simultaneously, you can create **watches** for them. The values of the expressions are updated with each step through the application, but are only visible when the application is suspended. Unlike the [Expression Evaluation feature](#), these expressions are persisted as the part of your project.

This section describes how to add items to watches, change and remove watches.

Accessing the Watches pane

By default, the Watches pane is hidden and the watches are shown in the [Variables pane](#).

- To have the Watches pane displayed separately and view the configured watches in it, release the Show watches in Variables tab toggle button  on the toolbar of the Variables pane. By default, the button is pressed.
- To hide the Watches pane and view the watches in the Variables pane, press the  toggle-button on the toolbar of the Watches pane.

Creating watches

To add an item to a watch

Do one of the following:

- In the [Watches](#) pane, click **+**, or just press .
- On the [Variables](#) pane, in the Inspection window, or in the [Evaluate Expression](#) dialog box, right-click the desired item and choose Add to Watches on the context menu.
- Select the desired item in the Variables pane and drag it to the Watches pane.
- Select item in the editor, right-click it and select Add to Watches on the context menu.

Tip You can navigate from a backtrace in the Watches pane to the respective line of the source code. To do that, right-click a line of backtrace, and choose Jump to Source on the context menu, or just press .

Editing watches

To edit a watch

- To change the expression represented by a watch, right-click the desired watch and select Edit on the context menu.

Deleting watches

To remove a watch

1. In the Watches pane, select a watch to be deleted.
2. On the context menu, choose Remove Watch, or press .

Evaluating Expressions

On this page:

- [Basics](#)
- [Limitations](#)
- [Evaluating expressions or code fragments in a stack frame](#)
- [Evaluating arbitrary expressions](#)
- [Evaluating expressions in the editor](#)

Basics

PyCharm enables you to evaluate expressions and code fragments in the context of a stack frame currently selected in the [Frames pane](#) of the [Debug tool window](#).

In addition to regular expressions, you can also evaluate operator expressions, lambda expressions, and anonymous classes.

The following evaluation modes are available:

- **Expression Mode** for evaluating single-line expressions.
- **Code Fragment Mode** for evaluating short code portions. You can evaluate declarations, assignments, loops and `if/else`.

Besides, PyCharm provides a way to [quickly evaluate](#) an expression in the editor at caret or a selection.

Limitations

While using the Expression Evaluation feature, be aware of the following:

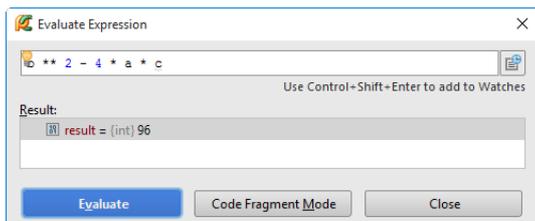
- A method can be invoked within the Expression Evaluation dialog only if the debugger has stopped at a breakpoint, but has not been paused.
- Expression Evaluation can only be "single-level". In other words, if PyCharm stops at a breakpoint within a method called from the Expression Evaluation, you cannot use the Expression Evaluation feature again.

Tip Note that in certain operating systems the key and mouse combinations may not work as described here. In this case, it's necessary to tweak the operating system's keymap. For example, if you are using Ubuntu, mind the windows manager, whose [shortcuts conflict](#) with that of PyCharm.

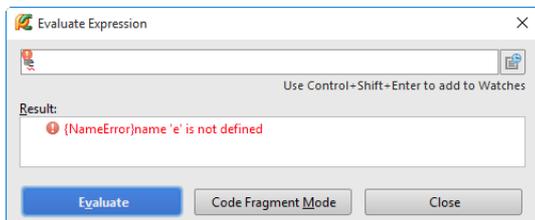
Evaluating expressions or code fragments in a stack frame

To evaluate an expression or a code fragment in a stack frame, do the following:

1. In the [Frames](#) pane, select the thread where you want an expression to be evaluated.
2. Invoke the [Evaluate Expression](#) command in one of the following ways:
 - On the main menu, choose Run | Evaluate Expression
 - On the context menu of the editor, choose Evaluate Expression
 - Press `Alt+F8`
 - Click  on the [Stepping toolbar](#) in the [Debug tool window](#).
3. Select an evaluation mode. If you want to evaluate a code fragment, click the Code Fragment Mode button.
4. Depending on the selected mode, type the expression or statements to evaluate in the text field and click Evaluate.



If the specified expression cannot be evaluated, the possible reason will be briefly described in the Result pane of the dialog box.



Tip If you have [assigned a label](#) to a variable, object, or watch, you can reference it by this label as if it were a local variable `<label-name>_DebugLabel` defined in the same context where the expression is evaluated. PyCharm also displays this label in the completion suggestion list.

Evaluating arbitrary expressions

1. Open the [Evaluate Expression](#) dialog box in one of the following ways:
 - Choose Run | Evaluate Expression on the main menu.
 - Press `Alt+F8`.
 - To evaluate a specific variable, select it on the [Variables pane](#) of the [Debug tool window](#), then choose Run | Evaluate Expression or press `Alt+F8`.
2. In the Evaluate Expression dialog box, specify the expression you want to evaluate. Do one of the following:

- In the Expression field, type the expression in question or choose one of the previously evaluated expressions from the drop-down list. If you have selected a specific variable on the Variables pane, this variable will be displayed in the Expression text box.
 - To evaluate a code fragment, click the Code Fragment Mode button and fill in the Code Fragment text box. To return to the original mode, click the Expression mode button.
3. Click the Evaluate button. The Result read-only field shows the evaluation output. If the specified expression cannot be evaluated, the Result field explains the reason.

Evaluating expressions in the editor

During a debugger session, the value of any expression is shown in the tooltip every time you hover your mouse pointer over it. If an expression contains children, clicking **+** expands the node and displays all children.

```

c = (int) 1 = 0:

```

You can also use the Quick Evaluate expression functionality that lets you view the value of an expression using the keyboard only.

There are two ways to evaluate an expression quickly:

1. By using the Show value tooltip on code selection functionality:

- a. In the [Debugger | Data Views](#) settings page, enable the Show value tooltip on code selection option.
- b. Select a code fragment with the mouse, or by clicking **Ctrl+W**. A tooltip with the expression value automatically appears under the selection and changes each time you change the selection:

```

conversionResult = (java.lang.String[2]@445)
conversionResult[0] = null
conversionResult[1] = <FormatException nfe) {

```

2. By manually invoking the tooltip with the expression value:

- a. Place the caret at the desired location, or select an expression to be evaluated.
- b. Choose **Run | Quick Evaluate Expression** on the main menu, or press **Ctrl+Alt+F8**. The tooltip with the expression value appears under the selected expression.

Inspecting Watched Items

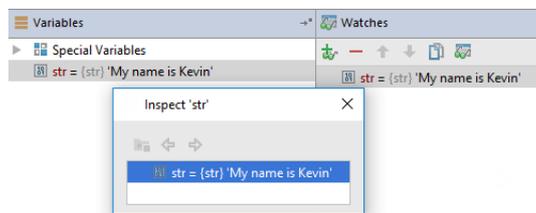
On this page:

- [Introduction](#)
- [Accessing the Watches pane](#)
- [Inspecting references](#)

Introduction

PyCharm helps inspect any variables or watches item in its own window. For example, if you need to examine several references in detail, you can open an inspection window for each of them. So doing, a separate window is created for each variable or watch reference and all of its child references.

The inspection windows are non-modal, and you can launch as many as of them required. All changes of the references are immediately reflected in the corresponding inspection windows.



Accessing the Watches pane

By default, the Watches pane is hidden and the watches are shown in the [Variables pane](#).

- To have the Watches pane displayed separately and view the configured watches in it, release the Show watches in Variables tab toggle button  on the toolbar of the Variables pane. By default, the button is pressed.
- To hide the Watches pane and view the watches in the Variables pane, press the  toggle-button on the toolbar of the Watches pane.

Inspecting references

To inspect a reference

1. Select the item to be inspected on the Variables or Watches pane.
2. On the context menu, choose Inspect.

To navigate to the source code, do one of the following:

- Select the desired item in the [Variables](#) tab and press **F4**.
- Right-click an item in the Variables tab, and select Jump to Source from the context menu.

Finding the Current Execution Point

When a program is suspended, the source file, associated with the current execution point, is opened in the editor. The current execution point (the next line to be executed) is marked with a blue line.

You can visit the other files, and then return to the current execution point using the actions described in this section.

To find the current execution point, do one of the following

- On the main menu, choose Run | Show Execution Point.
- Press `Alt+F10`.
- Click  on the [stepping toolbar](#) of the Debug tool window.

Stepping Through the Program

In this section:

- Stepping Through the Program
 - [Introduction](#)
 - [Stepping through the program](#)
 - [Tips and tricks](#)
- [Choosing a Method to Step Into](#)

Introduction

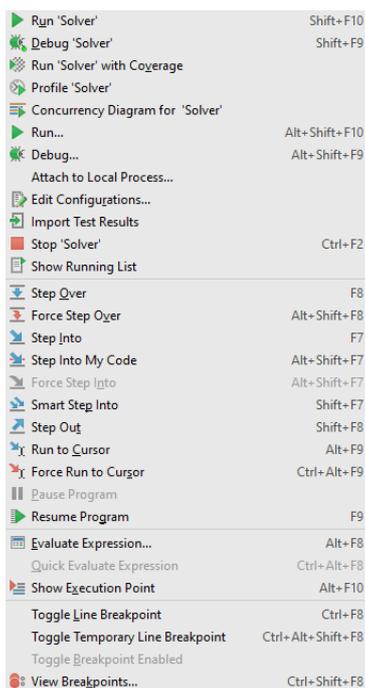
When a breakpoint is reached or your program is [suspended](#), the [Debug](#) tool window becomes active and enables you to get control over the program's execution. For this purpose, you can use the Run menu commands, or the icons on the [stepping toolbar](#) of in the Debug tool window.

Each stepping action advances the [execution point](#) to the next execution location, depending on the action you choose.

Stepping through the program

Do one of the following:

- On the main menu, choose Run | <stepping command>



- Use the [keyboard shortcuts](#).
- Use the buttons in the [stepping toolbar](#) of the Debug tool window.



Tips and tricks

- The Force Step Into command  enables you to step into a method of a class not to be stepped into .

The classes, stepping into which is suppressed, are specified on the [Stepping](#) page of the [Settings/Preferences](#) dialog box.

- The Force Step Over command  enables you to jump over the method call ignoring the breakpoints on the way.
- The Force Run to Cursor command  enables you to jump to the cursor position ignoring existing breakpoints on the way.
- Step Into My Code command  helps skip stepping into library sources and keep focused on your own code.

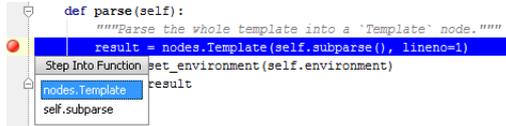
Choosing a Method to Step Into

When you reach a line with calls of several methods, you can choose the method you want to step into.

Tip If you choose a method of a class stepping into which is suppressed on the [Stepping](#) page of the [Settings](#) dialog box, the suppression is overridden as when you invoke the Force Step Into command.

To choose a method to step into

1. On the main menu, choose Run | Smart Step Into or press `Shift+F7`.
2. In the pop-up window, choose the desired method from the list.



```
def parse(self):  
    """Parse the whole template into a 'Template' node."""  
    result = nodes.Template(self.subparse(), lineno=1)  
    Step Into Function get_environment(self.environment)  
    result  
nodes.Template  
self.subparse
```

Remote Debugging

This feature is supported in the Professional edition only.

PyCharm provides two ways to debug remotely:

- [Using a remote interpreter](#)
- [Using Python Debug Server](#)

Remote debug with a remote interpreter

If remote debugging is performed with a remote Python interpreter, then everything is done within a single SSH connection, and port numbers are not required.

To debug remotely using a remote interpreter, follow these general steps

1. Make sure that a [remote interpreter is configured](#).
2. [Launch the debug process](#) with a regular run/debug configuration.

Tip The Python debugger is available in [PyE](#) so that it can be installed for doing remote debugging with `pip`.

When debugging a process that runs in another machine, it's possible to `pip install pydevd` instead of copying archives from `debug-eggs` directory under PyCharm's installation.

Remote debug with a Python Debug Server

For remote debugging, PyCharm provides archives with `pydev` directory. Depending on the Python version, these archives are:

- `pycharm-debug.egg` for Python up to version 2
- `pycharm-debug-py3k.egg` for Python 3

This archive resides in the `debug-eggs` directory under the PyCharm installation.

The process of remote debugging involves the following general steps:

- [Configuring a remote debug server](#).
- [Preparing for debugging](#).
- [Launching the debug server](#).
- [Launching the script externally](#).

To configure a remote debug server

1. Open the Run/Debug Configuration dialog box, as described in the section [Creating and Editing Run/Debug Configurations](#), and select the [Python Remote Debug](#) configuration type.
2. Specify the local host name and the number of the port where the debug server will run. If you don't specify any value, PyCharm will provide port number at random.
3. If you are going to debug a script that resides on a remote machine, specify the source root of the remote script, and the root of the local sources to keep mapping.

To prepare for remote debugging

1. In PyCharm, open the local script for editing, and add the following command (you can copy it from the remote debug configuration dialog):
`pydevd.settrace(<host name>, port=<port number>)`

Also, add the corresponding import statement to your script:

```
import pydevd
```

2. Copy the local script to the remote location, where you want to debug it.
3. Include the `pycharm-debug.egg` archive. You can do it in a number of ways, for example:
 - Add the archive to `PYTHONPATH`.
 - Append the archive to `sys.path`.
 - Just copy the `pydev` from the archive to the directory where your remote script resides.

To launch the debug server

1. Select the desired remote configuration:



2. Click , or press `Shift+F9`. PyCharm launches debug server at the specified host and port, which is shown in the Console pane of the

[Debug tool window](#).

If the process doesn't stop, but you still want to stop tracing and disconnect from Python Remote Debug Server, add the following function to the end of a remote script being debugged:

```
pydevd.stoptrace()
```

This function stops remote debug process that has been launched with `pydevd.settrace()`. Thus the Python Remote Debug Server will pass to the state of waiting for a new connection.

To debug a remote script

1. Locate your remote script, and launch it as required by your specific operating system. For example, on Windows, use the command:

```
python <script name>
```

If the corresponding local script exists, PyCharm opens it in the editor in the suspended mode; the first executable statement is marked with the blue stripe. Also, PyCharm shows the frames and variables for its first executable statement in the [Debug tool window](#). If the corresponding local script cannot be found, PyCharm opens a dedicated editor tab, and suggests you to do one of the following:

- Edit the local and remote roots in the remote debug configuration dialog box.
- Detect the mapping files automatically, and then select the one to be used as the local version from the list of encountered files with the matching names.
- Download the remote file to the specified location.

2. [Step through the script](#) and [explore frames](#).

Inline Debugging

On this page:

- [Basics](#)
- [Enabling inline debugging](#)
- [Viewing inline debugging results](#)

Basics

The **inline debugging** functionality facilitates the debugging procedure, as it lets you view the value of variables used in your source code right next to their usage, without having to switch to the [Variables pane](#) of the [Debug tool window](#).

Enabling inline debugging

To enable the **inline debugging** functionality, do one of the following:

- In the [Debug tool window](#) toolbar, click the Settings icon  and select the Show Values Inline option from the popup menu.
- Open the [Data Views](#) page of Setting/Preferences dialog, and select the check box Show values inline.

Viewing inline debugging results

If this option is enabled, when you launch a debug session and [step](#) through the program, the values of variables are displayed at the end of the lines where these variables are used.

```
class Solver:
    def demo(self): self: Solver: <__main__.Solver
        while True:
            a = int(input("a ")) a: 1
            b = int(input("b ")) b: 1
            c = int(input("c ")) c: 1
            d = b ** 2 - 4 * a * c d: -3
            if d >= 0:
                disc = math.sqrt(d)
```

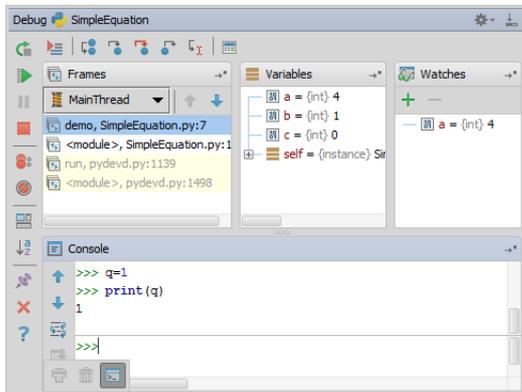
Using Debug Console

The Debug Console enables you to view the output and error messages. You can optionally make command line prompt available.

If in course of the debugging session, you need to work with an interactive debugger console, do one of the following:

- In the toolbar of the [debug console](#), click the button .
- On the main menu, choose Tools | Open Debug Command Line

The console becomes interactive: it shows prompt, where you can type commands, execute them, and review results:



In an interactive console, you can

- Type commands in the lower pane of the console, and press `Enter` to execute them. Results are displayed in the upper pane.
- Use [basic code completion](#) `Ctrl+Space`.
- Use Up and Down arrow keys to scroll through the history of commands, and execute the required ones.
- [Load](#) source code from the editor into console.
- Use context menu of the upper pane to copy all output to the clipboard, compare with the current contents of the clipboard, or remove all output from the console.
- Use the toolbar buttons to control your session in the console.

Attaching to Local Process

On this page:

- [Introduction](#)
- [Attaching to local process](#)

Introduction

PyCharm makes it possible to attach to a Python process, while running a Python script launched either outside of PyCharm, or inside PyCharm, but NOT in the debug mode.

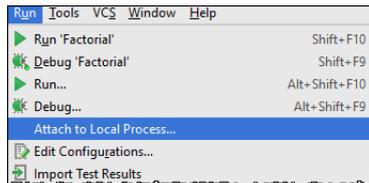
Attaching to local process

To attach to a local process, follow these general steps:

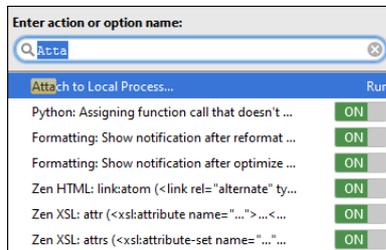
1. Launch the process intended for debugging. You can do it from operating system or using the PyCharm terminal.

2. To find the process to attach to, do one of the following:

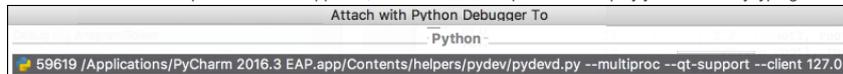
- On the main menu, choose Run | Attach to Local Process:



- On the main menu, choose Help | Find Action or press `Ctrl+Shift+A`. In the list of actions that appears, find the desired action by typing the first letters, and select it:



3. From the list of available processes that appears, select the desired process. Simplify your search by typing the first letters of its name or PID



4. Proceed with [debugging](#) the same way as you usually do it in PyCharm ([set breakpoints](#), [step through](#), [pause and resume](#) the process, [evaluate expressions](#) etc.)

5. When finished, detach the process: select the Run | Stop or click the Stop the process button  of the [Debug Tool Window](#).

Thread Concurrency Visualization

This feature is supported in the Professional edition only.

In this section:

- [Overview](#)
- [Starting the concurrency visualization session](#)
- [Concurrency visualization for the applications that use asyncio](#)
- [Working with the Concurrent Activities Diagram tool window](#)
- [Graphs' context menu](#)

Overview

This feature helps gain full control over the multi-threaded applications. The concurrency visualization session runs with the current run/debug configuration in the **Concurrency Diagram** mode.

Starting the concurrency visualization session

1. Do one of the following:
 - On the main menu, choose Run | Concurrency Diagram for <script name>.
 - On the context menu of the editor, choose Concurrency Diagram for <script name>.
 - Provided that the main toolbar or the navigation bar is visible, click .

The concurrency visualization diagram shows the real time states of threads inside the running process in the Threading graph tab of the Concurrent Activities Diagram tool window.

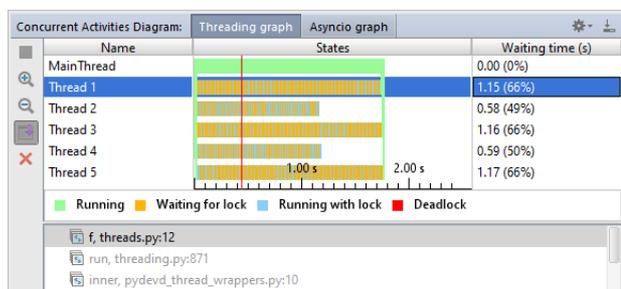
Concurrency visualization for the applications that use asyncio

Concurrency visualization also works well with the `asyncio` module introduced in Python 3.4.

To make use of the concurrency visualization, run the application that uses `asyncio` same way as described above, and then switch to Asyncio graph tab.

Working with the Concurrent Activities Diagram tool window

- The left-hand side of the tool window contains the toolbox with the following icons:
 -  - click this button to terminate the running process.
 -  - click these buttons to increase or decrease scale of the diagram.
 -  - when the diagram is drawn for a long time and output is too long, then a scrollbar appears. When this button is pressed, this scrollbar is automatically scrolled to the end.
 -  - click this button to close the tool window.
- Next to the toolbox, there is the list of thread names; the right-hand side shows the waiting time for each thread:



- Different states are marked with different colors. The legend is shown in the window.
- To zoom in and out, use the magnifier glass icons in the toolbox, or `Ctrl+mouse wheel`.
- To navigate to a particular stack frame, click the diagram.

Graphs' context menu

Right-clicking on a graph invokes a context menu with the following commands:

ItemDescription

Show related locks This command highlights on a graph all the thread expectations, which work with the same lock where a context menu has been invoked.

Hide related locks This command hides the highlighting described above.

Viewing as Array or DataFrame

In this section:

- [Limitations](#)
- [Viewing as array or DataFrame](#)
 - [From the Variables tab of the Debug tool window](#)
 - [From the Python console](#)
- [Actions available via the Data View tool window](#)

Limitations

This command is available for:

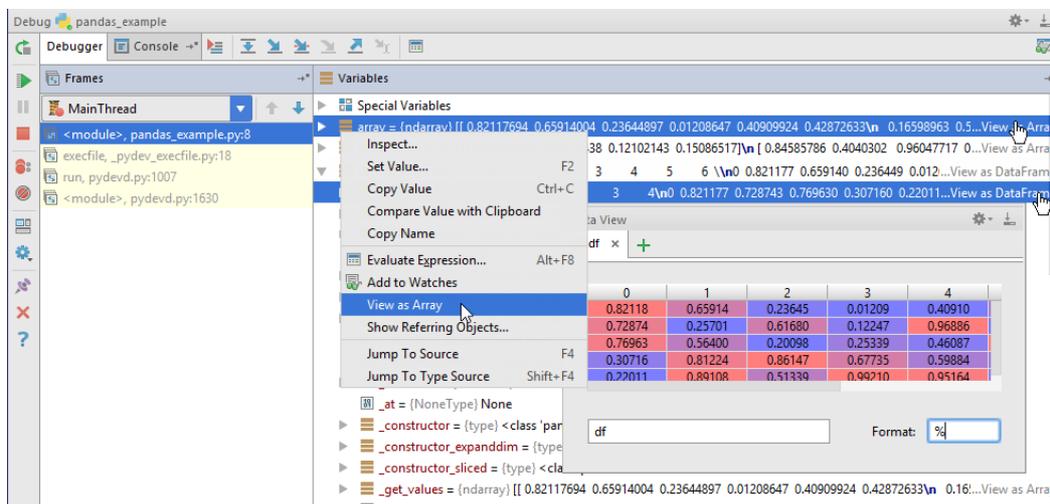
- The variables that represent [NumPy](#) arrays.
- The variables that represent [pandas dataframes](#).

Hence, NumPy and/or pandas must be downloaded and installed in your Python interpreter.

Viewing as array or DataFrame

To use the command View as Array/View as DataFrame, follow these steps:

1. [Launch the debugger session.](#)
2. In the [Variables tab of the Debug tool window](#), select an array or a DataFrame.
3. Click a link View as Array/View as DataFrame to the right.



Alternatively, you can choose View as Array or View as DataFrame on the context menu.

The Data View tool window appears.

One can also use the command View as Array from the Python console.

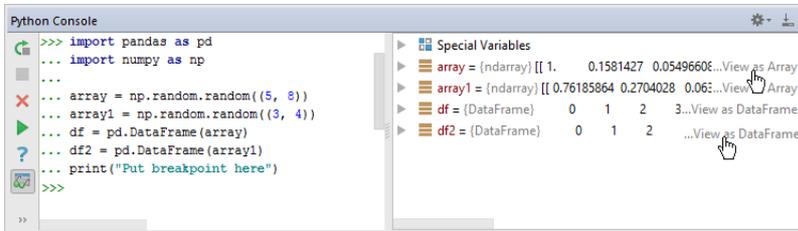
To view as array from the Python Console, follow these steps:

1. [Launch the Python Console.](#)
2. Execute a Python code, for example:

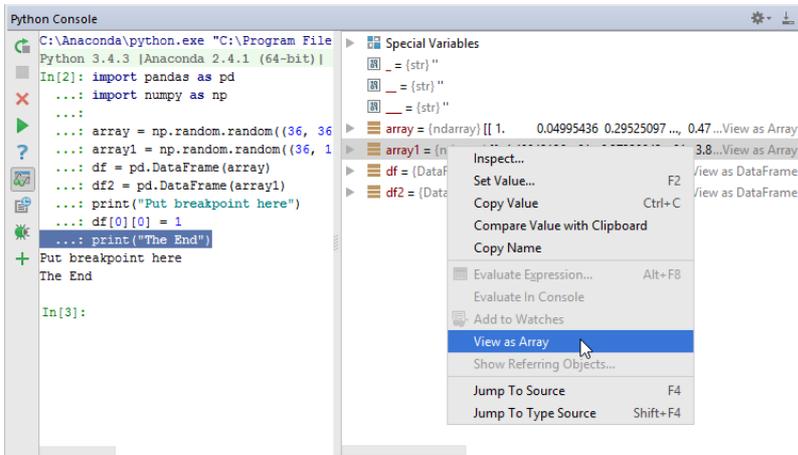
```
import pandas as pd
import numpy as np

array = np.random.random((36, 36))
array1 = np.random.random((36, 10))
df = pd.DataFrame(array)
df2 = pd.DataFrame(array1)
print("Put breakpoint here")
df[0][0] = 1
print("The End")
```

3. In the toolbar of the console, click . The variables declared in the console, appear to the right.
4. Do one of the following:
 - Click the link View as Array/View as DataFrame:



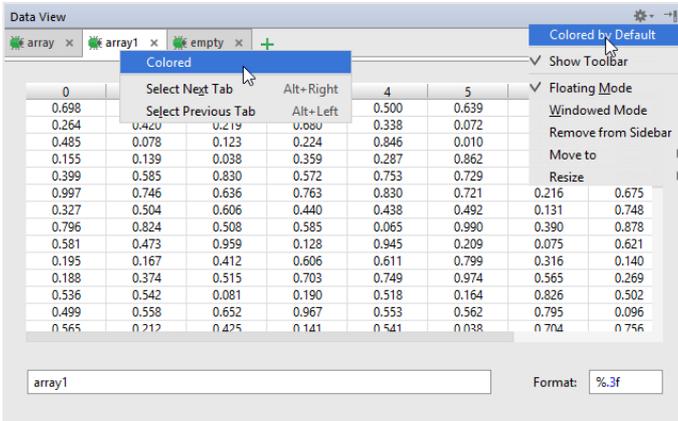
– On the context menu of a variable, choose View as Array/ View as DataFrame:



Actions available via the Data View tool window

In the Data View tool window, one can do the following:

- Change the format of presentation. For example, if in the Format field one specifies `% .5f`, then 5 digits will appear after dot; if one specifies `% .2f`, the presentation of the data will change to showing 2 digits after dot. See [Python documentation](#) for details.
- Close the viewer tab by clicking `x`, and open a new one by clicking `+`.
- Make the presentation black-and-white in the **new** tabs by clearing the check-command Colored by Default that appears in the drop-down menu . If this command is cleared, and a new DataFrame/array is opened, then the new presentation will be colorless.
- It's possible to change from the colored to colorless modes for **the current** tab by right-clicking a tab and selecting the check-command Colored:



Data View is a tool window, and as such, it inherits all the behaviors that are common to all the tool windows. Refer to the section [PyCharm Tool Windows](#) to learn more.

Testing

PyCharm provides support for several language-specific [testing frameworks](#). This section covers the issues that are common for all the supported testing frameworks:

- [Creating Tests](#)
- [Creating Run/Debug Configuration for Tests](#)
- [Running Tests](#)
- [Rerunning Tests](#)
- [Terminating Tests](#)
- [Monitoring and Managing Tests](#)
- [Viewing and Exploring Test Results](#)

For the framework-specific testing procedures, refer to the sections:

- [Testing Frameworks](#)

Creating Tests

PyCharm suggests a way to create tests for classes and individual methods.

To create a test for a class or method, follow these general steps

1. In the editor, place the caret at the class declaration, or somewhere within a method.
2. Do one of the following:
 - On the main menu, choose `Navigate | Test`.
 - On the context menu, choose `Go To | Test`.
 - Press `Ctrl+Shift+T`.

PyCharm shows the list of available tests.

3. If the desired test doesn't yet exist, click `Create new test`.
The `Create Test` dialog box opens.
4. In the `Create Test` dialog box, specify the following settings:
 - Target directory, where the new test class will be generated.
 - The name of the test file, and the name of the test class.
 - Select the check boxes next to the methods you want to include in the test class.

Note that if the caret has been placed within a method, only this method name is included in the list.

5. Click `OK` when ready. PyCharm generates the test file in the specified location.

Creating Run/Debug Configuration for Tests

On this page:

- [Introduction](#)
- [Creating a test configuration](#)

Introduction

You can run your tests (test cases, test suites, etc.) using run/debug configurations, in the way similar to running ordinary applications. PyCharm provides a framework for creating special run/debug configurations for testing purposes, where a test can be specified as a target.

In addition to the regular procedure described in the section [Creating and Editing Run/Debug Configuration](#), PyCharm provides a shortcut that allows you to create run/debug configurations for all tests in a container, for a single test case, or even for a test method.

Creating a test configuration

To create a test configuration

1. Right-click the desired target, for example a directory or an individual test in the Project tool window. In case of an individual test, you can open it in the editor, and right-click the background.
2. On the context menu of the selection, choose Create <name> for an individual test. For your convenience, the corresponding context menu commands are marked with icons:
 -  for Python unittest
 -  for Python doctests
 -  for Python nosetest
 -  for Django unit tests
 -  for py.test framework

Tip For a test method, open the class in the editor and right click anywhere in the method. The context menu suggests the command Create < method name>.

3. In the dialog box that opens, specify the run/debug configuration parameters, apply changes and close the dialog.

 code completion works in run/debug configuration dialog boxes, for example in the Working directory field.



Running Tests

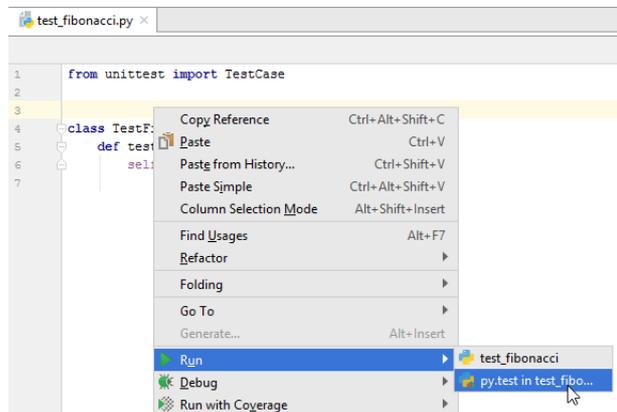
In this section:

- [Introduction](#)
- [Running all tests in a directory](#)
- [Running test cases or test scripts](#)

Introduction

PyCharm makes it possible to run all tests in a container, individual tests or test methods. For each one, PyCharm provides a temporary run-debug configuration that can be saved if necessary as a permanent one.

Note also that the commands shown in the context menu, are context-sensitive, that is the testing command that shows depends on the test runner and the place where this command is invoked.

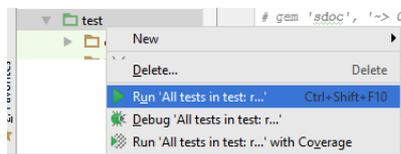


Running all tests in a directory

To run all tests in a directory

1. In the Project tool window, select the directory that contains tests to be executed.
2. On the context menu, choose the corresponding run command.
If the directory contains tests that belong to the different [testing frameworks](#), select the configuration to be used.

For example, choose Run 'All tests in: <directory name>'.



3. [Explore results](#) in the test runner.

Tip For Django versions 1.1 and higher, PyCharm supports custom test runner, if this test runner is a class.

Running test cases or test scripts

To run a test case or test script

1. Open the desired test in the editor, or select it in the Project tool window.
2. On the context menu of the selection, choose Run <test class name>.

Rerunning Tests

On this page:

- [Introduction](#)
- [Rerunning a testing session](#)
- [Rerunning an individual test](#)
- [Rerunning a failed test](#)
- [Debugging a failed test](#)

Introduction

You can repeat your test session, or individual tests without leaving your [test runner tab](#) of the Run tool window. The tests are performed again using the same run configuration as in the initial run.

Rerunning a testing session

Do one of the following:

- Click the rerun button  on the toolbar of the Run tool window.
- Use `Ctrl+F5` keyboard shortcut.

Rerunning an individual test

1. In the testing tab of the test runner, right click a test case node or a test.
2. On the context menu, choose Run <test target>.

Rerunning a failed test

1. In the testing tab of the test runner, select a failed test.
2. In the [Run toolbar](#), click Rerun Failed Tests .

Tip PyCharm makes it possible to rerun/run/debug configuration of a test automatically, if the source code has been changed. To enable autotest-like runner facility, make sure that the Toggle auto-test button  in the [Run toolbar](#) of the Test Runner tab is pressed.

Debugging a failed test

1. In the [Test Runner tab](#), press `Shift` and click Rerun Failed Tests .
2. Select Debug from the Restart Failed Tests popup.

The tests that have failed will be rerun in the debug mode.

Terminating Tests

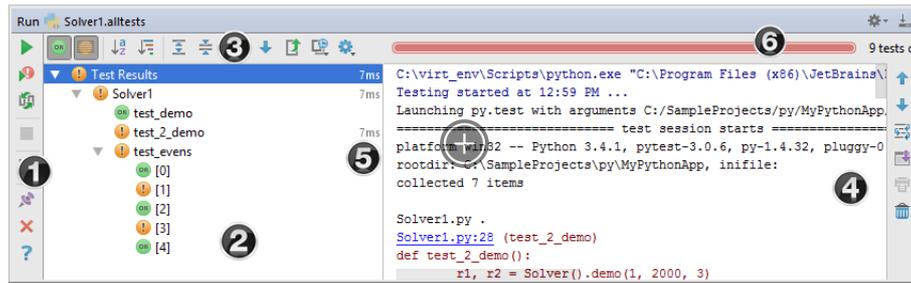
You can abort a running test session at any time. So doing, all tests that are current running will stop immediately. The icons of the tests in the test runner reflect the statuses of the tests (passed, failed, aborted, or never run).

To stop a testing session, do one of the following

- On the toolbar of the test runner, click the stop button .
- Use `Ctrl+F2` keyboard shortcut.

Monitoring and Managing Tests

Test progress and results display in the dedicated [test runner tabs](#) of the Run tool window.



You can [rerun](#), [terminate](#), and [suspend](#) execution of tests same way as you do it for running applications. In addition to the common running actions, in the test runner you can:

- Navigate through the list of test cases using arrow keys.
- Navigate between failed tests using the  and  buttons or `Ctrl+Alt+Up` or `Ctrl+Alt+Down` keyboard shortcuts.
- View the total number of tests being run in the current session. The summary information is displayed in a message line on the top of the tool window. After completion of the tests, the message informs you about the number of failed tests and elapsed time.
- View testing progress in the progress bar.
- Show or hide information about the passed tests, using the  button.
- Show the ignored tests in the tree view of all tests within the current run/debug configuration or test class using the  button.
- Navigate from the stack trace to the problem location in the source code by clicking the hyperlink in the Output pane.
- Enable and disable the following functionality by clicking the cog button  and selecting the relevant items from the context menu:
 - Monitoring execution of the current test.
 - Have the first failed test selected automatically upon completing the suit.
 - Navigating to the stack trace.
 - Automatic scrolling to the source code.
 - Have the corresponding source code opened at exceptions.
 - Have statistic shown in the Statistics pane.

Viewing and Exploring Test Results

On this page:

- [Overview](#)
- [Viewing statistics](#)
- [Important note](#)

Overview

Depending on the selected node in the tree view of tests, the Test Runner displays information on the various levels. In the Test Runner, you can view statistics of the tests, navigate to stacktrace, show or hide successful tests, and more.

Use the [Testing toolbar](#) to control visual representation of the test results.

Viewing statistics

The Statistics tab shows information about the elapsed time and memory usage of each test.

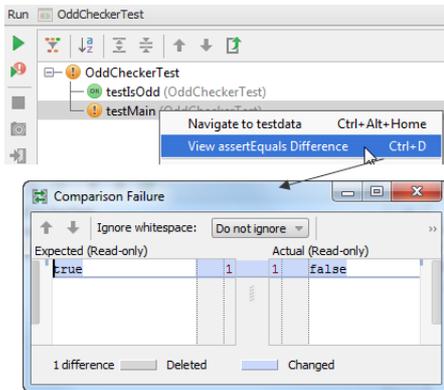
To view statistics, click  on the Testing toolbar to reveal the drop-down menu, and then click the Show Statistics check command.

The values displayed in the Statistics tab are not accurate and only give an approximate estimate of the test performance. For example, if a garbage collector works during the test run, the memory usage shown in the Statistics tab is wrong.



Important note

If a unit test contains string `assertEquals` failures, the test runner provides the ability to view differences between the compared strings. Choose the View assertEquals Difference on the context menu of the failed test that contains `assertEquals`, and explore differences in the [Differences viewer](#).



Deployment. Working with Web Servers

In this section:

- Deployment. Working with Web Servers
 - [Basics](#)
 - [Interaction between PyCharm and servers](#)
- [Configuring Synchronization with a Web Server](#)
- [Uploading and Downloading Files](#)
- [Accessing Files on Web Servers](#)
- [Comparing Deployed Files and Folders with Their Local Versions](#)
- [Editing Individual Files on Remote Hosts](#)
- [Running SSH Terminal](#)

Basics

Among numerous ways to configure your development and production environments the most frequent ones are as follows:

- The Web server is installed on your computer. The sources are under the server document root (for example, `/htdocs`), and you do your development right on the server.
- The Web server is installed on your computer but the sources are stored in another folder. You do your development, then copy the sources to the server.
- The Web server is on another computer (remote host). Files on the server are available through the FTP/SFTP/FTPS protocol, through a network share, or a mounted drive.

Now let's see how to use PyCharm in the above environment configurations. PyCharm assumes that all development, debugging, and testing is done on your computer and then the code is deployed to a production environment.

Please note the following:

- The reason to stick to this "local development - deployment" model lies in the way PyCharm provides its coding assistance which includes code completion, code inspections & validations, code navigation, etc. All this functionality is based on the **index of the project files** which PyCharm builds when the project is loaded and updates on the fly as you edit your code.
- To provide efficient coding assistance, PyCharm needs to re-index code fast, which requires fast access to project files. The latter can be ensured only for **local** files, that is, files that are stored on you hard disk and are accessible through the file system. Therefore PyCharm does not support the mode when you access your files over a network folder (very often it becomes slow and unresponsive, performs random look-ups for no obvious reason, etc).

Interaction between PyCharm and servers

Interaction between PyCharm and servers is controlled through **server access configurations**. Anytime you are going to use a server, you need to define a **server access configurations**, no matter whether your server is on a remote host or on your computer.

Taking into account all the above, let's define the following basic concepts related to synchronization between PyCharm and servers.

- An **in-place server** is a server whose **document root** is the parent of the project root, either immediate or not. In other words, the Web server is running on your computer, your project is under its document root, and you do your development directly on the server.
- A **local server** is a server that is running in a local or a mounted folder and whose **document root** is **NOT** the parent of the project root.
- A **remote server** is a server on another computer (remote host).
- The **server configuration root** is the highest folder in the file tree on the **local** or **remote** server accessible through the server configuration. For **in-place** servers, it is the project root.
- A **local file/folder** is any file or folder under the project root.
- A **remote file/folder** is any file or folder on the server, either local or remote.

Suppose you have a project `C:/Projects/My_Project/` with a folder `C:/Projects/My_Project/My_Folder` and a local server with the document root in `C:/xampp/htdocs`. You upload the entire project tree to `C:/xampp/htdocs/My_Project`. In the terms of PyCharm, the folder `C:/Projects/My_Project/My_Folder` is referred to as **local** and the folder `C:/xampp/htdocs/My_Project/My_Folder` is referred to as **remote**.

- **Upload** is copying data from the project **TO** the server, either local or remote.
- **Download** is copying data **FROM** the server to the project.

After you have configured synchronization with a server, you can upload, download, and manage files on it directly from PyCharm. Moreover, you can suppress uploading or downloading specific files or entire folders. Finally, you can optimize you workflow by configuring content roots so specific folders are not involved in indexing, which significantly saves project indexing time.

Synchronization with servers, uploading, downloading, and managing files on them are provided via the Remote Hosts Access bundled plugin, which is by default enabled. If the plugin is disabled, activate it in the Plugins page of the Settings dialog box. For details, see [Enabling and Disabling Plugins](#).

Configuring Synchronization with a Web Server

In this section:

- Configuring Synchronization with a Web Server
 - [Before you start](#)
 - [Basics](#)
 - [Server access configuration](#)
- [Creating an In-Place Server Configuration](#)
- [Creating a Local Server Configuration](#)
- [Creating a Remote Server Configuration](#)
- [Customizing Upload/Download](#)
- [Excluding Files and Folders from Upload/Download](#)

Before you start

Synchronization with servers, uploading, downloading, and managing files on them are provided via the Remote Hosts Access bundled plugin, which is by default enabled. If the plugin is disabled, activate it in the Plugins page of the Settings dialog box. For details, see [Enabling and Disabling Plugins](#).

Basics

PyCharm distinguishes among **in-place**, **local**, and **remote** servers, however the meaning of these terms in the context of PyCharm slightly differs from their common meaning:

- An **in-place server** is a server whose **document root** is the parent of the project root, either immediate or not. In other words, the Web server is running on your computer, your project is under its document root, and you do your development directly on the server.
- A **local server** is a server that is running in a local or a mounted folder and whose **document root** is **NOT** the parent of the project root.
- A **remote server** is a server on another computer (remote host).

For more information about possible configuration of the production and development environment and working with servers from PyCharm, see [Deployment. Working with Web Servers](#).

Server access configuration

PyCharm controls interaction with Web servers through **server access configurations**. Anytime you are going to use a server, you need to define a server access configurations, no matter whether your server is on a remote host or on your machine.

A **server access configuration** defines:

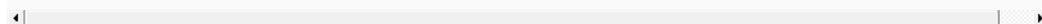
- The server type (**in-place**, **local**, or **remote**).
- The computer (host) where the server is running. For **in-place** and **local** servers, PyCharm presupposes that it is the current computer where your project is.
- The server access configuration root: the highest folder in the server hierarchy that can be accessed through the server configuration.
- The URL address to access the server configuration root. Both the **HTTP** and the **HTTPS** protocols are supported.

To access a server through HTTPS, you need to acquire a certificate file `<certificate_name>.cert` signed by a recognized **authority** and import this certificate in the [truststore/keystore](#) of the Oracle JRE (Java Runtime Environment) on which PyCharm runs. Note that self-signed certificates are rejected as unsafe.

To import a certificate in Oracle JRE:

1. Open the embedded Terminal and type the following command:

```
<jre_home>/bin/keytool.exe -importcert -keystore <path to jre truststore/keystore> -file <full_path_to_<cert_name>.cert
```



If you are using the **Oracle JRE** bundled with PyCharm, the default path to the truststore/keystore is

```
<%product_installation_folder>/jre/jre/lib/security/jssecacerts or
```

```
<%product_installation_folder>/jre/jre/lib/security/cacerts .
```

Otherwise it is `<jre_home>/jre/lib/security/jssecacerts` or `<jre_home>/jre/lib/security/cacerts` .

2. When asked to enter a password for the truststore/keystore, specify the default one `changeit` .
3. Open the `PyCharm.exe.vmoptions` file in the `<PyCharm_installation_folder>/bin` and add the following line to it:

```
-Djavax.net.ssl.keyStore=<path to keystore>
```

4. Restart PyCharm.

Learn more at [Java6](#) and [Java7](#).

- The protocol to transfer the data through.
- Correspondence between local (project) folders, destination folders on the server, and URL addresses to access the data on the server. This correspondence is called **mapping**.

You can define as many configurations as necessary, thus enabling flexible switching between upload/download setups.

On this page:

- [Basics](#)
- [Creating a server configuration: specifying its name, type, and visibility](#)
- [Configuring access to an in-place server: specifying the URL address of the server document root](#)
- [Specifying the project root folder and the URL address to access it](#)

Basics

An **in-place server** is a server whose **document root** is the parent of the project root, either immediate or not. In other words, the Web server is running on your computer, your project is under its document root, and you do your development directly on the server.

To configure access to the server in this set-up, you only need to specify the URL address of the sever **document root**, appoint the project root folder, and specify the URL address to access it.

Creating a server configuration: specifying its name, type, and visibility

1. Open the [Deployment](#) page by doing one of the following:
 - Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Deployment under Build, Execution, Deployment.
 - Choose Tools | Deployment | Configuration on the main menu.
2. In the left-hand pane, that shows a list of all the existing server configurations, click the Add toolbar button `+`. The [Add Server dialog box](#) opens.
3. Specify the server configuration name in the Name text box. From the Type drop-down list, choose the server configuration type In-place. Use the Up and Down keyboard keys to scroll through the list of server configuration types.
4. Use the Visible only for this project check box to configure the visibility of the server access configuration (deployment configuration).
 - Select the check box to restrict the use of the configuration to the current project. Such configurations cannot be reused outside the current project, they do not appear in the list of available configurations in other projects.
 - When the check box is cleared, the configuration is visible in all PyCharm projects and the settings from, including SSH credentials, can be reused.
5. Click OK. The Add Server dialog box closes and you return to the [Connection](#) tab of the [Deployment](#) node.

Configuring access to an in-place server: specifying the URL address of the server document root

In the Web server root URL text box, type the URL address associated with the **document root** of your Web server as defined in the Web server configuration file. This URL address will be the starting point for building the URL address of your application. Both the **HTTP** and the **HTTPS** protocols are supported.

To access a server through **HTTPS**, you need to acquire a certificate file `<certificate_name>.cert` signed by a recognized **authority** and import this certificate in the [truststore/keystore](#) of the **Oracle JRE (Java Runtime Environment)** on which PyCharm runs. Note that self-signed certificates are rejected as unsafe.

To import a certificate in Oracle JRE:

1. Open the embedded Terminal and type the following command:

```
<jre_home>/bin/keytool.exe -importcert -keystore <path to jre truststore/keystore> -file <full_path_to_<cert_name>.cert>
```

If you are using the **Oracle JRE** bundled with PyCharm, the default path to the truststore/keystore is

```
<%product_installation_folder>/jre/jre/lib/security/jssecacerts or
```

```
<%product_installation_folder>/jre/jre/lib/security/cacerts .
```

Otherwise it is `<jre_home>/jre/lib/security/jssecacerts` or `<jre_home>/jre/lib/security/cacerts` .

2. When asked to enter a password for the truststore/keystore, specify the default one `changeit` .
3. Open the `PyCharm.exe.vmoptions` file in the `<PyCharm_installation_folder>/bin` and add the following line to it:

```
-Djavax.net.ssl.keyStore=<path to keystore>
```

4. Restart PyCharm.

Learn more at [Java6](#) and [Java7](#).

For example, the [Apache httpd](#) server configuration file is `httpd.conf` . The default **document root** is the `htdocs` folder and the default URL address to access the data in it is `http://localhost` . If you have changed the default port `80` , you have to specify the port explicitly: `http://localhost:<port>` .

Specifying the project root folder and the URL address to access it

1. Switch to the [Mappings](#) tab.
2. In the Local path text box, specify the full path to your project root folder. Type the path manually, or click the Browse button `...` and choose the folder in the dialog box, that opens.
3. In the Web path on server text box, type the path to the project root folder relative to the server document root specified in the server configuration file. As you type, PyCharm dynamically builds the URL address through which your project root folder will be accessible and shows it as a link in the Project URL read-

only field. To check that the URL address is constructed correctly and ensures access to the project root, click the link.

Creating a Local Server Configuration

On this page:

- [Basics](#)
- [Creating a server configuration: specifying its name, type, and visibility](#)
- [Specifying the server configuration root and the URL address to access it](#)
- [Example of specifying a server configuration root](#)
- [Mapping project folders with folders on the server and the URL addresses to access them](#)
- [Example of mapping project folders with folders on the server](#)

Basics

A **local server** is a server that is running in a local or a mounted folder and whose **document root** is **NOT** the parent of the project root.

To configure access to the server in this set-up, you need to specify the following:

1. The **server configuration root** folder and the URL address to access it.
2. Correspondence between the **project root folder**, the folder on the server to copy the data from the **project root folder** to, and the URL address to access the copied data on the server. This correspondence is called **mapping**.

Creating a server configuration: specifying its name, type, and visibility

1. Open the [Deployment](#) page by doing one of the following:
 - Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Deployment under Build, Execution, Deployment.
 - Choose Tools | Deployment | Configuration on the main menu.
2. In the left-hand pane, that shows a list of all the existing server configurations, click the Add toolbar button **+**. The [Add Server dialog box](#) opens.
3. Specify the server configuration name in the Name text box. From the Type drop-down list, choose the server configuration type Local or mounted folder. When editing the server configuration name in the Name text box, use the Up and Down keys on your keyboard to change the preselected server access to type in the Type drop-down list.
4. Use the Visible only for this project check box to configure the visibility of the server access configuration (deployment configuration).
 - Select the check box to restrict the use of the configuration to the current project. Such configurations cannot be reused outside the current project, they do not appear in the list of available configurations in other projects.
 - When the check box is cleared, the configuration is visible in all PyCharm projects and the settings from, including SSH credentials, can be reused.
5. Click OK. The Add Server dialog box closes and you return to the [Connection](#) tab of the [Deployment](#) node. Click the Use as Default toolbar button  to have PyCharm silently apply the current configuration in the following cases:
 - [Automatic upload of changed files](#).
 - [Manual upload of files](#) without choosing the target host.
 - [Comparing local files and folders](#) with their remote versions.

Specifying the server configuration root and the URL address to access it

1. In the Folder text box of the Upload/download project files area, specify the **server configuration root**. The **server configuration root** is the highest folder in the file tree on the server that can be accessed through the server configuration. The easiest way is to use the **document root** of your Web server as defined in the Web server configuration file. However you can appoint any other existing folder under the **document root**.
2. In the Web server root URL text box of the Browse files on server area, specify the URL address of the **server configuration root**. This URL address will be the starting point for building the URL address of your application. Depending on your choice of the **server configuration root**, do one of the following:
 - Type the URL address associated with the **document root** of your Web server as defined in the Web server configuration file.
 - Type the URL address in the format `<URL of the server document root>/<path to the relevant folder relative to the server document root>`.

Both the **HTTP** and the **HTTPS** protocols are supported.

To access a server through **HTTPS**, you need to acquire a certificate file `<certificate_name>.cert` signed by a recognized **authority** and import this certificate in the [truststore/keystore](#) of the **Oracle JRE (Java Runtime Environment)** on which PyCharm runs. Note that self-signed certificates are rejected as unsafe.

To import a certificate in Oracle JRE:

1. Open the embedded Terminal and type the following command:

```
<jre_home>/bin/keytool.exe -importcert -keystore <path to jre truststore/keystore> -file <full_path_to_<cert_name>.cert
```

If you are using the **Oracle JRE** bundled with PyCharm, the default path to the truststore/keystore is

```
<%product_installation_folder>/jre/jre/lib/security/jssecacerts or  
<%product_installation_folder>/jre/jre/lib/security/cacerts .
```

Otherwise it is `<jre_home>/jre/lib/security/jssecacerts` or `<jre_home>/jre/lib/security/cacerts`.

2. When asked to enter a password for the truststore/keystore, specify the default one `changeit`.

3. Open the `PyCharm.exe.vmoptions` file in the `<PyCharm_installation_folder>/bin` and add the following line to it:

```
-Djavax.net.ssl.keyStore=<path to keystore>
```

4. Restart PyCharm.

Learn more at [Java6](#) and [Java7](#).

Example of specifying a server configuration root

For example, the [Apache httpd](#) server configuration file is `httpd.conf`, according to it, the default **document root** is the `htdocs` folder, and the default URL address to access the data in it is `http://localhost`. For the sake of simplicity, let's suppose that you are using the [XAMPP](#) package and it is installed in the root of the `C:/` drive.

So if you decide to copy your project files directly under the **server document root**, your **server configuration root** will be `C:\xampp\htdocs` and its URL will be `http://localhost:<port>`.

You can establish a more complicated folder structure on the server, for example, to have `MySite1` and `MySite2` folders under the **server document root**. In this case the you will have to decide which of these folders you will use in the current configuration, let it be `MySite2`. Accordingly, the **server configuration root** will be `C:\xampp\htdocs\MySite2` and its URL address will be `http://localhost:<port>\MySite2`.

Mapping project folders with folders on the server and the URL addresses to access them

Configure **mappings**, that is, set correspondence between the project folders, the folders on the server to copy project files to, and the URL addresses to access the copied data on the server. The easiest way is to map the entire project root folder to a folder on the server, whereupon the project folder structure will be repeated on the server, provided that you have selected the Create Empty directories check box in the [Options dialog box](#). "For more details, see [Customizing Upload/Download](#)."

1. Switch to the Mappings tab.
2. In the Local Path text box, specify the full path to the desired folder in the project tree. In the simplest case it is the project root.
3. In the Deployment Path text box, specify the folder on the server where PyCharm will upload the data from the folder specified in the Local Path text box.
Type the path to the folder relative to the **server configuration root**. If the folder with the specified name does not exist yet, PyCharm will create it, provided that you have selected the Create Empty directories check box in the [Options dialog box](#). For more details, see [Customizing Upload/Download](#).
4. In the Web Path text box, type the path to the folder on the server relative to the **server configuration root**. Actually, type the relative path you typed in the Deployment Path text box.

Example of mapping project folders with folders on the server

For example, if your project is `C:\My_Projects\Mapping_project`, the **server document root** is `C:\xampp\htdocs`, the **server configuration root** is `C:\xampp\htdocs\MySite2`, and its URL address is `http://localhost:<port>\MySite2`, fill in the fields as follows:

1. In the Local Path text box, type `C:\My_Projects\Mapping_project`.
2. In the Deployment Path text box, type `Mapping_project`.
3. In the Web Path text box, type `Mapping_project`.

Creating a Remote Server Configuration

On this page:

- [Basics](#)
- [Creating a server configuration: specifying its name, type, and visibility](#)
- [Specifying user credentials defined during registration on the host](#)
- [Enabling connection to the server and specifying the server configuration root](#)
- [Mapping local folders to folders on the server and the URL addresses to access them](#)
- [Overloading the deployment destination by configuring nested mappings](#)

Basics

A **remote server** is a server on another computer (remote host).

To configure access to the server in this set-up, you need to specify the following:

1. Connection settings: server host, port, and user credentials.
2. The **server configuration root** folder and the URL address to access it.
3. Correspondence between the **project root folder**, the folder on the server to copy the data from the **project root folder** to, and the URL address to access the copied data on the server. This correspondence is called **mapping**.

Creating a server configuration: specifying its name, type, and visibility

1. Open the [Deployment](#) page by doing one of the following:
 - Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Deployment under Build, Execution, Deployment.
 - Choose Tools | Deployment | Configuration on the main menu.
2. In the left-hand pane, that shows a list of all the existing server configurations, click the Add toolbar button **+**. The [Add Server dialog box](#) opens.
3. Specify the server configuration name in the Name text box. From the Type drop-down list, choose the server configuration type depending on the protocol you are going to use to exchange the data with the server.
 - **FTP**: choose this option to have PyCharm access the server via the [FTP file transfer protocol](#).
 - **SFTP**: choose this option to have PyCharm access the server via the [SFTP](#) file transfer protocol.
 - **FTPS**: choose this option to have PyCharm access the server via the FTP file transfer protocol over SSL (the [FTPS](#) extension).

When editing the server configuration name in the Name text box, use the Up and Down keys on your keyboard to change the preselected server access to type in the Type drop-down list.

4. Use the Visible only for this project check box to configure the visibility of the server access configuration (deployment configuration).
 - Select the check box to restrict the use of the configuration to the current project. Such configurations cannot be reused outside the current project, they do not appear in the list of available configurations in other projects. For example, if this check box is selected in an SFTP configuration, you cannot use your SSH credentials from it when you configure a remote interpreter.
 - When the check box is cleared, the configuration is visible in all PyCharm projects and the settings from, including SSH credentials, can be reused.

See [Configuring Node.js Interpreters](#) and [Configuring Remote Python Interpreters](#) for details.

5. Click OK. The Add Server dialog box closes and you return to the Connection tab of the Deployment dialog box.
Click the Use as Default toolbar button  to have PyCharm silently apply the current configuration in the following cases:
 - [Automatic upload of changed files](#).
 - [Manual upload of files](#) without choosing the target host.
 - [Comparing local files and folders](#) with their remote versions.

Specifying user credentials defined during registration on the host

1. Specify the registration mode:
 - To login in a regular mode, specify the login in the User name text box.
 - To enable [anonymous access](#) to the server with your email address as password, select the Login as anonymous check box.
2. Specify the way to authenticate to the server:
 - For **FTP server**, type your password and select the Save password check box to have PyCharm remember it.
 - For **SFTP server**, choose the way to authenticate to the server. Do one of the following:
 - To use standard authentication, choose Password, specify the password, and select the Save password check box to have PyCharm remember it.
 - To use [SSH authentication](#) via a key pair, choose Key pair (OpenSSH).
To apply this authentication method, you need to have your private key on the client machine and your public key on the remote server you connect to. PyCharm supports private keys generated using the [OpenSSH](#) utility. See http://wiki.qnap.com/wiki/How_To_Set_Up_Authorized_Keys for details.
Type the private key file and the passphrase in the corresponding text boxes.
 - For **FTPS server**, specify your user name and password and choose the security mechanism to apply.
 - Choose Explicit to have the [explicit \(active\) security](#) applied. Immediately after establishing connection, the FTP client on your machine sends a command to the server to establish secure control connection through the default FTP port.
 - Choose Implicit to have the [implicit \(passive\) security](#) applied. In this case, security is provided automatically upon establishing connection to the server which appoints a separate port for secure connections.

Enabling connection to the server and specifying the server configuration root

- Specify the host name of the FTP/SFTP/FTPS server to exchange data with and the port to which this server listens. The default values are:
 - 21 for FTP and FTPS
 - 22 for SFTP
- In the Root path text box, specify the **server configuration root** relative to your **user home** which was defined when you registered your account. This folder will be the highest one in the folder structure accessible through the current server configuration. Do one of the following:
 - Accept the default value `/`, which points at the **user home** folder on the server.
 - Type the path manually.
 - Click the Browse button  and select the desired folder in the Choose Root Path dialog box that opens.
 - Click the Autodetect button and have PyCharm detect the user home folder settings on the FTP/SFTP server and set up the root path according to them. The button is only enabled when you have specified your user name and password.
- In the Web server root URL text box, type the URL address to access the server configuration root (The **server configuration root** is the highest folder in the file tree on the **local** or **remote** server accessible through the server configuration. For **in-place** servers, it is the project root.). Both the **HTTP** and the **HTTPS** protocols are supported.

To access a server through **HTTPS**, you need to acquire a certificate file `<certificate_name>.cert` signed by a recognized **authority** and import this certificate in the [truststore/keystore](#) of the **Oracle JRE (Java Runtime Environment)** on which PyCharm runs. Note that self-signed certificates are rejected as unsafe.

To import a certificate in Oracle JRE:

- Open the embedded Terminal and type the following command:

```
<jre_home>/bin/keytool.exe -importcert -keystore <path to jre truststore/keystore> -file <full_path_to_<cert_name>.cert
```

If you are using the **Oracle JRE** bundled with PyCharm, the default path to the truststore/keystore is

```
<%product_installation_folder>/jre/jre/lib/security/jssecacerts or  
<%product_installation_folder>/jre/jre/lib/security/cacerts .
```

Otherwise it is `<jre_home>/jre/lib/security/jssecacerts` or `<jre_home>/jre/lib/security/cacerts` .

- When asked to enter a password for the truststore/keystore, specify the default one `changeit` .
- Open the `PyCharm.exe.vmoptions` file in the `<PyCharm_installation_folder>/bin` and add the following line to it:

```
-Djavax.net.ssl.keyStore=<path to keystore>
```

- Restart PyCharm.

Learn more at [Java6](#) and [Java7](#).

- Click the Open button to make sure that the specified URL address is accessible and points at the correct Web page.

Mapping local folders to folders on the server and the URL addresses to access them

Configure **mappings**, that is, set correspondence between the project folders, the folders on the server to copy project files to, and the URL addresses to access the copied data on the server . The easiest way is to map the entire project root folder to a folder on the server, whereupon the project folder structure will be repeated on the server, provided that you have selected the Create Empty directories check box in the [Options dialog box](#). "For more details, see [Customizing Upload/Download](#).

- Switch to the Mappings tab.
- In the Local Path text box, specify the full path to the desired folder in the project tree. In the simplest case it is the project root.
- In the Deployment Path text box, specify the folder on the server where PyCharm will upload the data from the folder specified in the Local Path text box. Type the path to the folder relative to the **server configuration root**. If the folder with the specified name does not exist yet, PyCharm will create it, provided that you have selected the Create Empty directories check box in the [Options dialog box](#). For more details, see [Customizing Upload/Download](#).
- In the Web Path text box, type the path to the folder on the server relative to the **server configuration root**. Actually, type the relative path you typed in the Deployment Path text box.

Overloading the deployment destination by configuring nested mappings

You can configure separate mappings for a specific folder under the your project root to have the contents of this folder synchronized with another location on the remote host.

Suppose you have configured the mappings as follows:

Local Path Deployment Path

<code><project_root></code>	<code>ftp://.../htdocs/my_project</code>
<code><project_root>/my_folder</code>	<code>ftp://.../htdocs/my_folder</code>

Then the files in your project will be uploaded as follows:

Local Path Deployment Path

<code><project_root>/file1.js</code>	<code>ftp://.../htdocs/my_project/file1.js</code>
<code><project_root>/my_folder/file2.js</code>	<code>ftp://.../htdocs/my_folder/file2.js</code>

instead of `ftp://.../htdocs/my_project/my_folder/file2.js`

On this page:

- [Basics](#)
- [Setting common upload/download options](#)
- [Specifying additional protocol-specific customization options for FTP/SFTP/FTPS servers](#)

Basics

In addition to the mandatory settings that ensure successful upload and download in various project - server set-ups, you can choose additional options to customize interaction with the server. Most of these options apply to all types of **server access configuration**. For FTP/FTPS/SFTP server configurations, you can specify additional protocol-specific options.

Setting common upload/download options

1. Open the [Options](#) dialog box by doing one of the following:
 - On the main menu, choose Tools | Deployment | Options.
 - Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS. Expand the Build, Execution, Deployment node, and then click Options under Deployment.
2. In the Options dialog box that opens, specify additional settings:
 - To have PyCharm skip specific files or entire folders during upload/download, in the Exclude items by name text box, specify the patterns that define the names of these files and folders. Use semicolons as delimiters. Wildcards are welcome.
The exclusion is applied recursively. This means that if a matching folder has subfolders, the contents of these subfolders are not deployed either.

For more details about excluding files and folders from upload/download, see [Excluding Files and Folders from Upload/Download](#).

 - Specify the details of the upload/download procedure by selecting or clearing the corresponding check boxes.

Specifying additional protocol-specific customization options for FTP/SFTP/FTPS servers

1. Open the [Deployment](#) dialog box by doing one of the following:
 - Choose Tools | Deployment | Configuration on the main menu.
 - Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Deployment under Build, Execution, Deployment.
2. In the [Deployment](#) dialog box, click the Advanced Options button and specify additional uploading settings in the Advanced Options dialog box that opens:
 - To set the client to the [passive mode](#), select the Passive mode check box. In this mode, the client on your machine connects to the server to inform about being in the passive mode, receives the port number to listen to, and established data connection through the port with the received number. This mode is helpful when your machine is behind a firewall.
 - To have the hidden files and directories (those with names that start with a dot `.`) shown in the [Server Browser Tool Window](#), select the Show Hidden Files check box.
 - Select the Compatibility mode check box to ensure [compatibility in child file naming](#) with your FTP server.
This option is helpful if the remote FTP server reports the following error:

```
Invalid descendant file name <file name>
```

Selecting this option may slow down synchronization with the server.

- Use the Retrieve accurate files timestamps drop-down list to specify the [MDTM](#) FTP command calling policy to retrieve the last-modified time of a given file on the remote host. The available options are:
 - Always - select this option to have MDTM called for every file shown in the [Remote Host](#) tool window.
 - On copy - select this option to have MDTM called in the following cases:
 - To check whether a file is up to date when the Overwrite up-to-date files check box in the [Options](#) dialog box is cleared.
 - To preserve the actual time stamp of a file during download.
 - Never - select this option to suppress calling MDTM.
- Select the Limit concurrent connections check box to have PyCharm restrict the number of connections to be supported simultaneously and specify the maximum number of allowed connections in the text box.
- In the Control encoding text box, specify the encoding that matches the encoding used by your server. Accept the default value if you are not sure that it supports [UTF-8](#) encoding.
- Select the Always use LIST command check box to use the standard `LIST` command for listing instead of the `MLSD` command. This lets you avoid problems, for example, failure during upload with the **Invalid descendant file name** exception if the FTP server supports `MLSD` and returns `cdlr`.
- In the Send keep alive message each text box, specify how often you want PyCharm to send commands to the server to reset the timeout and thus preserve the connection.
- From the Use keep alive command drop-down list, choose the commands to be sent to the server to reset the timeout and thus preserve the connection.
- On some [SFTP](#) servers, the [SSH banner](#) may be enabled. Every time a connection is established, a pop-up window with an information message may be shown and to continue you would need to click OK.
To suppress showing the information pop-up window, select the Ignore info messages check box.

Excluding Files and Folders from Upload/Download

On this page:

- [Basics](#)
- [Excluding a folder on server from upload/download after project creation](#)
- [Excluding a local folder from upload/download](#)
- [Excluding files and folders from upload/download by name](#)
- [Removing the exclusion mark](#)

Basics

Suppressing uploading, downloading, and synchronization for files or folders with sources ensures that the sources are protected against accidental update. When applied to non-sources, it saves system resources because huge amounts of media, caches, or temporal files are no longer copied hither and thither.

You may need to suppress upload/download in the following cases:

1. You are going to work with externally created and uploaded source code. To process these remote sources in PyCharm, you have to download them and arrange them in a project. However, there are some sources that you should not update at all. On the other hand, the folders on the remote host also may contain huge amounts of media, caches, temporal files, that you actually do not need in your work.
2. You have already downloaded the data from the server and arranged them in a PyCharm project. However, for this or that reason, you need to have some files or folders on the server protected against upload/download, for example, to prevent accidental overwriting.
3. The local copy of an application contains both source code and other data that you do not need to upload. Besides, you want to protect some sources against overwriting by mistake. In this case, you can suppress upload/download for all files and folders that should not be uploaded.

There are two ways to **exclude folders** from upload/download:

- Explicitly, by marking the corresponding paths as excluded in the [Remote Host](#) tool window or in the [Excluded Paths](#) tab of the [Deployment dialog box](#).
 1. The names of all the not-excluded folders and files are displayed on green background. The names of excluded items are displayed without background.
 2. In the [Remote Host](#) tool window, you can exclude both entire folders and specific files.
- **By name**, that is, by specifying [patterns that determine the names](#) of files and folders to be excluded in the Exclude Items by Name text box of the [Options](#) dialog box.

Separate files can be protected against upload/download only through [excluding them by name](#).

Note In either case, the exclusion is applied recursively. This means that if the path to a folder is explicitly marked as excluded or the folder name matches the pattern, the contents of all its subfolders, if any, are also protected against upload/download.

Excluding a folder on server from upload/download after project creation

1. Choose the required folder right in the [Remote Host](#) tool window:
 1. On the main menu, choose Tools | Deployment | Browse Remote Host or View | Tool Windows | Remote Host .
 2. In the [Remote Host](#) tool window that opens, select the relevant [server configuration](#) from the drop-down box.
 3. Select the folder to exclude and choose Exclude Path on the context menu of the selection.
2. Add the required folder to the list of excluded paths:
 1. Open the [Deployment](#) dialog box by doing one of the following:
 - Choose Tools | Deployment | Configuration on the main menu.
 - Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Deployment under Build, Execution, Deployment.
 2. In the [Deployment](#) dialog box, switch to the [Excluded Paths](#) tab. The tab shows the list of the previously excluded local and remote folders.
 3. Click the Add deployment path button. An empty line is added to the list.
 4. Click the Browse button . The Select remote excluded path dialog box that opens shows the data on the host accessed through the selected server configuration. Select the required folder.
 5. When you OK, you return to the [Excluded Paths](#) tab, where the selected remote folder is added to the list.

Excluding a local folder from upload/download

1. Open the [Deployment](#) dialog box by doing one of the following:
 - Choose Tools | Deployment | Configuration on the main menu.
 - Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Deployment under Build, Execution, Deployment.
2. In the [Deployment](#) dialog box, switch to the [Excluded Paths](#) tab. The tab shows the list of the previously excluded local and remote folders.
3. Click the Add local path button. In the empty line that is added to the list, specify the location of the folder to be protected against upload/download. Type the path manually or click the Browse button  and choose the required folder in the [dialog that opens](#).

Excluding files and folders from upload/download by name

1. Open the [Options](#) dialog box by doing one of the following:
 - On the main menu, choose Tools | Deployment | Options.
 - Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS. Expand the Build, Execution, Deployment node, and then click Options under Deployment.
2. In the [Options](#) dialog box that opens, specify the patterns that define the names of these files and folders in the Exclude items by name text box. Use semicolons as delimiters. Wildcards are welcome.

The exclusion is applied recursively. This means that if a matching folder has subfolders, the contents of these subfolders are not deployed either.

Removing the exclusion mark

To return a file or folder to upload/download, select it and choose **Remove Path from Excluded** on the context menu of the selection. Returning a folder to upload/download automatically affects all its subfolders and files.

Uploading and Downloading Files

On this page:

- [Uploading files and folders](#)
 - [To upload a file or folder manually:](#)
 - [To upload a file or folder to the default server manually:](#)
 - [To upload checked-in files immediately after commit:](#)
 - [To configure automatic upload of changed files to the default server:](#)
- [Downloading files and folders](#)

Uploading files and folders

PyCharm provides the following main ways to upload project files and folders to a deployment server:

- **Manually**, at any time through a menu command.
- **Automatically**, every time a file is updated, or before starting a debugging session, or during commit to your version control system.

To upload a file or folder manually:

In the Project tool window, select this file or folder, choose Upload to on the context menu, and then select the target deployment server from the list.

To upload a file or folder to the default server manually:

In the Project tool window, select this file or folder and then choose Upload to <default deployment server> on the context menu of the selection.

Tip If the area is folded, click  to expand it.

To upload checked-in files immediately after commit:

1. [Start checking in your changes.](#)
2. In the After Commit area, choose the target server from the Upload file to drop-down list. Choose one of the existing server access configurations or create a new one: click the Browse button  and [configure access to the relevant server](#) in the [Deployment](#) dialog box that opens.
3. To have PyCharm automatically upload checked in files to the chosen server, select the Always use selected server check box.

Tip PyCharm considers a local file changed as soon as it is saved either automatically or manually (File | Save All or `Ctrl+S`), see [Saving and Reverting Changes](#). Changed files can be automatically uploaded only to the [default deployment server](#).

To configure automatic upload of changed files to the default server:

1. Open the [Options](#) dialog box: Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS. Expand the Build, Execution, Deployment node, and then click Options under Deployment.
2. In the Upload changed files automatically to the default server drop-down list, specify when you want PyCharm to upload changed files:
 - To have PyCharm upload any saved file no matter whether the save was invoked manually or automatically, choose Always.
 - To have PyCharm upload only files that were saved manually, choose On explicit save action.
 - To suppress automatic upload, choose Never.

Downloading files and folders

To download a file or folder, select it in the Remote Hosts tool window and choose Download from here on the context menu of the selection.

To download a file from the default deployment server, choose Tools | Deployment | Download from <default server configuration> on the main menu.

Accessing Files on Web Servers

On this page:

- [Basics](#)
- [Accessing a server](#)
- [Handling files and folders on the server](#)

Basics

Once you have [set up synchronization](#) between your local application sources and the application sources on a server, you can create new folders, move, rename, and delete existing files and folders. You can also [compare](#) files and folders on the server with their local versions.

For the sake of simplicity, any file or folder in your PyCharm project is called **local** and any file or folder on the server is called **remote**, even if the server is actually installed on your machine. For details, see [Configuring Synchronization with a Web Server](#).

Although PyCharm supports direct editing of individual non-project files on servers, to keep your local and remote sources synchronized, configure automatic upload using the >Upload changed files automatically to the default server drop-down list in the [Options](#) dialog box.

Access to servers is controlled through [server configurations](#) of the type FTP, FTPS, SFTP, or Local or Mounted Folder.

Accessing a server

1. Open the [Remote Host tool window](#) by choosing Tools | Deployment | Browse Remote Host or View | Tool Windows | Remote Host on the main menu.
2. Select the required deployment server from the drop-down list. The tool window shows a tree view of files and folders under the **server root**. If no relevant server is available in the list, click , and in the Deployment dialog box that opens [configure access to the required server](#).

Handling files and folders on the server

From the Remote Host tool window, you can create, move, rename, and delete files and folders on the server.

To create a folder:

Select the parent directory and choose Create Folder on the context menu.

To remove a file or folder:

Select the required file or folder and choose Delete on the context menu.

To rename a file or folder:

Select the required file or folder in the tree view, choose Rename on the context menu, and specify the new name in the Rename dialog box that opens.

 Alternatively, select the required file or folder and drag it to the new location.

To move a file or folder:

1. Select the file or folder to move and choose Cut on the context menu.
2. Select the new parent folder and choose Paste on the context menu. Then confirm the changes in the Move remote items dialog box that opens.

Comparing Deployed Files and Folders with Their Local Versions

On this page:

- [Basics](#)
- [Accessing a server](#)
- [Comparing files and folders on the server with their local versions](#)
- [Comparing local files and folders with their versions on the server](#)
- [Comparing and synchronizing two folders in the Difference Viewer](#)

Basics

Correspondence between files and folders in your PyCharm project and their versions on a server is set through [deployment server mappings](#). For the sake of simplicity, any file or folder in your PyCharm project is called **local** and any file or folder on the server is called **remote**, even if the server is actually installed on your machine. For details, see [Configuring Synchronization with a Web Server](#).

Accessing a server

1. Open the [Remote Host tool window](#) by choosing Tools | Deployment | Browse Remote Host or View | Tool Windows | Remote Host on the main menu.
2. Select the required deployment server from the drop-down list. The tool window shows a tree view of files and folders under the **server root**. If no relevant server is available in the list, click , and in the Deployment dialog box that opens [configure access to the required server](#).

Comparing files and folders on the server with their local versions

Each remote file or folder is [mapped](#) to one and only one local file or folder. Therefore for each remote file or folder, PyCharm unambiguously detects its local version so you can compare them at any time.

To compare a remote file with its local version:

1. Open the [Remote Host tool window](#) (Tools | Deployment | Browse Remote Host or View | Tool Windows | Remote Host) and select the required deployment server from the drop-down list.
2. Select the file in question, and then choose Compare with local version on the context menu of the selection.
3. In the [Differences Viewer for Files](#) dialog box, that opens, explore the differences and apply them, if necessary, using the » and « buttons. For details, see [Viewing Differences Between Files](#).

To compare a remote folder with its local version:

1. Open the [Remote Host tool window](#) (Tools | Deployment | Browse Remote Host or View | Tool Windows | Remote Host) and select the required deployment server from the drop-down list.
2. Select the folder in question and choose Sync with local on the context menu of the selection.
3. In the [Differences Viewer for Folders](#) that opens, explore the differences and synchronize the files, where applicable, as described in [Comparing two folders in the Difference Viewer](#).

Comparing local files and folders with their versions on the server

Because local file or folder can be mapped to an unlimited number of remote counterparts, PyCharm can uniquely identify remote versions of local files or folders only when they are mapped through the [default deployment server](#). If no such default deployment server is appointed, you have to choose the relevant configuration manually.

To compare a local file with its remote version:

1. Select the file in question in the [Project](#) tool window.
2. On the context menu of the selection, choose Deployment | Compare with Deployed Version on <default server access configuration>.
3. In the [Differences Viewer for Files](#) dialog box, that opens, explore the differences and apply them, if necessary, using the » and « buttons. For details, see [Viewing Differences Between Files](#).

To compare a local folder with its remote version:

1. Select the folder in question in the [Project](#) tool window.
2. On the context menu of the selection, choose Sync with Deployed to <default deployment server> if a default server is appointed. Otherwise, choose Sync with Deployed to and then choose the relevant server from the list.
3. In the [Differences Viewer for Folders](#) that opens, explore the differences and synchronize the files, where applicable, as described in [Comparing two folders in the Difference Viewer](#).

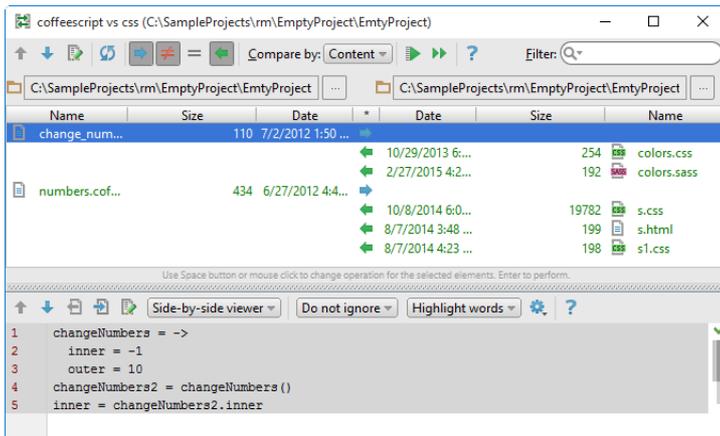
Comparing and synchronizing two folders in the Difference Viewer

PyCharm provides a dedicated [Differences Viewer for Folders](#) for comparing files in remote folders and their local versions against the file size, content, or timestamp. Besides exploring differences, the tool also provides interface for synchronizing the contents of folders.

- The Item List shows the contents of the local and remote folders. Use the [toolbar buttons](#) to narrow down or widen the set of items to show. For example, show or hide files that are present only locally or remotely, equal files, different files, files [excluded from synchronization](#), etc.
- The contents of the remote folder are always shown in the right-hand pane and the contents of its local version are always shown in the left-hand pane.
- The contents of the selected file are shown in the lower pane, with the differences being color-highlighted.
- The remote files in the Difference Viewer have the status `read-only`. This means that you cannot update them directly in the Difference Viewer. Make all the necessary changes to the local version of the file in question and upload the updated file to the server.

To compare two folders:

1. Specify the parameter for comparison. In the Compare by drop-down list, select one of the possible options (contents, size, or time stamp).
2. Filter the folders' contents. To do that, type filtering string in the Filter text field, and press to apply it. Using the asterisk * wildcard to represent any number of characters is welcome.
3. To switch to another pair of folders to compare, update the fully qualified paths to them. Click the Browse button next to the Paths read-only fields and choose the required folders in the [dialog box that opens](#).
4. Explore the detected differences between files in the Differences Pane.



To synchronize the contents of two folders:

1. For each pair of items, in the * field specifies the action to apply. Click the icon in the field until the required action is set.
 - the file will be uploaded, possibly overwriting the remote version.
 - the file will be downloaded, possibly overwriting the local version.
 - the files are treated identical with regard to the selected criterion of comparison. No action will be performed by default.
 - the files differ with regard to the selected criterion of comparison. No action will be performed by default. Explore the differences in the Differences Pane of the Difference Viewer and change the intended action by clicking the icon.

The remote files in the Difference Viewer have the status `read-only`. This means that you cannot update them directly in the Difference Viewer. Make all the necessary changes to the local version of the file in question and upload the updated file to the server.
 - the file is present only locally or remotely and will be removed.
2. Do one of the following:
 - To synchronize the currently selected item, click the Synchronize Selected button on the toolbar.
 - To synchronize all the items, click the Synchronize All button on the toolbar.

Editing Individual Files on Remote Hosts

On this page:

- [Introduction](#)
- [Editing files on remote hosts](#)

Introduction

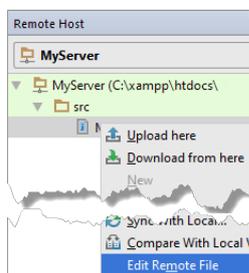
Once you've set up synchronization with a remote host, you can open individual files directly from the remote host and edit them in PyCharm, without adding/downloading them to the local project. Note that for such files not included in a project, some PyCharm features are not supported.

To take advantage of debugging, refactorings, and other advanced features, consider including the files into a project; see [Accessing Files on Web Servers](#) for details.

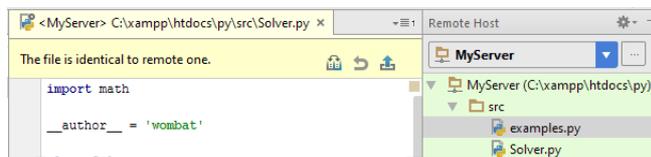
Editing files on remote hosts

To edit a file on a remote host

1. Open the [Remote Host tool window](#) by choosing Tools | Deployment | Browse Remote Host or View | Tool Windows | Remote Host on the main menu.
2. Select the required deployment server from the drop-down list. The tool window shows a tree view of files and folders under the **server root**. If no relevant server is available in the list, click , and in the Deployment dialog box that opens [configure access to the required server](#).
3. To start editing a file, double click its name or select the file name in the [Remote Host Tool Window](#), and choose Edit Remote File on the context menu:

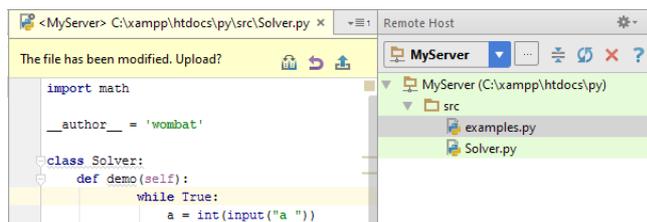


The file opens in the PyCharm editor, without being added or downloaded to the local project.



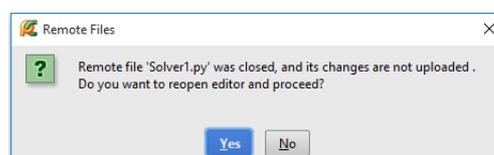
When you work with a remote file, a special toolbar appears at the top of the editor, showing the editing status ("The file is identical to the remote one" or "The file has been changed. Upload?").

Remote files can be easily distinguished from local ones by looking at the annotation, which includes the server name (in our case it is <MyServer>).



4. When you are done editing the file, do one of the following:
 - To upload the file to the remote host, click .
 - To compare the currently opened file with the last uploaded version, click . The files are opened in the standard PyCharm [Difference Viewer](#), see [Differences Viewer for Files](#).
 - To discard the changes introduced to the file after the last upload, click .

As it has been mentioned above, an individual file is not added to a project. As a result, all the changes to it are discarded as soon as you close the file or the currently opened project unless these changes have been uploaded. To prevent losing your data accidentally, PyCharm displays the following dialog box when you attempt to close the edited file or the entire project:



Running SSH Terminal

On this page:

- [Introduction](#)
- [Preparing to work in the SSH Terminal](#)
- [Launching an SSH Terminal](#)

Introduction

You can launch an **SSH Session** right from PyCharm. By running commands in a dedicated SSH terminal, you can access data on a remote Web server or the default remote interpreter via an SSH tunnel, mainly upload and download files.

Preparing to work in the SSH Terminal

1. Make sure the **SSH Remote Run** plugin is enabled. The plugin is bundled with PyCharm and activated by default. If the plugin is not activated, enable it on the [Plugins](#) page of the [Settings / Preferences Dialog](#) as described in [Enabling and Disabling Plugins](#).
2. Make sure, an SSH server is available in the **destination environment**: a remote Web server or the default remote interpreter.
3. Register an account on the SSH server in the **destination environment** and generate a pair of SSH keys or a password, depending on the server policy.
4. Appoint the **destination environment** and specify the settings to establish connection with it:
 1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click SSH Terminal under Tools. The [SSH Terminal](#) page opens.
 2. In the Connection settings area, appoint the **destination environment**:
 - Default remote interpreter: select this option to have commands in the SSH terminal executed on the same host, where the [default remote interpreter](#) runs.
 - Current Vagrant: select this option to have the commands in the SSH Terminal executed on the currently running Vagrant virtual machine. For details, see [Vagrant: Working with Reproducible Development Environments](#).
 - Deployment server: select this option to have the commands in the SSH Terminal executed on the local or remote Web server accessible through one of the [server access configurations](#). From the drop-down list, choose the server access configuration that specifies the destination environment and the settings to establish connection to it.
 - Select server on every run: if this option is selected, you will have to choose the desired server access configuration from the pop-up window, every time you choose Tools | Start SSH Session on the main menu.
 - If the desired server access configuration does not appear in the drop-down list, click the link Configure Servers, and define one in the [Deployment](#) page. For details, see [Configuring Synchronization with a Web Server](#).
 3. From the Default encoding drop-down list, select the desired encoding to be used in the SSH terminal.

Launching an SSH Terminal

1. On the main menu, choose Tools | Start SSH Session.
2. Depending on the connection settings, defined in the [SSH Terminal](#) page of the Settings/Preferences dialog, the following types of behavior are possible:
 - If the Default remote interpreter option has been selected, the SSH terminal will provide access to the same host, where the [default remote interpreter](#) runs.
 - If the Current Vagrant option has been selected, the SSH Terminal will provide access to the currently running Vagrant virtual machine. For details, see [Vagrant: Working with Reproducible Development Environments](#).
 - If the option Deployment server has been selected, the SSH Terminal will provide control over the data on the local or remote Web server accessible through the [server access configuration](#) selected from the list. For details, see [Configuring Synchronization with a Web Server](#).
 - If the option Select server on every run has been selected, PyCharm will show a pop-up list to choose the desired [server access configuration](#) from.

Code Coverage

This feature is supported in the Professional edition only.

In this section:

- Code Coverage
 - [Basics](#)
 - [Running with code coverage](#)
 - [To use code coverage in project, follow these general steps](#)
- [Configuring Code Coverage Measurement](#)
- [Running with Coverage](#)
- [Viewing Code Coverage Results](#)
- [Managing Code Coverage Suites](#)
- [Generating Code Coverage Report](#)

Basics

Code coverage in PyCharm allows you to perform on-the-fly line coverage measuring for your code with low runtime overhead. In general, line coverage answers the question, "Was this line of code executed during unit testing simulation?"

Code coverage results are reflected in the dedicated [Coverage tool window](#), in the Project view of the [Project tool window](#), and in the editor. The tool windows show the following information:

- For a directory: the percentage of the covered classes and lines.
- For a file: the percentage of the covered lines.

When a file is opened in the editor, each line is highlighted with regard to its code coverage status:

- Lines executed during simulation are marked green.
- Lines not executed during simulation are marked red.

The coverage measurement results comprise a coverage suite. You can have the results of a new simulation merged with any existing suite. In this case, a line will be considered covered if it is covered by at least one of the simulations.

A coverage suite is generated every time a test or application with code coverage measurement is executed. It is possible to have an unlimited amount of coverage suites.

Running with code coverage

To use code coverage in project, follow these general steps

1. Specify how you want to [process the coverage results](#).
2. [Create tests](#) for the target code, if you are going to measure code coverage for testing.
3. [Configure code coverage measurement](#) in the desired run/debug configuration.
4. [Run with coverage](#), using the dedicated command on the main menu Run | Run with Coverage, or .
5. Once the run with coverage has been executed, you can
 - Use the [various coverage suites](#).
 - [View code coverage data](#).
 - [Generate code coverage report](#).

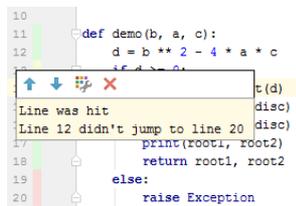
Configuring Code Coverage Measurement

PyCharm makes it possible to configure the various aspects of code coverage measurement. In this section:

- [Configuring the way coverage suites are processed](#)
- [Changing colors of the coverage highlighting](#)

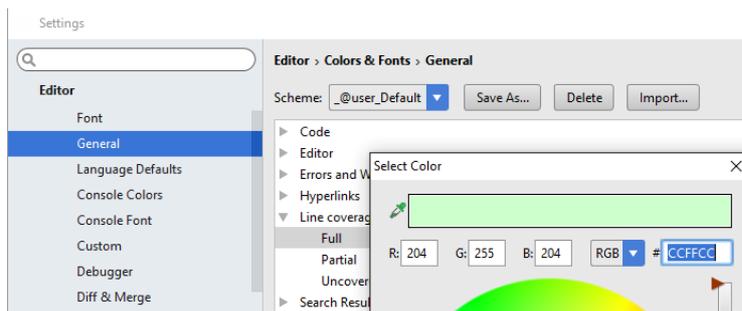
To configure code coverage behavior

1. Open the [Settings/Preferences dialog box](#), and then click Coverage under Build, Execution, Deployment. The [Coverage](#) page opens.
2. Define how the collected coverage data will be processed:
 - To have the Code Coverage dialog box shown every time you launch a new run configuration with code coverage, choose Show options before applying coverage to the editor.
 - To discard the new code coverage results, choose Do not apply collected coverage.
 - To discard the active suites and use the new one every time you launch a new run configuration with code coverage, choose Replace active suites with the new one.
 - To have the new code coverage suite appended to the active suites every time you launch a new run configuration with code coverage, choose Add to active suites.
3. Define the behaviour of the [Coverage](#) tool window when an application or test is run with coverage:
 - To have the Coverage tool window opened automatically, select the Activate Coverage View check box.
 - To open the Coverage tool window manually, clear the Activate Coverage View check box.
4. Select the check box to specify which of the coverage tools you want to use - the one bundled with PyCharm, or included in the active Python interpreter.
5. Select the check box to specify whether you want to use the branch coverage. Thus additional information to the pure line coverage reports is added, marking the coverage of lines with conditional statements as incomplete in case one or more branches haven't been executed:



To configure code coverage colors

1. Open the [Colors and Fonts](#) page of the editor settings. Alternatively, just click  in the statistics pop-up.
2. Expand Colors and Fonts node, and select General.
3. In the list of textual components, select the required type of coverage, for example, Full, Partial or Uncovered, and then choose the desired colors:



Running with Coverage

PyCharm provides a dedicated action that allows you to perform run with code coverage measurement. The code coverage data are processed according to the option selected in the [Coverage](#) page of the Settings/Preferences dialog box.

To run with code coverage measurement

1. Do one of the following:
 - Open the desired file in the editor, and choose Run <name> with coverage on the context menu. When running tests with coverage, note that you can run the entire test class, or each individual test method, depending on the caret location.
 - Select the desired run/debug configuration, and then on the main menu choose Run | Run <run/debug configuration name> with coverage.
 - On the main toolbar, click . This will launch the selected run/debug configuration.
2. If the Show options before applying coverage to the editor check box has been selected in the [Coverage](#) page of the Settings/Preferences dialog box, choose whether you want to replace the active coverage suites, or add the collected data to the active suites, or do not want not apply coverage data. You can also opt to skip this dialog in the future.
In case any other option has been selected, the respective action will be performed silently.
3. [Explore the collected coverage data](#) in the [Coverage Tool Window](#).

Viewing Code Coverage Results

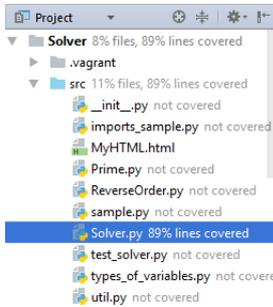
Viewing code coverage helps you detect pieces of your source code that are not affected by simulation.

To view code coverage results

1. Do one of the following:
 - Run the desired class with coverage, select suite to show, and open class in the editor.
 - On the main menu, choose Tools | Show Code Coverage Data.
 - Press `(Ctrl+Alt+F6)`.

2. View coverage results:

- In the Project tool window:

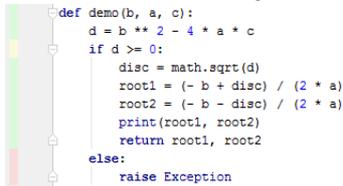


- In the dedicated Coverage tool window:

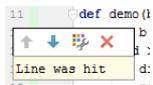
Element	Statistics, %
.idea	
.vagrant	
src	11% files, 89% lines covered
test_dir	0% files, not covered
CIMG6084.JPG	
python_stubs.py	not covered
python_stubs.pyi	not covered
requirements.txt	

3. Open in the editor the files you want to explore.

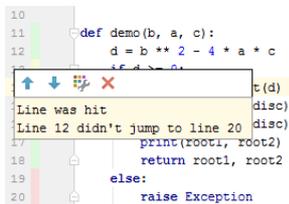
4. Use the color indicators in the left gutter to detect the uncovered lines of code.



5. To find out how many times a line has been hit, click the line in the gutter area.



The pop-up window that opens shows the statistic for the line at caret. For lines with conditions, the pop-up window also provides statistic:



Use the following toolbar buttons:

- : jump to the next/previous groups of covered or uncovered lines.
- : open the Colors and Fonts settings, where you have to choose the node Line Coverage.
- : hide coverage information.

6. Create missing tests, as described in the section [Creating Tests Creating JavaScript Unit Tests](#). For JavaScript, tests can be generated as described in section [Creating JavaScript Unit Tests](#).

Managing Code Coverage Suites

PyCharm provides a tool to select coverage suites for showing or hiding, adding and removing suites.

In this section:

- [Accessing the Choose Coverage Suite to Display dialog box](#)
- [Selecting coverage suites to show](#)
- [Hiding coverage suites](#)
- [Adding coverage suites from disk](#)

To open the Choose Coverage Suite to Display dialog box, do one of the following

- On the main menu, choose Tools | Show Coverage Data.
- Press `Ctrl+Alt+F6`.

To select coverage suites to show

1. Open the Choose Coverage Suite to Display dialog box, as described above.
2. In the Choose Coverage Suite to Display dialog box, select the check boxes next to the desired suites.
3. Click Show selected. The dialog box closes.
4. Open in the editor the classes, for which the coverage suites have been selected, and explore the coverage results.

To hide coverage suites

1. Open the Choose Coverage Suite to Display dialog box, as described above.
2. In the Choose Coverage Suite to Display dialog box, select the check boxes next to the suites you want to hide.
3. Click No coverage. The dialog box closes. The coverage results are not shown for the target classes.

To add or delete a coverage suite

Consider a situation when a file that contains code coverage information, has been obtained from the build server. You can load this file from disk and make it available for review. Also, you can bring for examination the coverage suite that has been produced some time ago.

On the other hand, PyCharm allows you to remove unnecessary coverage suites.

1. Open the Choose Coverage Suite to Display dialog box, as described above.
2. Add new suites or delete the existing ones:
 - Click `+`, and select the desired `*.es` file in the file chooser dialog.
 - Select one or more suites in the list, and click `-`. The selected suite will be deleted from the list, and from storage.

Generating Code Coverage Report

PyCharm suggests two ways of generating HTML reports on the base of the code coverage measurement results: using the menu command, or using the [Coverage tool window](#).

To generate a code coverage report

1. Do one of the following:
 - On the main menu, choose Tools | Generate Coverage Report.
 - In the toolbar of the [Coverage tool window](#), click .
2. In the Generate Coverage Report dialog box that opens, specify the target directory where the generated report will be stored, and optionally select the check box Open generated HTML in browser.
3. Click Save. PyCharm will store the generated report to the specified location, and also open it in the default browser, if the corresponding check box has been selected.

Please note that if you are going to save a code coverage report for one of the [multiple projects opened in the same window](#), it is important to check the suggested target location, because PyCharm suggests the previously used location.

Using Language Injections

You can inject a language (such as HTML, CSS, XML, SQL, RegExp, etc.) into a string literal in your code and, as a result, get comprehensive coding assistance when editing that literal.

- [Prerequisite](#)
- [Example: Injecting HTML. Opening a fragment editor](#)
- [Accessing language injection functions](#)
- [Ways to inject a language](#)
- [Using language injection comments](#)
- [Accessing injection settings](#)
- [Using language injection prefixes and suffixes](#)

Prerequisite

For language injection features to be available, the IntelliLang [plugin](#) must be enabled. (This plugin is bundled with the IDE and enabled by default.)

Example: Injecting HTML. Opening a fragment editor

To get an impression of how language injections work:

1. In the editor, add a code fragment containing a string literal.
2. Place the cursor within the literal (between the quotation marks).
3. Click  or press `Alt+Enter`, select Inject language or reference, and then select HTML (HTML files).
4. Type:

```
<body><h1>Hello, World!</h1></body>
```

When typing, note that auto-completion for HTML tags is now available. Also note how the HTML code is highlighted.

5. Let's now open a fragment editor for the injected HTML code: press `Alt+Enter` and select Edit HTML Fragment. You can use the fragment editor as an alternative (or in addition) to editing injected string literals in the "main editor".
6. To complete the example, let's cancel the injection: switch to the main editor, press `Alt+Enter` and select Un-inject Language/Reference. Note that the text between the quotation marks has become green which is the default color for string values. This indicates that the value in the quotation marks is now treated simply as text.

Don't close the editor yet. Later in this topic, we'll use our code for showing other language injection features.

Accessing language injection functions

Most of the functions related to language injections are accessed through [intention actions](#) ( or `Alt+Enter`).

Ways to inject a language

You can inject a language by using:

- Inject language or reference [intention action](#). You have already seen that in [Example: Injecting HTML. Opening a fragment editor](#). This way of injecting a language is temporary: the string literal stays injected for a limited period of time.
- `# language=<language_ID>`, see [Using language injection comments](#).

Using language injection comments

To inject a language by means of an injection comment, on a separate line preceding the one that contains the target string literal, add:

```
# language=<language_ID>
```

e.g.

```
# language=HTML
```

NOTE: The syntax of comments should be appropriate for the language that you are using. So you may want to use `// language=...` or `-- language=...` rather than `# language=...`.

Example

1. On the line preceding the one that contains your string literal, depending on the language, type `// language=HTML` or `# language=HTML`.
2. Check the light bulb menu (`Alt+Enter`). The Edit HTML Fragment command is available which means that HTML has been injected into the string literal.
3. Remove the commented line (e.g. `Ctrl+Y`) to come back to the previous state.

Language IDs

The language IDs, generally, are intuitive, e.g. MySQL, RegExp, XML, HTML. If not sure about the language ID, use the suggestion list for the Inject language or reference command. What precedes the opening parentheses there is the language IDs.

See also, [Using language injection prefixes and suffixes](#).

Accessing injection settings

To access the language injection settings:

1. Open the Settings / Preferences dialog (e.g. `Ctrl+Alt+S`).
2. Go to the Language Injections page: Editor | Language Injections.

For more info, see [Language Injections page](#).

Using language injection prefixes and suffixes

Injecting a language may be accompanied with adding a prefix and a suffix. The prefix is added before the injected fragment, and the suffix - after the fragment.

Adding the prefix and the suffix is "imaginary". It doesn't change the actual string value. The prefix and the suffix act as a "wrapper" and their main purpose is to turn the injected fragment into a syntactically complete language unit. In this way, you give PyCharm a broader context for validating the injected code fragment.

When editing your code, you can see the prefix and the suffix only in the fragment editor; the prefix and the suffix are not shown in the main editor.

The prefix and the suffix can be included in the injection comment whose complete form is

```
# language=<language_ID> prefix=<prefix> suffix=<suffix>
```

where the prefix and the suffix are optional.

Example

In this example, we'll remove the opening and closing `<body>` tags from the injected code fragment and add these tags to the injection comment as the prefix and suffix.

1. Remove the opening and closing `<body>` tags: e.g. place the cursor within the injected fragment, press `Ctrl+Shift+Delete` and select Remove Enclosing Tag body.
2. On the line preceding the one that contains your string literal, depending on the language, type `// language=HTML prefix=<body> suffix=</body>` or `# language=HTML prefix=<body> suffix=</body>`.
3. For the injected fragment, open the fragment editor.
Compare the fragments shown in the main and in the fragment editors.

Using Local History

This section describes how to use [Local History](#), which is your personal real-time version control system. Local History is independent of external version control systems and works with the directories of your project even when they are not under any VCS control.

Local history is cleared when you install a new version of PyCharm or when you [invalidate caches](#). Therefore, check in the changes to your version control system before performing these operations.

This section describes how to:

- [View local history of a file or folder](#)
- [View local history of a selection](#)
- [View recent changes](#)
- [Restore files from local history](#)
- [Mark local versions with labels](#)

Viewing Local History of Source Code

On this page:

- [Basics](#)
- [Viewing local history of a source code block](#)

Basics

Besides [file history](#), you can track local changes for a [selected block of source code](#). This history shows only those changes that affect the code fragment.

Viewing local history of a source code block

To view local history of a source code block

1. In the editor, select a fragment of source code.
2. Do one of the following:
 - On the main VCS menu, or on the context menu of the selection, choose Local History | Show History for Selection.
 - Press `Alt+Back Quote`, and choose the desired command from the VCS Operations quick list.

Viewing Local History of a File or Folder

On this page:

- [Basics](#)
- [Viewing local history](#)

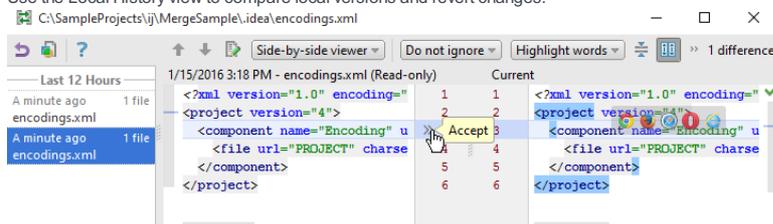
Basics

Local History makes it possible to view changes made to a certain file or a whole directory. Each entry in the Local History dialog box is displayed with its time stamp, action and optional label. So doing, the local history for a file includes all changes that affect both the selected file and the whole project; local history for a folder shows changes to the source code tree in general. You can explore changes, selecting the respective row in the Local History dialog box.

Viewing local history

To view local history

1. Select a folder or file in the Project tool window, or open a file in the editor.
2. Do one of the following:
 - On the main VCS menu, or on the context menu of the selection, choose Local History | Show History.
 - Press `Alt+Back Quote`, and choose the desired command from the VCS Operations quick list.
 - Use [View Recent Changes](#) that shows a summary of recent changes in a single pop-up list. Clicking an entry in this list shows the respective Local History.
3. Use the Local History view to compare local versions and revert changes.



Viewing Recent Changes

On this page:

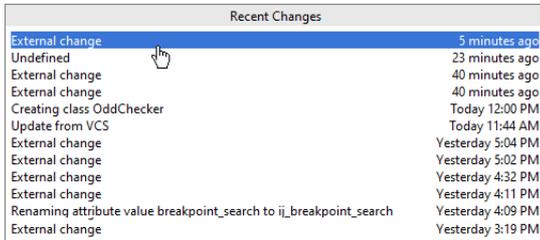
- [Basics](#)
- [Viewing recent changes](#)

Basics

PyCharm allows you to view the summary of recent changes to all recent projects, the PyCharm configuration directory, etc. From the Recent Changes pop-up, you can browse through the history of changes, navigate to a particular change, compare versions, and revert changes if necessary

Viewing recent changes

1. From the main menu, choose View | Recent Changes, or press `Shift+Alt+C`.
2. In the Recent Changes pop-up, select the change you are interested in:



Recent Changes	
External change	5 minutes ago
Undefined	23 minutes ago
External change	40 minutes ago
External change	40 minutes ago
Creating class OddChecker	Today 12:00 PM
Update from VCS	Today 11:44 AM
External change	Yesterday 5:04 PM
External change	Yesterday 5:02 PM
External change	Yesterday 4:32 PM
External change	Yesterday 4:11 PM
Renaming attribute value breakpoint_search to ii_breakpoint_search	Yesterday 4:09 PM
External change	Yesterday 3:19 PM

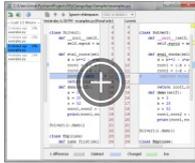
3. In the dialog that opens, review the differences and discard changes if necessary.

Restoring a File from Local History

Rolling back changes from the local history works same way as in the regular version control.

To roll back changes in the local history

1. [Open the Local History view.](#)
2. Select the version you want to roll back to.
3. On the context menu of the selection, choose Revert.



Putting Labels

On this page:

- [Basics](#)
- [Adding a label to a local version](#)

Basics

Before embarking on a risky change to your source code, it is a good idea to mark the stable version with some meaningful label. This will help you quickly roll back to a safe version.

Labels apply to a whole project.

Adding a label to a local version

To add a label to a local version

1. Select a file or folder in the Project tool window, or open a file in the editor.
2. Do one of the following:
 - On the main VCS menu, or on the context menu of the selection, choose Local History | Put Label.
 - Press `Alt+Back Quote` and choose Put Label command from the VCS Operations quick list.
3. In the Put Label dialog box, type the label name.

Version Control with PyCharm

In this part you can find descriptions of procedures that are common to all supported version control systems, or specific to certain VCS integrations:

- [Main Concepts](#)
- [Accessing VCS Operations](#)
- [Managing Projects under Version Control](#)
- [Enabling Version Control](#)
- [Configuring Version Control Options](#)
- [Common Version Control Procedures](#)
- [VCS-Specific Procedures](#)

Concepts of Version Control

PyCharm provides integration with several Version Control Systems (referred to as VCS in documentation). This includes both support of features specific for each VCS as well as unified interface and management for common VCS tasks.

In this part:

- [Supported Version Control Systems](#)
- [Unified Version Control Functionality](#)
- [Directory-Based Versioning Model](#)
- [Changelist](#)
- [Local History](#)
- [Local, Repository, and Incoming Changes](#)
- [Patches](#)
- [Shelved Changes](#)

Supported Version Control Systems

The list of supported integrations of version control systems is determined by the set of currently enabled [plugins](#).

The basic principles of getting started with the supported version control systems (VCS) in PyCharm are rather similar, though some commands and settings are specific and conform to the version control system conventions. In addition to the common information, you can find VCS-specific procedures in the following sections:

- [Using Git Integration](#)
- [Using GitHub Integration](#)
- [Using Subversion Integration](#)
- [Using Mercurial Integration](#)
- [Using CVS Integration](#)
- [Using Perforce Integration](#)

Unified Version Control Functionality

In addition to support for general and individual VCS commands, PyCharm provides several unique features that simplify and speed up the work with any version control system.

- For the projects with VCS support enabled, the standard VCS actions (commit, update, revert, show differences and show history) are added to the main toolbar.
- Commit and update an entire project.
- Uniform interface for configuring common version control system settings.
- Changelists support for all integrated version control systems.
- Next, Previous, Rollback, Show Difference actions are available from the dedicated gutter bar in changed locations.
- View revision history for file/directory.
- Automatic checkout of all affected files when refactoring.
- Advanced Version Control tool window, with multiple dedicated tabs: History, Status, Update Info, etc.

Mind the difference in terminology in the different version control systems. For example, to denote the check-in functionality, Git uses the term *commit*, Subversion uses *submit*, etc.

Directory-Based Versioning Model

PyCharm adopts a directory based model for version control association. A version control system is assigned to a project directory and/or additional directories that are part of or related to the project. Directories under version control are not required to be located under the project root. They can reside in any accessible location.

All the settings files in the `.idea` directory should be [put under version control](#) except the `workspace.xml`, which stores your local preferences. The `workspace.xml` file should be [marked as ignored by VCS](#).

Changelist

A changelist is a set of changes in files that represents a logical change in source code. The changes specified in a changelist are not stored in the repository until committed (pushed).

Any changes made to the source files, are automatically included in the active changelist. Initially, the Default changelist is active, but you can make any other changelist active. The active changelist is displayed on top of the Version Control tool window, with the name being highlighted in bold font.

In addition to the Default changelist, you can create new changelists, delete existing ones (except for the Default changelist), and move files between changelists.

All modified, deleted, unversioned and other files are managed in the [Version Control tool window](#). From this window you can:

- Commit (push) changelists.
- Create new changelists (if you want to keep an eye on certain files and changes).
- Remove existing changelists and set the default changelists.
- Rollback modified files in changelists.
- Add the unversioned files and directories to the version control.
- Move files between changelists.
- Show differences on selected files.
- Refresh the list of VCS changes.
- Jump to the source code from within a changelist.
- Shelve (stash) and unshelve (unstash) changes.

Local History

Your source code constantly changes as you edit, test, or compile. Any version control system tracks the differences between the committed versions, but the local changes between commits pass unnoticed. Local History is your personal version control system that tracks changes to your source code on your computer and enables you to compare versions and roll changes back, if necessary. Local History is always at your disposal, no steps are required to enable it.

Local History is independent of external version control systems and works with the directories of your project even when they are not under any VCS control. It applies to any structural artifacts: a project, a directory or package, a file, a class, class members, tags, or selected fragment of text.

Unlike usual version control systems, Local History is intended for your personal use, it does not support shared access.

With Local History, PyCharm automatically tracks changes you make to the source code, results of refactoring, and state of the source code based on a set of predefined events (testing, deployment, commit or update).

Local History revisions are marked with labels, which are similar to versions in traditional version control systems. Labels based on predefined events are added to the local revisions automatically; besides that, you can put your own labels to the project artifacts to mark your changes. Reverting or viewing differences are performed against these labels.

Local History has certain limitations:

- Tracking local changes is only possible for textual files. Binary files do not have Local History.
- For files larger than 1 Mbyte, Local History tracks only the very fact of changes, but does not preserve the respective contents.
- Local History does not support shared access.

Local, Repository, and Incoming Changes

PyCharm distinguishes between three categories of changes: Local, Repository, and Incoming.

Note Repository and incoming changes are not supported by distributed version control systems, such as Git and Mercurial .

- Local changes: the changes that you have introduced to your local working copy, but have not yet checked in to the repository.
- Repository changes: The changes that you and other team members have checked into the repository
- Incoming changes: the changes that have been checked into the repository, but that you have not checked out locally yet.

All these types of changes are displayed in the respective tabs of the [Version Control](#) tool window. The information on changes is stored in the history cache. Configure the size of this cache and frequency of refreshing information in the [General VCS Settings](#).

Patches

PyCharm helps you create and apply patches to the source code. A patch is a file in the standard text format that has the *.patch extension, and contains a list of differences between two sets of source files.

Patches only contain changes to textual files. Changes to binary files cannot be patched.

This concept is tightly related to the concept of [Shelved Changes](#).

Shelved Changes

You can run into a situation when you are short of time to bring your source code to a certain required condition or you need to work on an urgent high priority task. In this case you might want to put some changes aside and continue working on a stable version.

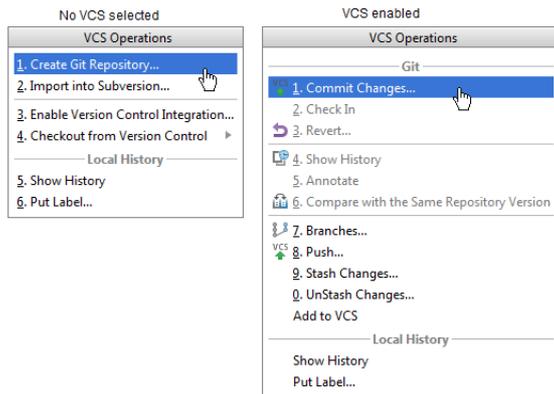
With PyCharm, you can use shelves for storing postponed changes temporarily. In due time, the desired changes can be taken back from the shelf (unshelved).

PyCharm enables shelving both separate files and entire changelists. Accordingly, you can unshelve entire shelves or specific files.

Accessing VCS Operations

Use the VCS Operations Pop-up to quickly invoke most required commands.

Note that the composition of VCS commands available on the pop-up menu, depends on the specific VCS.



To quickly invoke a VCS command using VCS Operations Pop-up

1. Open VCS Operations pop-up, in one of the following ways:
 - On the main menu, choose VCS | VCS Operations Pop-up.
 - Press `Alt+Back Quote`.
2. Choose command from the VCS Operations list. To do that, perform one of the following actions:
 - Click the desired command in the list.
 - Use up and down arrow keys to select the desired command, and then press `Enter`.
 - Press the number key that corresponds to the desired command in the list.

Managing Projects under Version Control

This section contains information related to sharing PyCharm [project](#) files with the other developers:

- [Directory based project format](#)
- [Sharing run/debug configurations](#)
- [Sharing inspection profiles](#)
- [Project settings files to share](#)

Directory based project format

The project settings are stored in the `.idea` directory. This format is used by all PyCharm versions by default. Here is what you need to **share**:

- All the files under `.idea` directory in the project root except `workspace.xml` and `tasks.xml`, storing user-specific settings.

Sharing run/debug configurations

You might want to share [run/debug configurations](#). To do that, just select the check box Share in the selected [run/debug configuration dialog box](#).

The shared run/debug configurations are kept in separate xml files under `.idea\runConfigurations` folder, while the local run/debug configurations are kept in the `.idea\workspace.xml`.

Sharing inspection profiles

To share inspection profiles, make sure to select the check box Share profile on the [Inspections](#) page of the Settings dialog.

The shared inspection profiles are stored in separate xml files under `.idea\inspectionProfiles` folder, while the local profiles are kept in the `.idea\workspace.xml`.

Project settings files to share

The `config` directory has several subfolders that contain xml files with your personal settings. You can easily share your preferred keymaps, color schemes, etc. by copying these files into the corresponding folders on another PyCharm installation. Prior to copying, make sure that PyCharm is not running, because it can erase the newly transferred files before shutting down.

The following is the list of some of the subfolders under the `config` folder, and the settings contained therein.

Folder name	User Settings
<code>codestyles</code>	Contains code style schemes .
<code>colors</code>	Contains editor colors and fonts customization schemes.
<code>filetypes</code>	Contains user-defined file types .
<code>inspection</code>	Contains code inspection profiles .
<code>keymaps</code>	Contains PyCharm keyboard shortcuts customizations.
<code>options</code>	Contains various options, for example, feature usage statistics and macros.
<code>templates</code>	Contains user-defined live templates .
<code>tools</code>	Contains configuration files for the user-defined external tools .
<code>shelf</code>	Contains shelved changes .

Warning! Be careful about sharing the following:

- `dataSources.ids`, `datasources.xml` - these files can contain database passwords.

Also, consider not sharing the following:

- The user dictionaries folder to avoid conflicts if another developer has the same name.

Enabling Version Control

PyCharm supports version control integration at two levels:

- At the IDE level, VCS integration is provided through a set of [bundled plugins](#) enabled by default.
- At the project level, VCS integration is enabled by associating project folders with one or several version control systems.

This section describes how to:

- [Associate a project root](#) with version control system.
- [Associate a directory](#) with version control system.
- [Change the associated VCS](#) for the project root or directory.

Associating a Project Root with a Version Control System

PyCharm allows you to quickly enable your project's integration with a version control system, and associate it with the project root. For instructions on how to associate separate project directories with different version control systems, refer to [Associating a Directory with a Specific Version Control System](#).

To assign a version control system to the project root

1. Choose VCS | Enable Version Control Integration on the main menu, or press `Alt+Back Quote`, and select Enable Version Control Integration....
2. In the [Enable Version Control Integration](#) dialog box that opens, select a version control system from the drop-down list that you want to associate with your [project root](#).

PyCharm supports a [directory-based versioning model](#), which means that each project directory can be associated with a different version control system.

Associating a directory with a version control system

To associate a directory with a version control system, follow these steps:

1. Open [version control settings](#) (File | Settings | Version Control). This page shows a list of project directories and version control systems associated with them (if no directories have been added, the list only contains the project root).
2. Click the Add button  on the right.
3. In the Add VCS Directory Mapping dialog box that opens, select the Directory option. Type the path to the directory that you want to associate with a version control system, or click the Browse button  and select the directory in the [dialog that opens](#).
4. From the VCS drop-down list, select the version control system that will be used to control the files in this directory. Note that this list only contains the version control systems for which the corresponding [Plugins](#) are currently enabled (see [Managing Plugins](#) for details).
5. Optionally, click the Configure VCS button that allows you to specify the settings for the selected version control system. The same settings are also available under the [Version Control settings](#) node.
6. Click OK to save the mapping and return to the Version Control page.

Managing unregistered directories

For projects with Git or Mercurial integration enabled, PyCharm scans project directories to check if there are Git/Mercurial repositories that are not controlled by the IDE. If such repositories are detected, PyCharm displays a notification.

To add an unregistered root, click the Add roots link in the notification. Alternatively, open the [Version Control settings page](#), select the unregistered root you want to add (they are marked grey) and follow the procedure [Associating a Directory with a Specific Version Control System](#).

If you do not want to be notified about these roots again, click the Ignore link in the notification. Note that if new unregistered repositories are added to the project, PyCharm will notify you about them.

To change the version control system associated with a directory

1. Open the [Version Control](#) settings page. This settings page displays a table of directories with associated version control systems.
2. In the table, locate the row that corresponds to the directory which you want to put under another version control system.
3. Click the VCS column. From the drop-down list that appears, select a new version control system.

Tip Optionally, click the Configure VCS button. The Version Control Configurations dialog box opens where you can configure settings for the selected version control system (see the corresponding configuration settings in the [Version Control](#) settings page reference for details).

4. Click OK to save the new mapping and close the Version Control dialog box.

Configuring Version Control Options

Configuring version control options involves:

- [Configuring General VCS Settings](#)
- [Configuring VCS-Specific Settings](#)

Configuring General VCS Settings

General version control settings apply to all version control systems integrated with PyCharm. General settings are specified on the [Version Control](#) page of the [Settings](#) dialog box, and include defining actions that require confirmation, background operations, ignored files, issue navigation, and depth of history.

To configure general version control settings, follow these general steps

1. [Open the Settings dialog box](#) and then click Version Control.
2. Specify which version control related actions should require [confirmation](#).
3. Specify which operations should be performed in the [background](#).
4. Create a list of [files to be ignored](#) by version control systems.
5. Configure [history cache handling](#).
6. Define [issue navigation](#) rules to switch from check-in comments to corresponding issues in a bug tracking system.

Specifying Actions to Confirm

You can define that certain version control related activities should be performed only upon confirmation from your side. The activities for which confirmation is required are specified in the [Confirmation](#) settings page.

To specify which actions should require confirmation

1. Below the [Version Control](#) node in the settings, select the [Confirmation](#) page.
2. In the Files Creation/Deletion area, define how files created or deleted in PyCharm should be added to or removed from a version control system. The available options are:
 - Show options before adding to version control
 - Add silently
 - Do not add
3. In the Display options dialog when these commands are invoked area, specify the commands, for which you want PyCharm to display the Options dialog box. The available options are:
 - Check Out
 - Status
 - Get Latest Version
 - Update
 - Undo Check Out
4. Configure additional settings for working with changelists by selecting or clearing the corresponding check boxes:
 - Specify whether the read-only state of files should require explicit cancellation.
 - Specify whether meaningful comments on committing files to the repository should be required.
 - Specify whether uncommitted changes should be moved to another changelist.
 - Specify whether and how to create a changelist if the commit operation fails.

You can enable background execution of certain version control related activities. These activities are specified in the [Background](#) settings page.

To specify the operations to run in the background

1. Below the [Version Control](#) node in the settings, select the [Background](#) page.
2. Enable background execution of the necessary actions by selecting the corresponding check boxes. Background execution can be set for the following actions:
 - [Update](#)
 - [Commit](#)
 - [Checkout](#)
 - Edit/Checkout
 - [Add/Remove](#)
 - [Revert](#)
 - [History Cache Handling](#)
 - Detecting "changed on server" conflicts

You can configure handling of the history cache in the [Background](#) settings page.

To configure history cache handling

1. Below the [Version Control](#) node in the settings, select the [Background](#) page.
2. Set the cache scope. Depending on the version control system you are using, do one of the following:
 - Specify the maximum number of changelists to be stored in the cache.
 - Specify the maximum number of days for a changelist to be stored in the cache.
3. Specify whether and how often (in minutes) you want your version control system to refresh the cache.

Configuring Ignored Files

On this page:

- [Basics](#)
- [Defining a list of ignored files](#)
- [Patterns](#)

Basics

Sometimes you may need to leave files of certain types unversioned. These can be VCS administration files, artifacts of utilities, backup copies, etc. You can create a global ignore list that will be stored in the workspace file and applied to all supported version control systems.

The files you want to ignore can be appointed explicitly by their names or through name patterns with wildcards. To ignore a directory, you need to specify the full path to it relative to the project root.

Use the [Ignored Files](#) settings page to list files that must be excluded from version control operations.

Tip If the version control system that you are using has its own ignore facilities, use the corresponding native command provided by the version control integration.

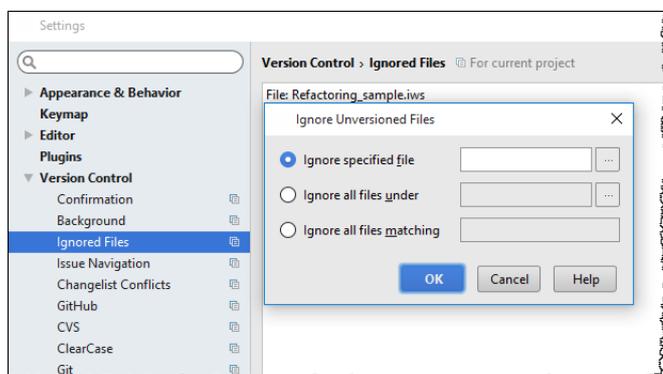
Note that once you've added a file to a version control system, ignoring it will have no effect. You need to remove it from your VCS first.

Defining a list of ignored files

1. Open the [Ignored Files](#) settings page by doing one of the following:

- Under the [Version Control](#) node of the [Settings](#) dialog box, click Ignored Files.
- In the [Local Changes](#) tab of the [Version Control](#) tool window, click the Configure Ignored Files toolbar button .

The Ignored Files dialog box opens.



2. Click **+** ([Alt+Insert](#)) to create a new entry, or select an existing entry and click  ([Enter](#)). The Ignore Unversioned Files dialog box opens.

3. Explicitly specify the files/directories to be ignored or define file name patterns. Do one of the following:

- Choose the **Ignore specified file** option and specify the file name relative to the project root, for example, `my_folder/my_subfolder1/my_subfolder2/my_file`. Type the name manually or click the **Browse** button  and select the desired file in the **Select File to Ignore** dialog box.
- Choose the **Ignore all files under** option and specify the directory whose contents should be ignored. Type the directory name relative to the project root, for example, `my_folder/my_subfolder1/`, or click **Browse** button  and select the desired folder in the **Select Directory to Ignore** dialog box. The rule is applied recursively to all subdirectories of the specified directory. If a directory has several subdirectories and you want only one of them ignored, specify the required directory explicitly, for example, `my_folder/my_subfolder1/my_subfolder2/`.
- Select the **Ignore all files matching** option and type the pattern that defines the names of files to be ignored. The rule is applied to all the directories under the project root. Using wildcards in combination with slashes (`/`) to restrict the scope to a certain directory is not supported. Patterns that define files to ignore, make use of two [wildcards](#).

4. Create as many entries as you need and close the dialog box.

Tip You can also add files to the ignore list on-the-fly by using the **Ignore** command on the context menu of a newly added file under the **Unversioned Files** node in the [Local Changes](#) tab of the [Version Control](#) tool window.

Patterns

Two characters can be used as wildcards:

- `*`: to replace any string.
- `?`: to replace a single character.

For example, `*.iml` will ignore all files with the `iml` extension; `*.?.ml` will ignore all files whose extension ends with `ml`.

Configuring VCS-Specific Settings

Certain settings are applicable to the version control systems assigned to the [whole project, or its directories](#). The others are related to the selected version control systems. Use the respective VCS pages under the VCSs node of the [Settings](#) dialog box to define VCS-specific settings.

To configure VCS-specific settings, follow these general steps

1. Open the [Settings](#) dialog box and then click Version Control.
2. Click a page that corresponds to the VCS to be configured.
3. Set up options as required. For detailed information, see VCS-specific pages of the Version Control settings.

Common Version Control Procedures

PyCharm makes it possible to interact with the different version control systems (VCS). Regardless of which VCS you use, certain basic operations are common to all (or almost all) of them. This section covers these basic operations and explains how to perform them from within PyCharm:

- [Adding Files to Version Control](#)
- [Browsing Contents of the Repository](#)
- [Changing Read-Only Status of Files](#)
- [Checking In Files](#)
- [Checking Project Files Status](#)
- [Copying, Renaming and Moving Files](#)
- [Getting Local Working Copy of the Repository](#)
- [Deleting Files from the Repository](#)
- [Handling Differences](#)
- [Handling Issues](#)
- [Managing Changelists](#)
- [Refreshing Status](#)
- [Reverting Local Changes](#)
- [Reverting to a Previous Version](#)
- [Shelving and Unshelving Changes](#)
- [Updating Local Information](#)
- [Using Patches](#)
- [Viewing Changes Information](#)

Adding Files to Version Control

If a new file is created with PyCharm in a directory that is already under version control, it automatically adds to the active changelist with the status Added. All you have to do, is to commit this change.

PyCharm's behavior on adding files is configurable in the [General Settings tab](#) of the Version Control dialog.

If a new file is created outside of PyCharm, or silent adding is disabled, you have to explicitly add it to the version control.

Another approach is VCS-specific. You can import an entire directory to your repository, provided that you have the corresponding access rights. This action is helpful for putting a whole project under version control.

To explicitly add a file to version control

1. Select file in the Project tool window.
2. On the main Version Control menu or on the context menu of the selected file, choose <VCS> | Add.
Alternatively, use the Version Control tool window. Select the desired files in the Unversioned files changelist in the Local Changes tab, and choose Add to VCS on the context menu, or just drag it to the target changelist.
3. Select the added file in the changelist and [check in \(commit\) changes](#).

Browsing Contents of the Repository

Prior to checking files out, you can browse the contents of the repository. This action is available for CVS, Git and SVN integrations.

- [Browsing CVS Repository](#)
- [Browsing Subversion Repository](#)

Browsing contents of CVS, Git and SVN repositories is always available, even when the respective VCS is not enabled in project. All you need is a valid user account.

Changing Read-Only Status of Files

In this section:

- [Basics](#)
- [Enabling explicit removal of read-only status](#)
- [Changing writable status by icon](#)
- [Example](#)

Basics

Different version control systems have different semantics for the action of removing read-only status from a file so that you can edit it. Some systems never put read-only status on local files at all unless specifically configured to do so (i.e. the system is configured to support the file locking model).

Different version control systems use different names for this action: check out, edit, Open for Edit, or Get. Regardless of the terminology used by your VCS, if it sets read-only status on your local working files, you can remove read-only status and make files writeable from within PyCharm, which will also take care of setting a lock on the server, or take whatever other action is required by the VCS, via the respective VCS integration.

This behavior is configurable in the [Confirmation](#) page of the Version Control settings.

Enabling explicit removal of read-only status

To enable explicit removal of read-only status:

- In the [Confirmation](#) page of the [Version Control settings](#), check the option Show "Clear Read-Only Status" Dialog.

Changing writable status by icon

You can make a file writable using the lock icon in the Status bar. Open the desired file in the editor, and click the lock, as shown below:



Example

Removing read-only status in Perforce looks as follows:

1. With the Show "Clear Read-Only Status" Dialog option enabled, an attempt to edit a file brings up the respective dialog box.
If you click the radio button Using file system, the file will not be added to the default changelist. If you click radio button Using version control integration, the file is added to the default changelist.

Checking In Files

In this section:

- [Basics](#)
- [Checking changed files in](#)

Basics

Different version control systems have different semantics for the action of uploading changed files to the repository. Two common terms are check in and commit.

In those version control systems, for example, [Git](#), that distinguish between local and remote repositories, the term commit denotes uploading changes to a local repository. Uploading changes to a remote repository is referred to as push.

Regardless of the terminology, you can perform this operation with the VCS configured for a directory from within PyCharm.

Checking changed files in

To check in (commit) changed files, perform these general steps

1. In the [Version Control](#) tool window, select one or more files you want to check in (commit) to version control.
2. Open the [Commit Changes](#) dialog box by doing one of the following:
 - On the Version Control tool window toolbar, or on the main toolbar, click .
 - Press `Ctrl+K`.
 - On the main menu, choose VCS | Commit Changes.
3. Review the changes to be committed in the [Details](#) pane. To do that, unfold the Details pane if it is hidden, and select the file in question in the [Changed Files](#) area.
The Details pane shows the base version and the local copy of the selected file.

Examine the details of each change:

- To move to the next updated piece of code, click the Next Change button .
 - To return to the previous updated code fragment, click the Previous Change button .
 - To expand or narrow the context of an updated code fragment, position the cursor at the change in question, click the More/Less Lines button , and then specify the number of lines to be shown above and below the current code fragment.
4. Add a commit comment. As you type, PyCharm checks the spelling and highlights words in question, provided that the [Spelling code inspection](#) is enabled.
 5. Specify which actions should be performed on the files before and after submitting them to the repository.
 6. Click the Submit/Commit button to launch the [Check-in Changes](#) operations.

Tip For Git and Mercurial. To have the changes immediately pushed to your Git or Mercurial repository, do one of the following:

- Hover the mouse pointer over the Submit/Commit button and select Commit and Push on the context menu.
- From the Submit/Commit drop-down list, select Commit and Push.

7. To save the changes as a patch in a text file, hover the mouse pointer over the Submit/Commit button and select Create Patch on the context menu.

Tip Alternatively, use the Submit/Commit drop-down list to select the Create Patch item.

In the Create Patch dialog box, that opens, [configure the patch creation](#).

8. If any error occurs when trying to commit, PyCharm displays an error message. For example, you might have changed a file that has been already edited by another team member, or you might run into a branch conflict. In these cases, you need to [merge edits](#), or [update your local copy](#). The error messages are VCS-specific.

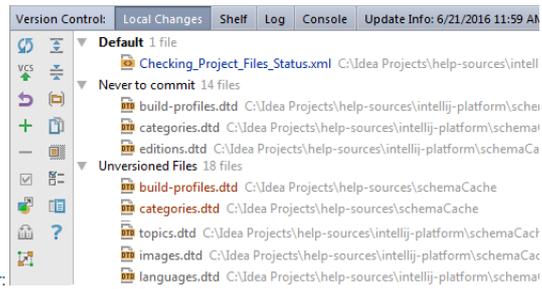
Tip Users of [JetBrains TeamCity](#) can obtain the TeamCity plugin for PyCharm. Among the features of this plugin is Remote Run, which enables you to create a special *personal* build that does not affect the *real* build. Your changes are built and run through your test suite. If all tests are passed, your changes are automatically committed to version control.

Checking Project Files Status

Apart from [indicating the status](#) of the currently opened file relative to the repository, PyCharm provides you with an accumulated view of the project files' statuses.

To view the differences between the current state of the project files and the repository, do the following:

1. From the main menu, choose VCS | Refresh File Status.
2. Switch to the [Version Control tool window](#) and open the [Local Changes](#) tab.

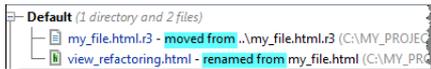


The status of each file is indicated by the color:

Copying, Renaming and Moving Files

There is no automatic renaming or moving files in the version control systems. Working with the clients, you have to manually copy a file to a new location, change its name, add to VCS and remove the old file. PyCharm's refactoring features make it possible to easily move and rename files under version control, performing all actions involved in these functions.

In the Version Control tool window, next to the resulting files PyCharm explicitly states: `renamed from <file name>` or: `copied from <folder name>`.



To copy, move or rename a file under version control

1. Select the desired file in the Project tool window, and perform refactoring procedures, as described in the sections:
 - [Copy/Clone](#)
 - [Move](#)
 - [Rename](#)

All added and deleted files are placed to a changelist.

2. [Commit changes](#) to the repository.

Tip You can revert refactorings, using the Undo command.

On this page:

- [Basics](#)
- [Checking out](#)

Basics

As soon as [version control support is enabled](#) in PyCharm, you can retrieve the data from the repository. Depending on your purpose and workflow, choose one of the following approaches:

- [Check out](#) the repository sources to a [content root](#) of an existing project. Then set up version control in the project, if it has not been done before, and put the downloaded sources under control of the VCS used.
- [Check out](#) the repository sources to a location of your choice and have PyCharm [create a project around them](#). This does not require any extra steps from your side. PyCharm arranges the downloaded sources in a project itself and suggests to open it as soon as the check-out is completed. Upon your consent, the [New Project from Existing Code Wizard](#) starts. When the project is created, set up version control in it.

Warning! This approach is not available for Perforce integration.

Checking out

The check-out procedure depends on the type of VCS you use. Refer to the following sections for details:

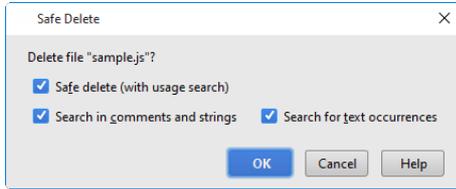
- [Retrieving Files from a CVS Repository](#)
- [Retrieving Files from an SVN Repository](#)
- [Cloning a Remote Git Repository](#)
- [Cloning a Remote Mercurial Repository](#)
- [Using Perforce Integration](#)

Deleting Files from the Repository

If you delete a file under version control, it still exists in the repository until deletion is committed. A deleted file is placed to the active changelist, and is displayed in grey font.

To delete a file

1. Select a file in any navigation view, and press `Delete`, or choose Delete on the context menu.
2. In the dialog that opens, you can opt to delete file without search for usages, or perform safe delete, to make sure that you are deleting an unused file, by checking Safe delete option. In this case, specify the refactoring options.



The encountered usages are reported in the Usages Detected dialog box. You can view and correct these usages in the Safe Delete tab of the Find tool window. The deleted files are added to a changelist.

3. [Commit changes](#) to the repository.

Comparing File Versions

PyCharm allows you to compare the local copy of a file with its repository versions. The following options are possible:

- [Compare your local copy with the repository version to which you have last synchronized](#)
- If somebody else has committed changes since your last update [compare your local copy with newest repository version](#)
- [Compare your local version with any repository version](#)

For some version control systems, it is possible to compare a file with a branch version. The differences are displayed in the Differences viewer.

Note You can explore changes to binary files in the same way as to textual files. For example, use this feature to check changes to images.

To compare with a repository version to which you last synchronized

1. Select a file in the [Project](#) tool window, or open it in the editor.
2. Do one of the following:
 - From the main VCS menu, or on the context menu of a file, choose <VCS> | Compare with the same repository version.
 - Select a file in the [Local CHanges](#) tab of the [Version Control](#) tool window, and choose Show Diff from the context menu.

To compare with the latest repository version

1. Select a file in the [Project](#) tool window, or open it in the editor.
2. On the main VCS menu, or on the context menu of a file, choose <VCS> | Compare with the latest repository version.

To compare with the specified version of a file

1. Select a file in the [Project](#) tool window, or open it in the editor.
2. On the main VCS menu, or on the context menu of a file, choose <VCS> | Compare with.
3. In the File Revision pop-up window, click the version to compare with.

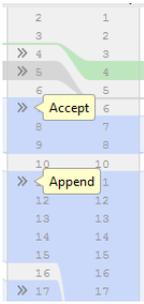
Tip Alternatively, use the [History](#) view of a file. Select the desired version, and choose Compare with Local on its context menu, or click  on the toolbar.

Integrating Differences

You may need to integrate the contents of different versions of a file, for example integrate changes from a certain branch into your local copy.

To integrate changes into your working copy, do the following:

1. Select the desired file and [compare it with a version](#)
2. In the [Differences Viewer](#), use the *chevron* buttons » « for any block of lines that existed in the read-only copy and were changed or deleted in your local copy. If you keep the `Ctrl` key pressed, the *chevron* buttons turn into ⇐ ⇒. Click these buttons to apply changes to the current local version of the file.



Integrating Project

You may need to integrate your local version of a project into a certain revision of that project in the repository.

Integrating a project is only available for projects that contain directories associated with Perforce or Subversion. If neither of these two VCSs is involved in your project, the Integrate Project command is disabled.

To integrate a project, do the following:

1. On the main menu, choose VCS | Integrate Project. The [Integrate Project](#) dialog appears.
2. Select the [Subversion](#) or the [Perforce](#) tab, and configure the merge options as required (see [Integrating SVN Projects or Directories](#) or [Integrating Perforce Files](#) respectively for details .

Resolving Conflicts

Depending on your version control system, conflicts may arise in different situations.

When you work in a team, you may come across a situation when somebody commits changes to a file you are currently working on. If these changes do not overlap (i.e. changes were made to different lines of code), the conflicting files are merged automatically. However, if the same lines were affected, your version control system cannot randomly pick one side over the other, and asks you to resolve the conflict.

Conflicts may also arise when merging, rebasing or cherry-picking branches.

In this topic:

- [Non-Distributed Version Control Systems](#)
- [Distributed Version Control Systems](#)
- [Conflict Resolution Tool](#)

Non-Distributed Version Control Systems

When you try to edit a file that has a newer version on the server, PyCharm informs you about that, showing a message pop-up window in the editor:



In this case, you should update your local version before changing the file, or merge changes later.

If you attempt to commit a file that has a newer repository version, the commit fails, and an error is displayed in the bottom right corner telling you that the file you are trying to commit is out of date.

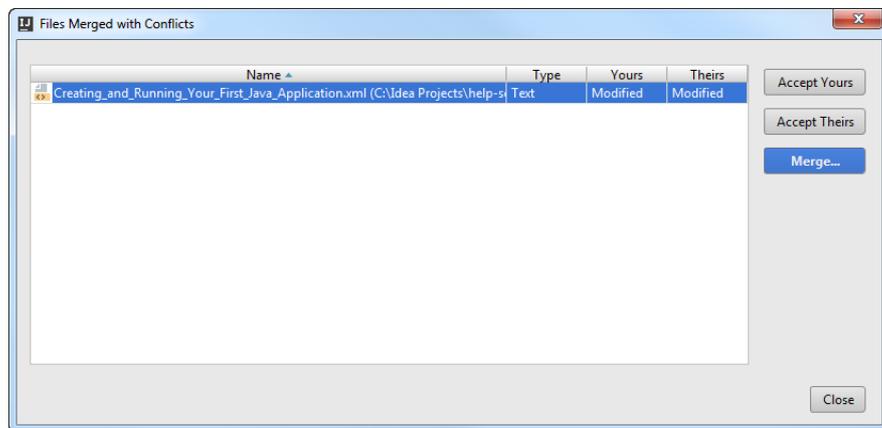


If you synchronize a file that already has local changes with a newer repository version committed by somebody else, a conflict occurs. The conflicting file gets the **Merged with conflicts** status. The file remains in the same changelist in the **Local Changes** tab of the **Version Control** tool window, but its name is highlighted in red. If the file is currently opened in the editor, the file name on the tab header is also highlighted in red.

Distributed Version Control Systems

Under distributed version control systems, such as Git and Mercurial, conflicts arise when a file you have committed locally has changes to the same lines of code as the latest upstream version and when you attempt to perform one of the following operations: [pull](#), [merge](#), [rebase](#), [cherry-pick](#), [unstash](#), or [apply patch](#).

If there are conflicts, these operations will fail and you will be prompted to accept the upstream version, prefer your version, or merge the changes manually:

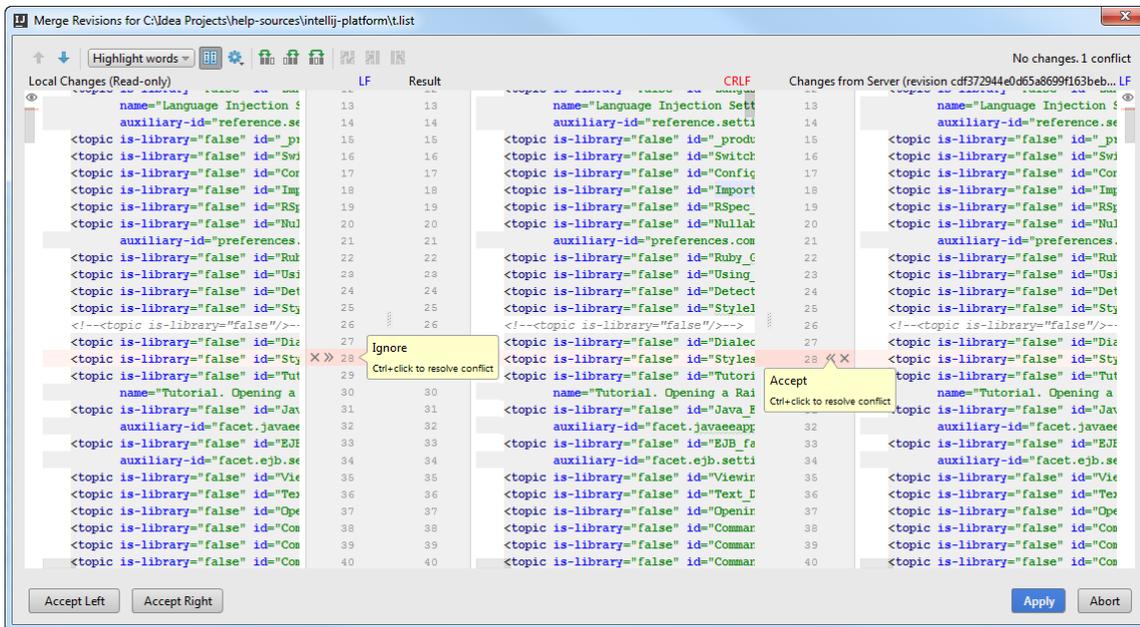


Conflict Resolution Tool

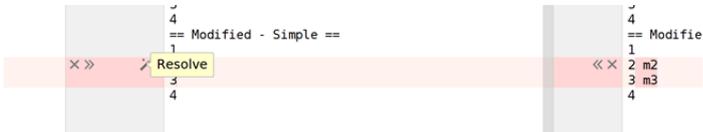
PyCharm provides a tool for resolving conflicts locally. This tool consists of three panes. The left pane shows the read-only local copy, the right pane shows the read-only version checked in to the repository. The central pane shows a fully-functional editor with the results of merging and conflict resolving are displayed. Initially, the contents of this pane is the same as the **base revision** of the file, that is, the revision from which both conflicting versions are derived.

To resolve conflicts, do the following:

1. Click Merge in the Files Merged with Conflicts dialog, or select the conflicting file in the editor and choose **VCS | <your_VCS> | Resolve Conflicts** from the main menu.
2. To automatically merge all non-conflicting changes, click  (Apply All Non-Conflicting Changes) on the toolbar. You can also use the  (Apply Non-Conflicting Changes from the Left) and  (Apply Non-Conflicting Changes from the Right) to merge non-conflicting changes from the left/right parts of the dialog respectively.
3. To resolve conflicts, select the change to be merged in the left or right pane. Use the **chevron** () or the  buttons to apply or discard the selected change:



For simple conflicts (for example, if the beginning and the end of the same line have been modified in different file revisions), the Resolve option is available that allows merging the changes in one click:



Note Such conflicts are not resolved with the Apply All Non-Conflicting Changes action since you must make sure that they are resolved properly.

4. Review merge results in the central pane and click Apply.

Tip You can configure PyCharm to always apply non-conflicting changes automatically instead of telling it to do so from the Merge dialog. To do this, in the [Settings/Preferences dialog](#), expand the Tools | Diff Merge node in the left pane and select the Automatically apply non-conflicting changes option.

Running PyCharm as a Diff or Merge Command Line Tool

In this topic:

- [Overview](#)
- [Enabling invocation of PyCharm operations from the command line](#)
- [Comparing files using PyCharm as a command line tool](#)
- [Merging files using PyCharm as a command line tool](#)
 - [Passing three arguments to merge tool](#)

Overview

Besides using PyCharm as an Integrated Development Environment, you can use it as a command line tool for comparing and merging files.

PyCharm executable is platform-dependent:

- **Windows:** `PyCharm XX\bin\pycharm.exe` or `PyCharm XX\bin\pycharm.bat`
- **UNIX:** `PyCharm XX/bin/pycharm.sh`
- **macOS:** `/Applications/PyCharm XX.app/Contents/MacOS/pycharm`
To add the launcher to your path, add its containing directory `/Applications/PyCharm.app/Contents/MacOS`.

However, for macOS and UNIX, one should create a wrapper script, since this helps avoid some drawbacks related to the usage of PyCharm launcher.

Enabling invocation of PyCharm operations from the command line

For macOS and UNIX platforms, we recommend creating the wrapper script, or the command line launcher to integrate PyCharm with your shell. Then, you need to ensure that the created launcher script is within the search path of your shell.

On Windows, we recommend to add the path to PyCharm to the environment variable `Path`. Everything is done outside of PyCharm, with the PyCharm executable.

Note that if you have specified location of the PyCharm executable as a `Path` environment variable, the command will work no matter which directory you are currently in.

To enable invoking PyCharm operations from the command line, follow these steps

- On **macOS** or **UNIX**:
 1. Make sure PyCharm is running.
 2. On the main menu, choose Tools | Create Command-line Launcher. The dialog box Create Launcher Script opens, with the suggested path and name of the launcher script. You can accept default, or specify your own path.
Make notice of it, as you'll need it later.
 3. Outside of PyCharm, add the path and name of the launcher script to your path.
- On **Windows**:
 - Specify the location of the PyCharm executable in the `Path` system environment variable. In this case, you will be able to invoke the PyCharm executable and other PyCharm commands from any directory.

Comparing files using PyCharm as a command line tool

To compare two files using PyCharm as a diff command line tool

1. [Enable invoking PyCharm operations from the command line](#).
2. Type the following command at the command line prompt:

```
<PyCharm launcher(Windows) or wrapper script (MacOS or UNIX)> diff <path to file1> <path to file2>
```

where `file1` is your local copy, `file2` is the repository version.

For example:

```
pycharm diff README.md.bak README.md
```

Merging files using PyCharm as a command line tool

Most often you need to merge three versions of the same file: your local version, the version in the repository or in the upstream, and the base revision, which is the origin for the two diverged versions.

To merge files using PyCharm as a command line tool

1. [Enable invoking PyCharm operations from the command line](#).
2. Type the following command at the command line prompt:

```
<PyCharm launcher(Windows) or wrapper script (MacOS or UNIX)>  
merge <path to file1> <path to file2> <path to file3> <path to output>
```

where `file1` is your local copy, `file2` is the repository version, `file3` is the base revision for `file1` and `file2`, and `output` is the file to save the merge results in (optional).

Passing three arguments to merge tool

It is possible to pass just three arguments to the merge tool:

```
<path to file1> <path to file2> <path to output>.
```

In this case, the contents of the output will be taken as the base revision:

```
<PyCharm launcher> merge <path to file1> <path to file2> <path to output> <path to output>
```

See the example in this [blog](#) to learn how to use PyCharm diff and merge tool with Git.

Handling Issues

With PyCharm, you can connect your check-in comments with the bug tracker or any issues data base and navigate from committed changes to the issues related to these changes.

To enable this navigation, you need to specify a so called **issue navigation pattern**, which means:

1. Figure out an **issue ID pattern**, that is, a format according to which you will reference issues in commit messages, and define this issue pattern using a regular expression.
2. Define the link to the referenced issue by combining the URL address of your tracking system and a regular expression to identify the issue ID.

In other words, an **issue navigation pattern** maps an **issue ID pattern** in commit messages and URL addresses of referenced issues. As soon as PyCharm encounters a match to the issue ID pattern in a commit message, the match is displayed as a link in the **Changes** and **Version Control** tool windows. If you mention several issues, all of them will show up as links. Clicking such link opens the matching issue in the browser according to the defined link.

On this page:

- [Example](#)
- [Enabling navigation from commit messages to issues related to them](#)
- [Navigating from a commit message to the related issues](#)

Example

Issue ID pattern	The regular expression that defines the format in which issues are referenced in commit messages.
	<pre>[A-Z]+\-\d+</pre>
	This regular expressions matches all character strings that consist of two substrings separated by an n-dash character: <ol style="list-style-type: none">1. Substring 1: An unlimited number of upper case alphabetic characters.2. Substring 2: An unlimited number of digital characters.
Issue link pattern	A combination of the URL address of your issue tracking system and a regular expression that identifies issues in it.
	<pre>http://mytracker/issue/\$0</pre>
	Here <code>\$0</code> indicates a back reference to the entire match. This means that as soon as PyCharm detects a match in a commit message, it is added to the URL address of the tracker as is.
Matching issue ID	PyCharm detects the following reference to an issue in the commit message of interest:
	<pre>MYPROJECT-110</pre>
Composed issue link	In accordance with the above issue navigation pattern, the detected matching reference is added to the URL of the tracker as is, so the link to the referenced issue is composed as follows:
	<pre>http://mytracker/issue/MYPROJECT-110</pre>

To enable navigating from commit messages to issues related to them

1. [Open the Settings dialog box](#), and click Issue Navigation under the Version Control node.
2. In the [Issue Navigation](#) dialog box that opens, configure a list of **issue navigation patterns** by setting correspondence between issue patterns in commit messages and URL addresses of referenced issues.
 - If you are using [JIRA](#) or [our bug tracking system YouTrack](#), click the Add JIRA pattern  or Add YouTrack Pattern  respectively, and type the URL to the installation of bug tracking system in question.

PyCharm adds the regular expression that defines such pattern automatically.
 - For other issue tracking systems, click the Add button  to create a new entry or select an existing entry and click the Edit button. In the Add Issue Navigation Link dialog box that opens, specify the following:
 1. The **regular expression** that defines the **issue pattern** in a commit message.
 2. The **replacement expression** that defines the URL to access the corresponding referenced issue.
 - To remove an **issue navigation pattern**, select it in the list and click Remove .

To navigate from a commit message to the related issues

1. Open one of the following views:
 - Local Changes, Incoming, or Log tab of the Version Control tool window.

- History tab of the Version Control tool window.
- [Changes Browser](#).

2. Find the commit of interest and click the hyperlink to the related issue.

Managing Changelists

This section describes how to:

- [Create, delete and rename](#) changelists.
- [Assign active changelists](#).
- [Group items in a changelist](#).
- [Move files between changelists](#).
- [Jump from an item in a changelist to the corresponding source code in the editor](#).

Assigning an Active Changelist

Active changelist is the one to which the changed files are added automatically. The name of the active changelist is highlighted in bold font.

To assign an active changelist

1. Select a changelist in the Version Control tool window.
2. On the context menu of the selected changelist, click Set Active Changelist.

To create a new changelist

1. In the toolbar of the Version Control tool window, click **+** button.

Tip You can also use one of these alternatives:

- Right-click anywhere in the Local Changes tab of the Version Control tool window and choose New Changelist on the context menu.
- Press `Alt+Insert`.

2. In the New Changelist dialog, specify the name of the new changelist, and optional comment.
3. Click OK.

Deleting a Changelist

When a [changelist](#) is deleted, all changes are moved to the active changelist.

To delete a changelist

1. In the [Version Control tool window](#), select a changelist to be deleted.
2. In the toolbar of the Version Control tool window, click . Alternatively, right-click the changelist node and choose Delete Changelist on the context menu, or just press  key.
3. If the changelist is not empty, you are prompted to confirm deletion and move uncommitted items to the active changelist. If you attempt to delete an active changelist, you are prompted to specify another.
If Perforce is used for a certain directory, deleting the default changelist is not allowed.

Grouping Changelist Items by Folder

Within each node of the [Version Control tool window](#), you can display the modified files as a flat list, or as a directory tree.

To toggle grouping items by directories, do one of the following

- On the toolbar of the Version Control tool window, click .
- Use `Ctrl+P` keyboard shortcut.

To move items between changelists in the Version Control tool window

1. In the [Version Control](#) tool window, select one or more desired items in a changelist. Use the **Ctrl** and **Shift** keys for multiple selection.
2. Choose Move to Another Changelist on the context menu of the selection.
You can also use one of these alternatives:
 - Click the Move to Another Changelist button  on the toolbar of the tool window.
 - Press **F6**.
 - Drag the selected items to the target changelist.
3. In the Choose Changelist dialog box, specify the changelist to move the selected items to:
 - If the target changelist exists, click the Existing Changelist option and select the desired changelist from the drop-down list.
 - To create a changelist, click the New Changelist option, type the name of the new changelist, and optionally provide a description.

To navigate from an entry in the changelist to the source code

1. In the Version Control Tool Window, expand a changelist and select the desired entry.
2. On the context menu of the selection, choose Jump to Source, or press **F4**.

To rename a changelist

1. Select a changelist in the Version Control Tool Window.
2. On the context menu, choose Edit Changelist.
3. In the Edit Changelist dialog, specify the new name and optional description, and click OK, or use the **Shift+F6** keyboard shortcut.

Refreshing Status

This feature is helpful if a server, that holds the sources of the project files, is down. The statuses of files and directories become irrelevant, and you are unable to work with the local copy of the project. To avoid such situation, you need to refresh the status of your source files.

When this command is applied, PyCharm refreshes the status of each file, no matter whether the file has been changed from PyCharm itself or using any other application.

When working under [Perforce](#) control, you can run refresh in two modes:

- **Standard Refresh** takes into consideration only the changes made through the PyCharm integration with Perforce. This improves performance because does not require connecting to the server. However, this approach does not let you know about the changes made outside PyCharm, for example, right through the `p4v client` application.
- **Force Refresh** considers all the changes made to project, both from PyCharm and from any other application, for example, right through `p4v client`.

The status of a file is refreshed only in accordance with the changes in your local workspace. Any changes on the server can be reflected only through [synchronization](#).

To refresh the status of files in your project, do one of the following

- On the main menu, choose VCS | Refresh File Status.
- In the Version Control tool window, press `Ctrl+F5`.
- In the Version Control tool window, click the Refresh toolbar button .
- For [Perforce](#) integration, do one of the following:
 - To run **Standard Refresh**, click the Refresh toolbar button  or press `Ctrl+F5`.
 - To run **Force Refresh**, click the Force Refresh toolbar button .

Reverting Local Changes

When you modify, add, or delete files, which are under version control, you are always able to revert such changes, rolling back the file's contents to what it was before the last successful update, check out, or commit.

The exact name of the command (revert or roll back), and the type of the action performed when you revert changes, depend on the file status and VCS used for a particular directory.

To revert local changes, do one of the following

- In the Local Changes tab of the Version Control tool window, select one or more items in the relevant [changelist](#), then choose Revert on the context menu of the selection or click the Revert button  on the toolbar of the Version Control tool window.
- Select the file to be reverted and choose the relevant VCS-specific revert (roll back) command on the VCS | <VCS> menu.

Reverting to a Previous Version

You can restore any previous version of a file, using the History view of a file. So doing, the current content of the file is replaced with the copy of the older content. After such operation, you have to commit the file to bring the repository up to date.

The exact name of the command (revert or roll back) depends on the specific VCS.

To revert a file to its previous version

1. Select the desired file in the Project tool window, and [open its history](#).
2. In the History tab, select the desired revision.
3. On the context menu of the selected entry, choose Get, or click the  button on the toolbar.

Shelving and Unshelving Changes

Shelving is temporarily storing pending changes you have not committed yet. This is useful, for example, if you need to switch to another high priority task and you want to set your changes aside to work on them later.

With PyCharm, you can shelve both separate files and entire changelists.

Once shelved, a change can be applied as many times as you need by unshelving and subsequently restoring it on the shelf.

Shelving changes

1. In the [Local Changes](#) tab of the [Version Control](#) tool window, select the files or a changelist you want to put to a shelf. On the main Version Control menu or on the context menu of the selection, choose [Shelve changes](#).
2. In the [Shelve Changes](#) dialog box, review the list of modified files and make sure that the files you are going to shelve have been checked out (for non-distributive version control systems).
3. In the Commit Message field, enter the name of the shelf to be created and click the [Shelve Changes](#) button.

Tip You can switch to a different changelist from the [Shelve Changes](#) dialog box by choosing the Changelist drop-down.

You can also shelve changes silently, without displaying the [Shelve Changes](#) dialog. To do this, select a file or a changelist you want to shelve, and click the [Shelve Silently](#) icon  on the toolbar, or press [Ctrl+Alt+H](#). The name of the changelist containing the changes you want to shelve will be used as the shelf name.

Tip You can also drag-and-drop a file or a changelist from the [Local Changes](#) tab to the [Shelf](#) tab to shelve it silently.

Unshelving changes

Unshelving is moving postponed changes from a shelf to a pending changelist. Unshelved changes can be filtered out from view or removed from the shelf.

1. In the [Shelf](#) tab of the [Version Control](#) tool window, select a changelist or files you want to unshelve.
2. If you want unshelved changes to be displayed, so that you can reapply them later if necessary, click the [Show Already Unshelved](#)  icon on the toolbar.
3. Press [Ctrl+Shift+U](#) or choose [Unshelve](#) from the context menu of the selection.
4. In the [Unshelve Changes Dialog](#) that opens, specify the changelist you want to restore the unshelved changes to in the Name field. You can select an existing changelist from the drop-down list or type a name for a new changelist to be created containing the unshelved changes. You can enter the description of the new changelist in the Comment field (optional).
If you want to make the new changelist active, select the [Set active](#) option. Otherwise, the current active changelist remains active.
5. If you want PyCharm to preserve the context of a task associated with the new changelist on its deactivation and restore the context then the changelist becomes active, select the [Track context](#) option (see [Managing Tasks and Context](#) for details).
6. If you want to remove the changes you are about to unshelve, select the [Remove successfully applied files from the shelf](#) option. The unshelved files will be removed from this shelf and added to another changelist and marked.0 as applied. They will not be removed completely until deleted explicitly by clicking the [Clean](#) icon  on the toolbar, or selecting [Clean Already Unshelved](#) from the context menu.
7. Click OK. If conflicts occur between the patched version and the current version, resolve them as described in [Resolving Conflicts](#).

You can also unshelve changes silently, without displaying the [Unshelve Changes](#) dialog. To do this, select a file or a changelist you want to unshelve, and click the [Unshelve Silently](#) icon  on the toolbar, or press [Ctrl+Alt+U](#). The unshelved files will be moved to the active pending changelist.

Tip You can also drag-and-drop a file or a changelist from the [Shelf](#) tab to the [Local Changes](#) tab to unshelve it silently.

Restoring unshelved changes

PyCharm lets you reapply unshelved changes if necessary. All unshelved changes can be reused until they are removed explicitly by clicking the [Clean](#) icon  on the toolbar, or selecting [Clean Already Unshelved](#) from the context menu.

To restore applied changes on the shelf do the following:

1. Make sure that the [Show Already Unshelved](#)  toolbar option is enabled.
2. Select the files or the shelf you want to restore.
3. On the context menu of the selection, choose [Restore](#).

Applying external patches

You can import patches created inside or outside PyCharm and apply them as shelved changes.

1. In the [Shelf](#) tab of the [Version Control](#) tool window, choose [Import Patches](#) from the context menu.
2. In the dialog box that opens, select the patch file to apply. The selected patch appears in the [Shelf](#) tab as a shelf.
3. Select the newly added shelf with the patch and choose [Unshelve Changes](#) from the context menu of the selection.

Automatically shelving base revision

If you are using [Git](#) or [Mercurial](#), it may be useful to have PyCharm always shelve base revisions of files that are under version control.

By default, PyCharm always "remembers" the last commit hash. However, this information is not sufficient if the history has been changed since the last commit as a result of running the [rebase](#) operation. In this case, having a copy of the base revision may help.

1. Open the [Settings](#) dialog, and select the [Version Control | Shelf](#) node on the left.
2. Select the [Shelve base revisions of files under distributed version control systems](#) option.

Modifying shelf location

By default, the shelf directory is located under your project directory. However, you may want to change the default shelf location. This can be useful, for example, if you want to avoid deleting shelves accidentally when cleaning up your working copy, or if you want to store them in a separate repository allowing shelves to be shared among your team members.

1. Open the [Settings](#) dialog and select the Version Control | Shelf node on the left.
2. Click the Change Shelves Location button and specify the new location in the dialog that opens.
3. If necessary, select the Move shelves to the new location option to move existing shelves to the new directory.

Updating Local Information

The Update command enables you to synchronize the contents of your local files with the repository. You can invoke this command on:

- [Single or multiple files and directories](#)
- [An entire project](#)

Depending on the updating options, the update procedure may take place silently. If all files are up-to-date, you will be notified about that. Otherwise, the [Update Info tab](#) opens in the [Version Control tool window](#) where you can [group the information](#) as required.

On this page:

- [Updating files and folders](#)
- [Updating a project](#)
- [Grouping update information by packages or changelists](#)

Updating files and folders

1. Select one or more files and folders to be updated in any navigation view (for example, in the [Project tool window](#)).
2. On the main Version Control menu, or on the context menu of the selection, choose <VCS>|Update.
3. In the Update dialog specify the update options, which are different for the supported version control systems, and click OK.

Updating a project

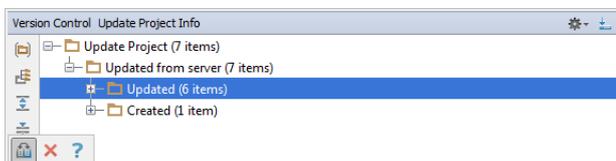
1. Do one of the following:
 - On the main menu, choose VCS | Update project
 - Press `Ctrl+T`.
 - On the main menu, click .
2. In the Update dialog, click the tab for your version control system.
3. Specify the update options, which are different for the supported version control systems and click OK.

Grouping update information by packages or changelists

To group the update information by packages or changelists, use the Group By Package  and Group By Changelist  buttons on the toolbar of the Update Info tab.

Note the difference in the appearance of the tab:

- Grouped by packages



- Grouped by changelists



Grouping by changelists is not available if the project is under Git or Mercurial control.

Using Patches

This section describes how to:

- [Create patches.](#)
- [Apply patches.](#)

Applying Patches

Postponed changes stored in a patch file can be applied to the target file or directory later. If the source code was edited after creating the patch, conflicts may arise. PyCharm suggests a handy way to resolve such conflicts and merge the patch with the changes.

When a patch is opened, PyCharm detects files with the same name as the modified files. For each detected file, PyCharm compares the path to it relative to the base directory, with the path from the patch, and chooses the closest match. If no matching path is found, the file is considered to be located in the project base directory and is highlighted in red.

You can apply changes to files stored in different locations from those specified in the patch by mapping an arbitrary directory as the base one, or stripping off the leading directories.

To apply a patch

1. On the main menu, choose VCS | Apply patch.
2. In the [Apply Patch](#) dialog box that opens, specify the fully qualified name of the patch file. Type the name manually or click the Browse button  and locate the desired patch file in the Select Patch File dialog box.

You can also drag and drop a file or an email attachment to the Apply Patch dialog, and it will be selected automatically.
3. Configure the patch presentation layout. To have changes shown in a flat view, press the Group by Directory toolbar button .

Release the button to have changes shown in a directory tree view.
4. To have a change applied, select the check box next to it.
5. To have a change applied to a modified file that has been moved to another location, specify the new file location.
 - To map another base directory, select the desired file, directory, or a group files/directories and click the Map base directory toolbar button . In the [dialog that opens](#) choose the directory relative to which file names will be interpreted.
 - To remove leading directories from the path, click the Strip Directory toolbar button  as many times as many leading directories you need to strip. The number of removed slashes is indicated in square brackets.
 - To revert the last strip directory action, click the Restore Directory toolbar button . Click the button as many times as many previously stripped leading directories you need to restore.
 - To revert all the strip directory actions in the selection, click the Reset Directories toolbar button .
 - To have all the leading directories stripped and have the changes applied to the file with the specified name in the base directory, click the Remove Directories toolbar button .
6. To view the differences and possible conflicts between your local working copy, the repository version, and the patch in the [Differences Viewer for Files](#), select the desired change and click the Show Differences button .
7. To [resolve conflicts](#) between the patched and the current versions, if any, in both versions select the changes to be merged to the resulting file, and then click Apply.

Note that you apply a patch that contains a lot of files and causes numerous conflicts, you can cancel applying the patch by clicking Abort, and then select whether you want to abort, skip or cancel the remaining conflicts.

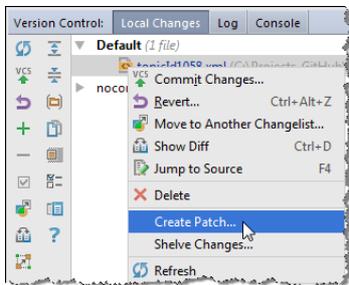
Creating Patches

PyCharm suggests two ways of creating patches:

- On the base of revisions, either local or committed to the repository
- On the base of revisions stored in the local history.

To create a patch file

1. In the [Local Changes](#) tab or the [Repository](#) tab of the Version Control tool window, select a change or changelist you want to create a [patch](#) for.
2. On the main Version Control menu or on the context menu of the selection, choose Create patch.



3. In the [Create Patch](#) dialog box that opens, review the list of changed files, and make sure that the files to be included in the patch are selected.

Tip You can specify the desired changelist immediately in the Create Patch dialog box: click Changelist combo box in the upper right corner, and select the desired changelist.

4. Add a commit comment.

Tip As you type, PyCharm checks the spelling and highlights erroneous words.

This functionality is available if the '[Spelling](#)' [code inspection](#) is enabled.

5. Click Create patch.

You can also create patch on the base of your local history. To do that, open the local history view for the desired directory, file or code fragment, as described in the section [Using Local History](#), right-click the desired revision, and choose the Create Patch command on the context menu, or click the create patch button  on the toolbar.

Viewing Changes Information

This sections explains different ways to keep track of the changes that you and your teammates introduce to the source code.

Reviewing project history

PyCharm allows you to review all changes made to the project sources that match the specified filters.

For distributed version control systems, such as Git and Mercurial, you can view project history in the [Log tab](#) of the [Version Control](#) tool window.

For centralised version control systems, such as Subversion, Perforce, CVS, ClearCase, and TFS, project history is available in the [Repository tab](#) of the [Version Control](#) tool window.

Tracking changes to a file in the editor

As you modify a file that is under version control, all changes are highlighted in the editor with **change markers** that appear in the left gutter next to the modified lines and show the type of changes introduced since the last synchronization with the repository. When you commit the modified file to the repository, the change markers disappear.

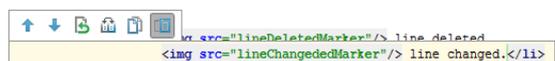
The changes you introduce to the text are color-coded:

-  line added.
-  line changed.

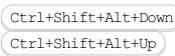
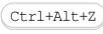
Tip You can customize the default colors for line statuses in [Colors and Fonts](#) settings page.

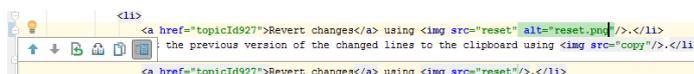
When you delete a line, the following marker appears in the left gutter: .

You can manage changes using the dedicated toolbar. To invoke it, hover the mouse cursor over a change marker and then click it. The toolbar is displayed together with a frame showing the previous contents of the modified line:



You can perform the following operations:

ItemTooltip and Shortcut	Description
 Previous Change / Next Change	 Use these buttons to navigate between changes.
 Rollback	 Click this icon to rollback the changes. Note that all changes to the file since its last revision will be reverted, not just the current line.
 Show Diff	 Click this icon to explore the differences between the current and the repository version of the current line in the Diff for Range dialog.
 Copy	 Click this icon to copy the previous version of the modified line to the clipboard.
 Show Detailed Differences	Toggle this icon to change the way differences to modified lines are presented when you click the line changed change marker. If enabled, the differences are highlighted with the corresponding color:



Comparing local changes with the repository version

Apart from [navigating](#) through your local changes within a file in the editor, you can review these changes compared to the base revision of the file in question.

You can review changes in one of the following ways:

- In the Change Details pane in the [Local Changes](#) tab of the [Version Control](#) tool window. Select a file you want to review in the Local Changes tab and click the Preview Diff  button on the toolbar.
- In the [Differences Viewer](#). To invoke the Differences Viewer do one of the following:
 - Select a file you want to review in the Local Changes tab and press 
 - Click the Show Diff icon  on the toolbar.
 - Right-click a file you want to review and select Show Diff or <your_VCS> | Compare With Latest Repository Version from the context menu.

The left pane shows the affected code as it was in the base revision, and the right page shows the affected code after changes have been made.

Use the toolbar buttons and controls to navigate between changes and configure the appearance of the Change Details pane or the Differences Viewer:

ItemTooltip and Shortcut	Description
 Previous Difference	Use these buttons to jump to the next/previous difference.

/ Next Difference

Shift+F7
F7

When the last/first difference is hit, PyCharm suggests to click the arrow buttons  /  once more and compare other files, depending on the [Go to the next file after reaching last change](#) option in the [Differences Viewer settings](#).

This behavior is supported only when the Differences Viewer is invoked from the [Version Control](#) tool window.



Compare Previous/Next File

Click these buttons to compare the local copy of the previous/next file with its update from the server.

Note These controls are only available if more than one file has been modified locally.

Alt+Left
Alt+Right

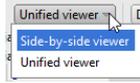


Jump to Source

Click this button to open the selected file in the active pane in the editor. The caret will be placed in the same position as in the Differences Viewer.

F4

Viewer type

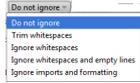


Use this drop-down list to choose the desired viewer type. The side-by-side viewer has two panels; the unified viewer has one panel only.

Both types of viewers enable you to

- Edit code. Note that one can change text **only** in the right-hand part of the default viewer, or, in case of the unified viewer, in the lower ("after") line, i.e. in your local version of the file.
- Perform the Apply/Append/Revert actions.

Whitespaces

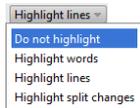


Use this drop-down list to define how the differences viewer should treat white spaces in the text.

- Do not ignore: white spaces are important, and all differences are highlighted. This option is selected by default.
 - Trim whitespaces: (" \t", " ") , if they appear in the end and in the beginning of a line.
 - If two lines differ in trailing whitespaces only, these lines are considered equal.
 - If two lines are different, such trailing whitespaces are not highlighted in the [By word](#) mode.
 - Ignore whitespaces: white spaces are not important, regardless of their location in the source code.
 - Ignore whitespaces and empty lines: the following entities are ignored:
 - all whitespaces (as in the 'Ignore whitespaces' option)
 - all added or removed lines consisting of whitespaces only
 - all changes consisting of splitting or joining lines without changes to non-whitespace parts.

For example, changing `a b c` to `a \n b c` is not highlighted in this mode.

Highlighting mode



Select the way differences granularity is highlighted.

The available options are:

- Highlight words: the modified words are highlighted
- Highlight lines: the modified lines are highlighted
- Highlight split changes: if this option is selected, big changes are split into smaller 'atomic' changes.

For example, `A \n B` vs. `A X \n B X` will be treated as two changes instead of one.

- Do not highlight: if this option is selected, the differences are not highlighted at all. This option is intended for significantly modified files, where highlighting only introduces additional difficulties.



Collapse unchanged fragments

Click this button to collapse all unchanged fragments in both files. The amount of non-collapsible unchanged lines is configurable in the [Diff & Merge](#) settings page.



Synchronize scrolling

Click this button to simultaneously scroll both differences panes; if this button is released, each of the panes can be scrolled independently.



Editor settings

Click this button to invoke the list of available settings. Select or clear this options to show or hide whitespaces, line numbers and indent guides, to use or disable the use of soft wraps, and to set the highlighting level. These commands are also available from the context menu of the differences viewer gutter.



Show diff in external tool

Click this button to invoke an external differences viewer, specified in the [External Diff Tools](#) settings page. This button only appears on the toolbar when the Use external diff tool option is enabled in the [External Diff Tools](#) settings page.



Help

Click this button to show the corresponding help page.

F1

N/A

Annotate

This option is only available from the context menu of the gutter.

Use this option to explore who introduced which changes to the repository version of the file in question, and when. The annotations view lets you see detailed information for each line of code, such as the version from which this line originated, the ID of the user who committed this line, and the commit date.

You can [configure the amount of information displayed in the annotations pane](#).

For more details on annotations, refer to [Viewing Changes Information](#)

The most useful shortcuts are the following:

Shortcut Description

Ctrl+Shift+D

Use this keyboard shortcut to show the popup menu of the most commonly user diff commands.

Ctrl+Tab

Use this keyboard shortcut to switch between the left and the right panes.

Ctrl+Shift+Tab

Use this keyboard shortcut to select the position obtained by **Ctrl+Tab** in the opposite pane.

Ctrl+Z / Ctrl+Shift+Z

Use this keyboard shortcut to undo/redo a merge operation. Conflicts will be kept in sync with the text.

Viewing changes history for a file or selection

PyCharm allows you to review changes made to files or even fragments of source code. The Show History and the Show History for Selection commands are available from the main VCS menu and from the context menu of files.

The change history for a file is displayed in the dedicated [History tab](#) of the [Version Control](#) tool window.

The change history for a selection of code is displayed in a separate window, in the form of the [differences viewer](#).

Viewing the History for a File

Do one of the following:

- Open a file in the editor. Then, on the main VCS menu or on the context menu of the editor tab, choose <VCS> | Show History.
- In the Project tool window, right-click a file and choose <VCS> | Show History from the context menu.

The [History](#) tab for the selected file appears in the Version Control tool window, the name of the file is shown on the title bar of the tab.

You can use the [toolbar](#) buttons to compare the selected revision with the local version, compare classes from the selected revision, checkout the selected revision from your VCS, annotate the selected revision, etc.:

ItemTooltip and Shortcut **Description**

	Compare Ctrl+D	Click this button to compare the selected revision of a file with its previous revision in the Differences Viewer for Files .
	Show Diff with Local	Click this button to compare the selected revision of a file with its local copy in the Differences Viewer for Files .
	Create Patch	Click this button to create a patch from the selected revision.
	Get	Click this button to retrieve the selected revision. If the local copy has already been modified, PyCharm prompts to overwrite the local version, or cancel the operation.
	Annotate	Click this button to open the selected revision of a file in the editor with annotations.
	Show All Affected Files Shift+Alt+A	Click this button to open the Paths Affected in Revision dialog where you can view all files that were modified in the selected revision.
	Copy Revision Number	Click this button to copy the revision number of the commit that the selected file belongs to to the clipboard.
	Compare all classes from revision on UML Ctrl+Shift+D	Click this button to view all classes of the selected revision as a UML Class diagram. See section Viewing Changes as Diagram .
	Open in GitHub	Click this button to open the page that corresponds to the selected commit on GitHub .
	Show All Branches	Click this button to display changes from branches other than the current one.
	Show Branches	Note This option is only available if you are using Perforce for version control. Click this button to show branches.
	Show All Revisions Submitted In Selected Changelist	Note This option is only available if you are using Perforce for version control. Click this button to display the list of all revisions committed in the same changelist as the selected revision of a file.
	Refresh	Click this button to refresh the current information.
	Show Details	Click this button to show the commit message for the selected revision.
	Close Ctrl+Shift+F4	Click this button to close the current history tab.

Viewing the History for a Selection

1. In the editor, select a fragment of the source code.
2. Choose <VCS> | Show History for Selection from the main VCS menu, or on the context menu of the selection.

The history for the selected fragment will open in a separate window.

Checking file status

PyCharm allows you to check the status of project files relative to the repository. File status shows you which operations have been performed on the file in question since you last synchronized with the repository.

You can check the status of a file in any interface element (e.g. the editor, or various tool windows) by the color used to highlight the file name.

Tip You can customize the default colors for file statuses in [Colors and Fonts](#) settings page.

Color	File Status	Description
Black	Up to date	File is unchanged. 
Gray	Deleted	File is scheduled for deletion from the repository. 
Blue	Modified	File has changed since the last synchronization. 
Green	Added	File is scheduled for addition to the repository. 
Violet	Merged	File is merged by your VCS as a result of an update. 
Brown	Unversioned	File exists locally, but is not in the repository, and is not scheduled for adding. 
Olive	Ignored	File will be ignored in any VCS operation. 
Light brown	Hijacked	File is modified without checkout . This status is valid for the files under Perforce, ClearCase and VSS. 
Red	Merged with conflicts	During the last update, file was merged with conflicts. 
Lilac	Externally deleted	File is deleted locally, but was not scheduled for deletion, and still exists in the CVS repository. 
Dark cyan	Switched	The file is taken from a different branch than the whole project. This status is valid for CVS and SVN. 

Using annotations

What are VCS annotations?

Annotation is a form of file presentation that shows detailed information for each line of code. In particular, for each line you can see the version from which this line originated, the user ID of the person who committed this line, and the commit date. The annotated view helps you find out who did what and when, and trace back the changes.

Annotating lines of code is available for ClearCase, TFS, Mercurial, Git, CVS, Perforce and Subversion.

The Annotate command is available from VCS-specific nodes of the Version Control menu, the context menu of the Editor left gutter, file context menus, and the [file history](#) view.

When annotations are enabled, the left gutter looks similar to the following example:

```

20 d3c36cc1 5/19/2015 cheptsov
21 b338ae7e 11/14/2016 Megorskaya
22 b338ae7e 11/14/2016 Megorskaya
23 b338ae7e 11/14/2016 Megorskaya
24 b338ae7e 11/14/2016 Megorskaya
25 f224ebf3 5/17/2016 Megorskaya
26 f224ebf3 5/17/2016 Megorskaya
27 d3c36cc1 5/19/2015 cheptsov
28 343931aa 11/14/2016 Megorskaya
29 01ed7f9f 1/13/2017 RayShade
30 d3c36cc1 5/19/2015 cheptsov
31 01ed7f9f 1/13/2017 RayShade
32 343931aa 11/14/2016 Megorskaya
33 d3c36cc1 5/19/2015 cheptsov
34 343931aa 11/14/2016 Megorskaya
35 01ed7f9f 1/13/2017 RayShade
36 343931aa 11/14/2016 Megorskaya
37 343931aa 11/14/2016 Megorskaya
38 28824a5c 10/14/2016 RayShade
39 28824a5c 10/14/2016 RayShade

```

Tip Annotations for lines modified in the current revision, are marked with bold type and an asterisk.

Configuring the amount of information shown in the annotations pane

1. Enable annotations and right-click the annotations gutter.
2. Select View in the context menu and select or deselect the following options:
 - Revision: select this option if you want to see the number of the changelist within which the annotated changes were checked in.
 - Date: select this option if you want to see the date when the annotated changes were checked in.
 - Author: select this option if you want to see the name of the user who checked in the annotated changes.
 - Commit number: select this option if you want to see the revision number of the current file.
 - Colors: use this control to toggle between the following highlighting modes:
 - Author: select this option if you want to highlight changes made by different authors with different colors.
 - Order: select this option if you want annotation colors to indicate how long ago a change was made. The entire file history is divided into several time periods containing an equal number of commits, and each time period is assigned its own color. The most recent changes are highlighted in green, while the oldest changes are highlighted in red:

```
36 2/8/2017 Megorskaya *
37 10/21/2016 Megorskaya
38 2/8/2017 Megorskaya *
39 5/19/2015 cheptsov
40 5/19/2015 cheptsov
41 2/8/2017 Megorskaya *
42 2/8/2017 Megorskaya *
43 2/8/2017 Megorskaya *
44 2/8/2017 Megorskaya *
45 2/8/2017 Megorskaya *
```

- Hide: select this option if you do not want to use color highlighting. In this case, all annotations will be displayed in gray.
 - Names: use this control to select how user names will be displayed. The following options are available:
 - Last name
 - First name
 - Full name
3. To view a commit message for an annotated change, hover the mouse cursor over an annotation. A tooltip will appear showing the commit message for the corresponding change:

```
1/25/2016 Megorskaya <14>To view
5/19/2015 commit c0f815fcec1a998c55a38eb53fee263f4698709e
10/14/2016 Author: Irina.Megorskaya
1/25/2016 Date: 1/25/2016 12:39 PM
1/25/2016
5/19/2015 https://youtrack.jetbrains.com/issue/IDEA-147758
```

Note The amount of information displayed in the tooltip depends on the version control system you are using and is not affected by the [annotation settings](#).

Annotating previous revisions

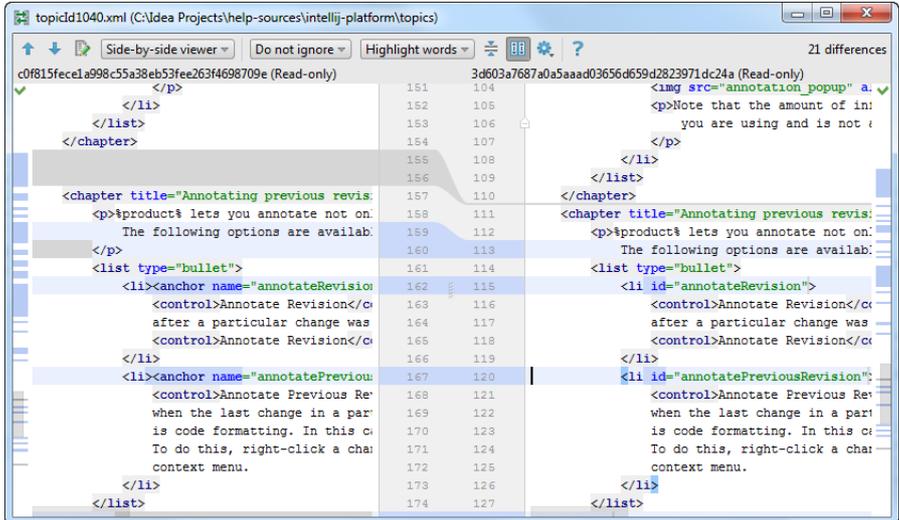
PyCharm lets you annotate not only the current file revision, but also its previous revisions. The following options are available from the context menu of the annotations gutter:

- Annotate Revision: this option is useful if you want to check what a file looked like after a particular change was committed. To do this, right-click this change and select Annotate Revision from the context menu.
- Annotate Previous Revision: this option is useful if you find yourself in a situation when the last change in a particular line is meaningless, for example if all that was changed is code formatting. In this case, you can check what the previous revision of the file looked like. To do this, right-click a change and select Annotate Previous Revision from the context menu.

You can also annotate a particular file from the [file history](#) view. In the History tab, select the file version you want to review, right-click the corresponding line and select Annotate from the context menu.

Viewing the differences between revisions

To review the differences between the annotated version of a file and its previous version, position the cursor on an annotation, right-click it and select Show Diff from the context menu. PyCharm opens the [Differences Viewer for Files](#):



Navigating to log

If you are using Git for version control, you can also jump from the annotations view to the corresponding commit in the [Log tab](#) of the [Version Control](#) tool window.

To do this, position the cursor on an annotation, right-click it and select [Select in Git log](#) from the context menu. You can also use the [Copy revision number](#) command to locate a revision in the log.

For projects hosted on <https://github.com/>, the [Open on GitHub](#) command is also available that takes you to the corresponding commit.

VCS-Specific Procedures

In most cases, PyCharm provides a unified approach to version control operations, as described in the previous sections. Nevertheless, there are certain VCS-specific features and peculiarities that the user should be aware of. Find helpful tips and notes in the following sections:

- [Accessing the Authentication to Server Dialog](#)
- [Using CVS Integration](#)
- [Using Git Integration](#)
- [Using Mercurial Integration](#)
- [Using Perforce Integration](#)
- [Using Subversion Integration](#)
- [Using GitHub Integration](#)

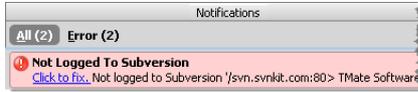
Accessing the Authentication to Server Dialog

For a number of reasons of various nature you may be not authenticated to the remote server while the authentication dialog box does not appear, as you might expect. Instead, PyCharm displays the corresponding message in a pop-up window in the bottom-left corner of the editor.

Tip The authentication dialog appears immediately only when you attempt to access a URL address outside your working copies. This feature applies to Perforce, Subversion, and CVS.

To access the authentication dialog box

1. Click the Notifications Pending button  on the Status bar.
2. In the Notifications pop-up window that opens, click the link Click to fix.



3. In the authentication dialog box that opens, specify your credentials.

Using CVS Integration

With the CVS integration enabled, you can perform basic CVS operations from inside PyCharm.

Note The information provided in the topics listed below assumes that you are familiar with the basics of CVS version control system.

In this section:

- Using CVS Integration
 - [Prerequisites](#)
 - [CVS support](#)
- [Browsing CVS Repository](#)
- [Checking Out Files from CVS Repository](#)
- [Configuring CVS Roots](#)
- [Configuring Global CVS Settings](#)
- [Ignoring Files](#)
- [Importing a Local Directory to CVS Repository](#)
- [Resetting Branch Head to a Selected Commit](#)
- [Resolving Commit Errors](#)
- [Updating Local Information in CVS](#)
- [Using CVS Watches](#)
- [Working Offline](#)
- [Working with Tags and Branches](#)

Prerequisites

- PyCharm comes bundled with the CVS plugin. This plugin is turned on by default. If it is not, make sure that the plugin is enabled.
- PyCharm CVS integration does not require a standalone CVS client. All you need is an account in your CVS repository.
- CVS [integration is enabled](#) for the current project root or directory.

CVS support

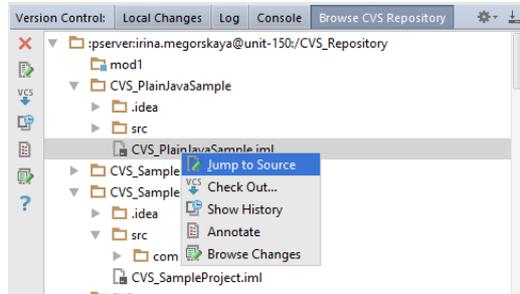
- When CVS integration with PyCharm is enabled, the CVS item appears on the VCS menu, and on the context menus of the [Editor](#) and [Project](#) views.
- The files in the folders under the CVS control are highlighted according to their status. See [File Status Highlights](#) for file status highlighting conventions.
- Modifications results are shown in the [Version Control tool window](#).
- When using CVS integration, it is helpful to open the [Version Control](#) tool window. The [Console](#) tab displays the following data:
 - All commands generated based on the settings you specify through the PyCharm user interface.
 - Information messages concerning the results of executing generated CVS commands.
 - Error messages.

Browsing CVS Repository

You can browse any CVS repository and modify the structure of the currently open project, or a different one. Browsing contents of a repository is always available, even when CVS is not enabled in project. All you need is a valid user account.

To browse the CVS repository and modify its structure

1. Open a project. Then, choose VCS | Browse VCS Repository | Browse CVS Repository... on the main menu. The Select CVS Root Configuration dialog is opened.
2. Select a root from the list of configured CVS roots, or click Configure to specify a new one, and then click OK. The Browse CVS Repository tab opens in the CVS tool window at the bottom of the Editor.
3. Browse the desired CVS repository and perform jump to source, checkout, browse changes and annotate operations for files and folders.



Icons for the tree nodes denote their respective types. For example:

-  means a CVS directory.
-  denotes a CVS module.

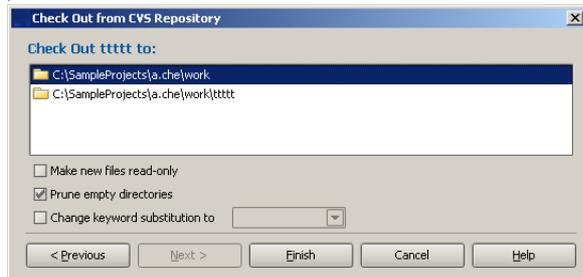
Icons appearing next to the files will denote the corresponding file types.

Checking Out Files from CVS Repository

Checking out action helps you obtain a writable copy of the repository, which you can edit as required. After making the necessary changes, you can publish results by committing, or checking in, to the repository. This section describes CVS-specific checkout procedure.

To check out files from a CVS repository

1. On the main menu, choose VCS | Checkout from Version Control.
2. On the submenu, choose CVS.
3. In the [Checkout From CVS Repository dialog](#), select the desired CVS configuration, and click Next. If the necessary CVS configuration is not available, click Configure [to create a new one](#).
4. Select the elements to check out, and click Next.
5. Specify the checkout destination directory where the local copy of the repository files will be created and click Next.
If you are checking out sources for an existing project, the destination folder should be below a project [content root](#).
6. If you have selected an element of a CVS module to check out, the last page of the wizard suggests you to add the module name to the local path:



7. Set up CVS checkout options, or accept defaults, and click Finish.
8. PyCharm suggests to create a project based on the sources, checked out from version control.
If you accept the suggestion, open new project, as described in the section [Opening Multiple Projects](#).

Configuring CVS Roots

CVS roots are global PyCharm properties. Once configured, a CVS root is available regardless of which project is currently opened. This is helpful if you need to check out an entire project from CVS.

You can define multiple CVS root configurations for future use and edit them whenever necessary. Alternatively, you can configure CVS roots when [checking out](#) files or directories from or [importing](#) them to a CVS repository.

From this section you will learn how to:

- Configure a [new CVS root](#)
- [Modify](#) a CVS root
- Configure a CVS root [based on an existing](#) configuration
- [Remove](#) a CVS root configuration

To configure a CVS root, follow these general steps:

1. [Open the CVS Roots](#) dialog box.
2. Click the Add button  on the toolbar.
3. Specify the [CVS root string](#).
4. Specify the [version to work](#) with.
5. Specify [additional connection settings](#).
6. Click the Test connection button to check that the specified settings ensure establishing successful connection to the CVS server.
7. Click OK to apply the specified settings and close the dialog box.

To modify an existing CVS root configuration

1. [Open the CVS Roots](#) dialog box.
2. Select the CVS root to modify.
3. Edit the settings as while [configuring a new](#) CVS root.
Changes made here apply to the current CVS root configuration only.

To configure a new CVS root based on an existing configuration

1. [Open the CVS Roots](#) dialog box.
2. Select the CVS root to be used as the basis for a new configuration.
3. Click the Copy button  on the toolbar or press `Ctrl+O`. The selected configuration is copied as a new CVS root.
4. [Edit](#) the newly created CVS root configuration, as required.

To remove a CVS root configuration

1. [Open the CVS Roots](#) dialog box.
2. Select the CVS root you want to remove and click the Remove button  on the toolbar.

You can access the CVS Roots dialog box in several ways, depending on the general task you are currently performing.

To open the CVS Roots dialog box, do one of the following:

- When **checking out** files or directories from a CVS repository: on the main menu, choose VCS | Checkout from Version Control | CVS and in the dialog box that opens click the Configure button.
- When **importing** files or directories to a CVS repository: on the main menu, choose VCS | Import into CVS and in the dialog box that opens click the Configure button.
- When defining a CVS root to use in the future: open a file which is already under CVS control and choose VCS | CVS | Configure CVS Roots on the main menu.

Assembling a CVS Root String

CVS root strings are specified in the CVS Root text box of the [CVS Roots](#) dialog box.

The CVS Root string syntax is:

```
[ :method: ] [ [user] [ :password ] @ ] hostname [ :port ] / path / to / repository .
```

You can obtain the valid string from your system administrator, or assemble the CVS root parameters into a correct string manually, or use the Edit by Field functionality, as described below.

To assemble the CVS Root parameters

1. In the [CVS Roots](#) dialog box, click the Edit by Field button next to the CVS Root text box. The [Configure CVS Root Field by Field](#) dialog box opens.
2. Choose the connection method and specify the user name, port, host, and repository. See the [Configure CVS Root Field by Field](#) dialog box reference for details.
3. Click OK. The dialog box closes and you return to the [CVS Roots](#) dialog box. The CVS Root text box displays the specified parameters assembled in a valid string.

Specifying a Version to Work With

By default, PyCharm suggests you to check out the latest (HEAD) revision to work with. However, you can synchronize your local working copy with any previous revision from the repository.

Tip Make sure that the CVS root field is filled in with valid data.

To specify the version to work with

1. In the Use Version section of the [CVS Roots](#) dialog box, select the criterion to search for the desired version. The available options are:
 - By tag
 - By date
2. Click the Browse button  next to the selected option and do one of the following depending on the selected search criterion:
 - If you have selected By tag, select the desired tag from the list of tags that opens. The list displays the tags obtained from the CVS server according to the specified CVS root string.
 - If you have selected By date, select the date and time in the calendar that opens.

You can also type the desired revision number, tag, or date in the text box next to the selected option.

Specifying Additional Connection Settings

You can flexibly configure connection to the CVS server using additional settings. The set of relevant options depends on the [connection method](#) specified in the CVS root string. The available additional options for each of the supported connection methods are displayed in the lower part of the CVS Roots dialog box. See the [CVS Roots](#) dialog box reference for details.

For a module associated with CVS you can specify global CVS settings, which includes character set, location of the password file, connection timeout etc.

To configure CVS global settings for a directory associated with CVS

1. On the main menu, choose VCS | CVS | Global Settings.
2. In the [Global CVS Settings](#) dialog box that opens, specify the global settings and click OK.

Tip Alternatively, you can define global CVS settings when [configuring a CVS root](#). To invoke the [Global CVS Settings](#) dialog box, click the Global Settings button in the [CVS Roots](#) dialog box.

Ignoring Files

If you want your CVS integration to ignore certain unversioned files under CVS-associated directories, and skip them when performing update, import etc., add these files to the CVS ignore list, which is stored in the `.cvsignore` file.

The way CVS integration handles unversioned files depends on the [general settings for file creation](#). If the new files, created with PyCharm, are not put under version control automatically, you can add them to the ignore list using CVS command. You can put the `.cvsignore` file under CVS version control, and it will be recognized by all CVS clients.

To include an unversioned file to the ignore list

1. Select an unversioned file under a CVS-associated directory (by default, such files are brown).
2. On the main VCS menu, or on the context menu of the selection, choose CVS | Ignore. If `.cvsignore` file is not under CVS version control, proceed to the next step. If `.cvsignore` already exists and is under version control, PyCharm adds the file in question to the ignore list silently.
3. If you want to put the ignore list under version control, in the Add File `.cvsignore` to CVS dialog box, click Add to CVS and optionally specify the desired [keyword substitution](#).

Tip If you want to remove a file from the ignore list, you can only do it manually. Open the `.cvsignore` file for editing by pressing `F4`, and remove the lines for the files that should not be ignored by CVS.

Importing a Local Directory to CVS Repository

You can import an entire directory to your CVS repository, provided that you have the corresponding access rights. This action is helpful for putting a whole project under version control.

Import to the repository is always available, even when CVS is not enabled in project.

To import a directory into CVS repository

1. On the main menu, choose VCS | Import into CVS.
2. On the first page of the [Import into CVS](#) wizard, select the target CVS root. If the desired target repository does not exist, you can create a new one. To do that, click Configure, and [define the desired CVS root](#). Click Next.
3. On the second page of the wizard, select the target directory in the repository, and click Next.
4. On the third page of the wizard, select the local directory that will be imported into the CVS repository. Click Next.
5. In the Customize Keyword Substitution page, specify the [keyword substitution rule](#) for the files imported into the repository, and click Next.
6. Specify the required [import settings](#). The fields of this page correspond to CVS command-line arguments for *import*, and additional options that define the checkout status of the imported directory.
7. Click Finish.

PyCharm allows you to reset the current Git branch head to a selected commit and update the working tree and the index.

To reset the current branch head to a selected commit

1. In the main menu navigate to View | Tool Windows | Changes.
2. In the [Version Control tool window](#) switch to the [Log tab](#).
3. Select a commit that you want to reset the current branch head to, and right-click this commit. Select Reset Current Branch to Here from the popup menu.
4. In the [Git Reset](#) dialog that opens, select the mode for updating the working tree and the index. The following options are available:
 - Soft: The modified files will not be changed, and the differences will be staged for commit.
 - Mixed: The modified files will not be changed, and the differences will be unstaged. This mode is used by default and is identical to the `git reset` command.
 - Hard: The modified files will be reverted to the selected commit, and all uncommitted changes will be lost.
 - Keep: The modified files will be reverted to the selected commit, but local changes will be preserved.

Resolving Commit Errors

In the section [Checking In Files](#), you have learnt how to check in (commit) your changes to the repository. In this section you can find examples of CVS-specific error messages and suggestions on resolving conflicts.

If any error occurs when trying to commit, PyCharm displays an error message. For example:

- If you have a modified file, which has been already changed on the server by someone else, since your last synchronization, you will get the following error:

```
Error: cvs server: Up-to-date check failed for 'source/com/...'
```

In this case you would need to [merge](#) your local copy with the current revision in the repository. When the copies are merged, and all possible conflicts are [resolved](#), so that the file is assigned the merged status, you can safely commit it to the repository.

- If you try to commit a file marked with a sticky tag, or sticky date, the CVS server will detect an attempt to *change the past*, and the error looks as follows:

```
Error: cvs server: sticky tag '1.1' for file 'source/com/impl/ManagerImpl.java' is not a branch
```

To solve the problem, you need to update with resetting sticky data; in this case your changes will be merged with the most recent revision of the file. After resolving possible conflicts (by calling the Merge command) you will be able to commit the files.

Updating Local Information in CVS

From the [Updating Local Information](#) topic, you have learnt the general procedure. CVS integration provides a special [Update File / Directory](#) dialog box with the options that map directly to the corresponding CVS command-line options of the `update` command. Refer to the [CVS documentation](#) for details.

This section will consider the CVS-specific procedure and several of the options in terms of their presentation in PyCharm.

To update local information

1. Select one or more files and/or directories in any navigation view (for example, [Project tool window](#)).
2. On the main menu, select `VCS | Update Project` or on the context menu of the selection choose `CVS | Update File (Directory)`. The Update dialog for a project or file opens. In case of project update, select `CVS` tab.
3. Specify the following options:

Branch Merging

You can choose to merge your local file(s) with the counterpart(s) in one or two CVS branches. In the CVS command-line interface, this is the `-j` option.

The option *Don't Merge* is selected by default, as merging across branches is not commonly needed. If you choose one of the other options, one or both of the text fields are enabled (depending on the choice of options). Clicking the Browse button  next to each field opens the Select Tag dialog box which lists all the branch tags maintained by the repository on the CVS server. Locate the branch you want to merge with, select it in the list, and click OK.

Use Version

You can optionally update your local system from some different revision. You can choose a revision by its tag or by its date. The Default option synchronizes with your file's current revision, as this is the most common synchronization. The other options for Use Version are:

- By tag (-r): When updating a single file, you can choose the revision either by Revision or Tag. When you choose this option, the text field and corresponding ellipsis button are enabled. Enter a revision number or tag in the text field, or click the ellipsis button and select either a revision or a tag in the resulting dialog.

When updating multiple files (selected individually, or when invoking update on a directory), you can only select the revision by Tag.

- By date (-d): You can update from the revision of a specific date. This is possible whether you are invoking update on one file or multiple files or directories. When you choose this option, the date defaults to the current date. To specify a different date, click the ellipsis button and specify the desired date in the Choose Date dialog which appears.

Reset sticky data (-A)

If the last checkout or update of the selected file(s) was from some revision that was specified by tag or date, the tag/date information is *sticky* for the file(s). If you now want to update these sticky-tagged files from the *HEAD* revision, select this option in combination with selecting the Default option in *Use Version* so that the sticky information is removed.

Change keyword substitution to

If checked, this is converted to the `-k` CVS parameter. Refer to [CVS Options: Default keyword substitution for text files](#) for details.

Do not show this dialog in the future

Select this option if you want the update operation take place silently. For details see [CVS Options](#).

To have PyCharm show this dialog box before update again:

1. Open the [Version Control - Confirmation](#) page of the [Settings](#) dialog box.
2. In the Display Option dialogs when these commands are invoked area, select the Update check box.

Using CVS Watches

CVS integration of PyCharm helps coordinate the activities of team members, who concurrently work on the same files or directories.

CVS watches enable you to notify the users of a CVS repository whenever a file has been opened for editing, or committed. If you watch a file or directory for changes and commits, you are added to the list of watchers.

Usage the commands related to watches depends on configuration of the `CVSROOT/notify` file.

Edit and Unedit commands change read-only status of the files or directories under CVS. If the sources were checked out with the option `-c`, you can apply Edit command to make them writable. In this case you are added to the list of editors. When you are done with editing, use Unedit command to restore read-only status. So doing, you are removed from the list of editors. If watching is configured, the watchers will receive email notification about these events.

In fact, the option Use read-only flag for not edited files in the CVS provides the same functionality automatically. If this option is checked, Unedit always applies to the source files after commit.

Edit and Watch commands apply to all the files you have selected in the current view (including all the files in any selected directory), or to the current file in the editor if you invoked the command there.

This section describes how to:

- [Access Edit and Watch commands](#)
- [Change read-only status of a file or directory](#)
- [View the other persons who edit the same file or directory](#)
- [Set watch on a certain event for a file or directory and thus add yourself to the list of watchers](#)
- [Enable or disable watching](#)
- [View the other persons who watch the same file or directory](#)

To access Edit and Watch commands

1. In one of the tool windows, select the desired files or directories, or open a file in the editor.
2. Do one of the following:
 - On the main menu, choose VCS | CVS | Edit and Watch
 - On the context menu, choose CVS | Edit and Watch

To get write access to a file or directory

1. [Open Edit and Watch menu.](#)
2. Choose Edit on the submenu. Edit Options dialog box is displayed.
3. If you want to gain exclusive write access, check the option Reserved edit (-c). Click OK.

To restore read-only status of a file or directory

1. [Open Edit and Watch menu.](#)
2. Choose Unedit on the submenu.

To view the other persons who edit the same file or directory

1. [Open Edit and Watch menu.](#)
2. Choose Show Editors on the submenu. This will display the list of all users who have run the Edit command on the same file or directory.

To set watch on a file or directory

1. [Open Edit and Watch menu.](#)
2. Choose Add Watch on the submenu.
3. In the dialog box that opens, select the type of action you would like to be notified about:
 - Edit: you will notified whenever Edit is applied to a watched file or directory.
 - Unedit: you will notified whenever Unedit is applied to a watched file or directory.
 - Commit: you will notified whenever Commit is applied to a watched file or directory.
 - All: you will notified whenever any of the above commands is applied to a watched file or directory.

To remove watch from a file or directory

1. [Open Edit and Watch menu.](#)
2. Choose Remove Watch on the submenu.
3. In the dialog box that opens, select the type of action for which you would like to skip notification (Edit, Unedit, Commit or All).

To suspend or resume watching

1. [Open Edit and Watch menu.](#)
2. Choose Watch Off or Watch On on the submenu.

To view the list of users who are watching the same files or directories

1. [Open Edit and Watch menu.](#)
2. Choose Show Watchers on the submenu.

Working Offline

Offline mode in CVS makes it possible to ignore network errors. When this mode is enabled, you will not receive any notifications about connection problems.

You can go to offline mode in two ways

- When a connection error occurs, PyCharm informs you about that. With the first successful transaction, offline modes automatically turns off.
- Using the CVS menu command on the main Version Control menu, or a context menu: VCS | CVS | Work Offline.

Working with Tags and Branches

From within PyCharm, you can create and delete CVS tags and branches. The names of the tags and branches must start with a letter, and contain only alphanumeric characters.

Branch and tag commands apply to all the files you have selected in the current view (including all the files in any selected directory), or to the current file in the editor if you invoked the command there.

This section describes how to:

- [Access the tags and branches commands.](#)
- [Create a branch](#) in the repository on the base of the current revision.
- [Tag a revision in you local working directory.](#)
- [Delete a tag.](#)

To access tags and branches commands

1. In one of the tool windows, select the desired files or directories, or open a file in the editor.
2. Do one of the following:
 - On the main menu, choose VCS | CVS
 - On the context menu, choose CVS
3. On the submenu, choose the appropriate command (Create Branch, Create Tag, or Delete Tag)

To create a branch

1. [Invoke](#) the Create Branch command.
2. In the Create Branch dialog box, specify the name of the new branch. To do that, type the name in the Branch name field, or click the Browse button  and select the desired name from the list of existing CVS tags.
3. Optionally, specify the following:
 - Select the Override existing check box, if you want to move an existing tag to a new branch, as defined by the option `-F` of `rtag` CVS command.
 - Select the Switch to this branch check box to switch your local working copy to the branch specified in the Branch name field.

To create a new tag

1. [Invoke](#) the Create Tag command.
2. In the Create Tag dialog box, specify the new tag name. To do that, type the name in the Tag name field, or click the Browse button  and select the desired name from the list of CVS tags.
3. Optionally, specify the following:
 - Select the Override existing check box, if you want an existing tag to point to the current revision, as defined by the option `-F` of `rtag` CVS command.
 - Select the Switch to this tag check box to switch your local working copy to the tag specified in the Tag name field.

To delete a tag

1. [Invoke](#) the Delete Tag command.
2. In the Delete Tag dialog box, specify the name of the tag you want to delete. To do that, type the name in the Tag name field, or click the Browse button  and select tag name from the list of the current tags in the repository.

Using Git Integration

With the Git integration enabled, you can perform basic Git operations from inside PyCharm.

Note The information provided in the topics listed below assumes that you are familiar with the basics of Git version control system.

In this section:

- Using Git Integration
 - [Prerequisites](#)
 - [Git support](#)
- [Setting Up a Local Git Repository](#)
- [Adding Files to a Local Git Repository](#)
- [Adding Tags](#)
- [Committing Changes to a Local Git Repository](#)
- [Managing Remotes](#)
- [Handling Passwords for Git Remote Repositories](#)
- [Fetching Changes from a Remote Git Repository](#)
- [Pulling Changes from the Upstream \(Git Pull\)](#)
- [Pushing Changes to the Upstream \(Git Push\)](#)
- [Checking Git Project Status](#)
- [Managing Branches](#)
- [Stashing and Unstashing Changes](#)
- [Handling LF and CRLF Line Endings](#)

Prerequisites

- [Git](#) is installed on your computer.
 - It is strongly recommended that you use version 1.7.1.1 or higher.
- The location of the Git executable file is correctly specified on the [Git](#) page of the Settings dialog box.
- Git [integration is enabled](#) for the current project root or directory.
- If you are going to use a remote repository, create a Git hosting account first. You can access the remote repository through the username/password and keyboard interactive authentication methods supported by the Git integration or through a pair of `ssh` keys.
 - Please note the following:
 1. `ssh` keys are generated outside PyCharm. You can follow the instructions from <https://help.github.com/articles/generating-ssh-keys/> or look for other guidelines.
 2. Store the `ssh` keys in the `home_directory \.ssh\` folder. The location of the `home_directory` is defined through [environmental variables](#):
 - `HOME` for [Unix-like](#) operating systems.
 - `userprofile` for the Microsoft Windows operating system.
 3. Make sure, the keys are stored in files with correct names:
 - `id_rsa` for the private key.
 - `id_rsa.pub` for the public key.
 4. PyCharm supports a standard method of using multiple `ssh` keys, by means of creating `.ssh/config` file.

Git support

- When Git integration with PyCharm is enabled, the Git item appears on the VCS menu, and on the context menus of the [Editor](#) and [Project](#) views.
- The files in the folders under the Git control are highlighted according to their status. See [File Status Highlights](#) for file status highlighting conventions.
- Modifications results are shown in the [Version Control tool window](#).
- When using Git integration, it is helpful to open the [Version Control](#) tool window. The [Console](#) tab displays the following data:
 - All commands generated based on the settings you specify through the PyCharm user interface.
 - Information messages concerning the results of executing generated Git commands.
 - Error messages.

Setting Up a Local Git Repository

Although Git provides high flexibility in arranging data and your work with repositories, the following scenarios are most commonly used for setting up a local Git repository:

- [Clone](#) an existing remote repository and create a new project with the downloaded data.
- [Create a local repository](#) which you can [push](#) to a remote location later, if necessary.

Warning! Git does not support external paths. So if you choose another directory, note that it must contain the tree where the project root resides.

To clone a remote Git repository

1. On the main menu, choose VCS | Checkout from Version Control | Git. The [Clone Repository](#) dialog box opens.
2. In the Git Repository URL text box, type the URL of the remote repository which you want to clone.
3. Click the Test button next to the Git Repository URL text box to check that connection to the remote repository can be established successfully.
4. In the Parent Directory text box, specify the directory where you want PyCharm to create a folder for your local Git repository. Use , if necessary.
5. In the Directory Name text box, specify the name of the new folder into which the repository will be cloned. Click Clone.
6. Create a new project based on the cloned data by accepting the corresponding suggestion displayed by PyCharm.

Git root mapping will be automatically set to the project root directory.

To create a local Git repository

1. Open the project you want to store in a repository.
2. On the main menu, choose VCS | Import into Version Control | Create Git Repository.
3. In the [dialog that opens](#), specify the directory where you want to create a new Git repository.
4. Put the required [files under Git](#) version control. The files appear in the Local Changes tab of the [Version Control](#) tool window, under the Default changelist.

Tip By default, PyCharm suggests the root of the current project.

- If you specify Git as the version control system for a directory in the [Version Control](#) dialog box, PyCharm will suggest to put each new file in this directory under Git control.

Adding Files to a Local Git Repository

On this page:

- [Basics](#)
- [Adding all currently unversioned files to Git control](#)
- [Adding specific files to a local Git repository](#)

Basics

After a Git repository for a project is [initialized](#), you need to add the project data to it.

If you have specified Git as the version control system for your project in the Settings | [Version Control](#) page, PyCharm suggests to put each newly created file under Git version control.

To make Git ignore some types of files, [configure files to ignore](#).

Adding all currently unversioned files to Git control

1. Switch to the [Version Control](#) tool window and switch to the [Local Changes](#) tab. tool window.
2. Expand the Unversioned Files node and choose Add to VCS from the context menu or press `Ctrl+Alt+A`.

Adding specific files to a local Git repository

Do one of the following:

- Switch to the [Version Control](#) tool window and switch to the [Local Changes](#) tab. Expand the Unversioned Files node, and select the files to be added. From the context menu, choose Add to VCS, or press `Ctrl+Alt+A`.
- Switch to the [Project](#) tool window and select the files to be added. From the context menu, choose Git | Add or press `Ctrl+Alt+A`.

Adding Tags

On this page:

- [Basics](#)
- [Assigning tags to commits](#)
- [Creating annotated tags](#)

Basics

The Git integration supports [tagging](#) a particular commit or object to refer to it in the future. Basically, by adding a tag you create a branch that never moves.

You can also create [annotated tags](#) and have annotations displayed to facilitate viewing and navigating through tagged commits.

Assigning tags to commits

To assign a tag to a commit

1. On the main menu, choose VCS | Git | Tag Files. The [Tag](#) dialog box opens.
2. From the Git Root drop-down list, select the required local repository.
The Current Branch read-only field shows the branch you are currently working with in the selected repository. The contents of the read-only field change as you change the selection in the Git Root drop-down list.
3. In the Tag Name text box, type the name of the new tag.

Tip If you specify a name that already exists, PyCharm displays an error message. To override the error and re-assign the tag, select the Force check box.
4. In the Commit text box, specify the commit or object which you want to tag:
 - To tag the latest commit in the branch (HEAD), leave the text box empty.
 - To tag a particular commit, specify its commit hash or use an expression, for example, of the following structure: `<branch>~<number of commits backwards between the latest commit (HEAD) and the required commit>`.
Refer to the Git [commit naming](#) conventions for details.
5. To check that the specified commit exists and is the one you actually need, click the Validate button. The Paths affected in commit dialog box shows the files that were affected in the specified commit. View the information and click OK.
If you specify a commit that does not exist, PyCharm displays an error message.

Creating annotated tags

To create an annotated tag

1. [Create](#) a tag.
2. In the Message text box, provide a description of the commit.

Committing Changes to a Local Git Repository

This topic describes how to commit changes to a [local Git repository](#). For information on uploading changes to a remote repository, see [Pushing Changes to the Upstream \(Git Push\)](#).

To commit changes to a local repository, do the following:

1. Open the [Version Control](#) tool window by pressing `Alt+9` or choosing View | Tool Windows | Version Control.
2. In the [Local Changes](#) tab, select the files or folders you want to commit.
3. Invoke the [Commit Changes](#) dialog box by doing one of the following:
 - On the tool window toolbar, click the Commit Changes button .
 - Select Commit Changes from the context menu of the selected file or changelist.
 - On the main menu, choose VCS | Commit Changes or VCS | Git | Commit File.
 - Press `Ctrl+K`.

The [Commit Changes](#) dialog lists all files that have been modified since the last commit, and all newly added unversioned files.

Note You can invoke the Commit Changes dialog even if you added new files that have not been put under version control yet from outside PyCharm.

4. Select the checkboxes next to the files you want to commit.
5. Enter a commit message and select the [before commit](#) actions you want PyCharm to perform before committing the selected files to the local repository.
6. Select the following options in the Git section if necessary:
 - Author: if you are committing changes made by another person, you can specify the author of these changes.

Tip Press `Ctrl+Space` to invoke a list of contributors to choose from.

- Amend commit: select this option if you want to replace an already published commit (see [Git Basics: Undoing Things](#) for details).
 - Sign-off commit: Select this option if you want to sign off your commit, i.e. to certify that the changes you are about to check in have been made by you, or that you take the responsibility for the code in question.
When this option is enabled, the following line is automatically added at the end of the commit message: **Signed off by: <username>**
7. Click the Commit button, or hover your mouse over this button to display one of the following available commit options:
 - Commit and Push: select this option to push the changes to the remote repository immediately after the commit. This option is available if you are using [Git](#) or [Mercurial](#) as a version control system.
 - Create Patch: select this option if you want PyCharm to generate a patch based on the changes you are about to commit. In the Create Patch dialog that opens, type the name of the patch file and specify whether you need a reverse patch.
 - Remote Run: select this option to [run your personal build](#). This option is only available when you are logged in to [TeamCity](#). Refer to [TeamCity plugin documentation](#) for details.

Managing Remotes

To be able to collaborate on your Git project, you need to configure remote repositories that you pull data from and push to when you need to share your work.

If you have cloned a remote Git repository, for example from GitHub, the remote is configured automatically and you do not have to specify it when you want to synchronize with it (i.e. when you perform a `pull` or a `push` operation).

Note If you've cloned your repository, the default name Git gives to the server you've cloned from is `origin`.

However, if you create a Git repository by choosing `VCS | Import into Version Control | Create Git Repository` from the main menu, you need to add a remote repository to be able to push your commits and pull data from it.

Do the following:

1. Invoke the `Push Dialog` when you are ready to push your commits by selecting `VCS | Git | Push` from the main menu, or press `Ctrl+Shift+K`.
2. If you haven't added any remotes so far, the `Define remote link` will appear instead of a remote name. Click it to add a remote.
Note that you can also add a remote from the `Push dialog` by clicking an existing remote's name.
3. In the dialog that opens, specify the remote name and URL and click `OK`.

In some cases, you also need to add a second remote repository. This may be useful, for example, if you have cloned a repository that you do not have write access to, and you are going to push changes to your own fork of the original project. Another common scenario is that you have cloned your own repository that is somebody else's project fork, and you need to synchronize with the original project and pull changes from it.

Do the following:

1. From the main menu, choose `VCS | Git | Remotes`. The `Git Remotes dialog` will open.
2. Click the `Add` button `+` on the toolbar or press `Alt+Insert`.
3. In the dialog that opens, specify the remote name and URL and click `OK`.

To edit a remote (for example, to change the name of the original project that you have cloned from `origin` to `upstream` if you are going to push to its fork), select it in the `Git Remotes dialog` and click the `Edit` button `E` on the toolbar, or press `Enter`.

To remove a repository that is no longer valid, select it in the `Git Remotes dialog` and click the `Remove` button `-` on the toolbar, or press `Alt+Delete`.

Tip You can also edit a remote from the `Push Dialog` by clicking its name.

On this page:

- [Basics](#)
- [Configuring the password policy](#)
- [Setting the master password](#)
- [Changing the master password](#)
- [Resetting the master password](#)

Basics

Every time you interact with a remote Git repository, for example, during a Pull, Update, or Push operation, you are requested to specify the private `ssh` key or passphrase to identify yourself. You may happen to use a number of remote repositories and accordingly need to remember a number of passwords.

PyCharm provides the ability to [configure a password policy](#) according to which passwords are either never saved, or saved during one session and cleared upon the session end, or stored in a special passwords database.

The passwords database is under protection of a master password. Once [specified](#), the master password can be [changed](#) or [reset](#) at any time. PyCharm stores the history of all updates to the master password until it is reset.

Warning! When you reset the master password, the history of all the previously used master passwords is removed.

Configuring the password policy

To configure the password policy

1. [Open the IDE Settings](#), then click Passwords.
2. On the [Passwords](#) page that opens, specify how you want PyCharm to process passwords for Git remote repositories. Do one of the following:
 - If you do not need PyCharm to save passwords at all, select the Do not remember passwords option.
 - To have passwords stored in the memory during a session, select the Remember passwords until closing of the application option.
 - To have passwords stored in a passwords database, select the Remember on disk (protected with master password) option and specify the master password for the storage.

Setting the master password

To set the master password

1. On the [Passwords](#) page of the Settings dialog box, select the Remember on disk (protected with master password) option.
2. Click the Reset button.
3. In the [Master Password](#) dialog box that opens, specify the password to use. Type the desired password once again to confirm your setting.

Changing the master password

To change the master password

1. On the [Passwords](#) page of the Settings dialog box, click the Change Password button.
2. In the [Change Master Password](#) dialog box that opens, type the currently used master password and the password to change it to. Confirm the new password by typing it once again.

Resetting the master password

To reset the master password

1. Open the [Reset Master Password](#) dialog box. Do one of the following:
 - On the [Passwords](#) page of the Settings dialog box, click the Reset Master Password button.
 - In the [Change Master Password](#) dialog box, click the Reset Password button.
2. In the New Password text box, type the master password to replace the current one with.
3. In the Confirm Password text box, type the new master password once again to confirm your setting.

Fetching Changes from a Remote Git Repository

The Fetch operation involves downloading changes from a remote repository without applying them locally.

Tip Basically, the Fetch operation is intended for downloading changes from a remote repository that is different from the [Origin](#).

To fetch changes from a remote repository

- On the main menu, choose VCS | Git | Fetch .
PyCharm retrieves the changes from the repository silently.

Pulling Changes from the Upstream (Git Pull)

Refreshing a local Git repository with the changes from the remote repository (Pull) involves retrieving changes (Fetch) and applying them to the local data (Merge). The Git integration with PyCharm provides interface for specifying the mandatory Pull settings and for customizing the Pull procedure.

To pull changes from a remote repository

1. On the main menu, choose VCS | Git | Pull Changes. The [Pull Changes](#) dialog box opens.
2. Specify the required Git root and the URL address or alias of the source remote repository.
3. From the Branches to Merge list, select the remote branches you want to merge to the current local branch.
4. Specify the pull strategy and additional settings using the dialog box controls described in the [Pull Changes](#) dialog box reference.

Pushing Changes to the Upstream (Git Push)

Git integration with PyCharm supports uploading changes from the [current branch](#) to its [tracked branch](#) or to any other remote branch. If [push](#) is rejected due to the lack of synchronization between your local repository and the remote storage, you can either interactively choose the strategy to update the local branch, or have PyCharm update it silently.

In this topic:

- [To push changes](#)
 - [Using force push](#)
- [To update a local branch if push is rejected](#)

To push changes

Do the following:

1. From the main menu, choose VCS | Git | Push. The [Push Commits dialog](#) opens showing all Git repositories (for multi-repository projects) and listing all commits made in the current branch in each repository since the last push.
If you have a project that uses multiple repositories that are not controlled synchronously, only the current repository is selected by default. For details on how to enable synchronous repositories control refer to [Version Control Settings: Git](#).
2. If necessary, you can modify the path to the remote repository by clicking it. The label turns into a text field where you can type the new path or invoke completion by pressing `Ctrl+Space`. If you add a new remote, it will be saved and you can edit it later via VCS | Git | Remotes (for details, see [Managing Remotes](#)).
If there are no remotes in the repository, the Define remote link appears. Click this link and specify the remote name and URL in the dialog that opens.
3. If you want to modify the target branch where you want to push, you can do this in the same way as for the remote repository. You can also click the Edit all targets link in the bottom right corner to edit all branch names simultaneously.
Note that you cannot change the local branch: the current branch for each selected repository will be pushed.
4. If you want to preview changes before pushing them, select the required commit. The right-hand pane shows the changes included in the selected commit. You can use the toolbar buttons to [examine the commit details](#).

Note If the author of a commit is different from the current user, this commit is marked with an asterisk.

6. Click the Push button when ready and select which operation you want to perform from the drop-down menu: `push` or `push --force`.

Note These choice options are only available if the Allow force push option is enabled (see [Version Control Settings: Git](#)), otherwise, you can only perform the `push` operation.

Tip You can also switch to the editing mode by pressing `Enter` or `F2` for the selected element.

Tip You can press `Ctrl+Q` for the selected commit to display extra info, such as the commit author, time, hash and the commit message.

Tip If you select an entire repository, all files from all commits will be listed in the right pane.

If the same file was modified within several commits, it will only be listed once if you select these commits or the entire repository, and if you invoke the [Differences Viewer for Files](#) for this file, all changes will be zipped together.

Using force push

When you run `push`, Git Mercurial will refuse to complete the operation if the remote repository has changes that you are missing and that you are going to overwrite with your local copy of the repository. Normally, you need to perform `pull` to synchronize with the remote before you update it with your changes.

The `--force push` command disables this check and lets you overwrite the remote repository, thus erasing its history and causing data loss.

A possible situation when you may still need to perform `--force push` is when you rebase a pushed branch and then want to push it to the remote server. In this case, when you try to push, Git Mercurial will reject your changes because the remote ref is not an ancestor of the local ref. If you perform `pull` in this situation, you will end up with two copies of the branch which you then need to merge.

Warning! Rebasing a pushed branch and modifying its history should be avoided unless absolutely necessary (for example, if you've accidentally pushed some sensitive data).

If you decide to force push the rebased branch and you are working in a team, make sure that:

- Nobody has pulled your branch and done some local changes to it
- All pending changes have been committed and pushed
- You have the latest changes for that branch

To update a local branch if push is rejected

If `push` is rejected due to the lack of synchronization between the current local branch and its tracked remote branch, PyCharm displays the [Push Rejected](#) dialog box, provided that the Auto-update if push of the current branch was rejected check box in the [Git settings](#) page of the Settings dialog box is cleared.

1. If your project uses several Git repositories, specify which of them you want to update. If you want to update all repositories, no matter whether `push` was rejected for them or not, select the Update not rejected repositories as well option. If this option is cleared, only the affected repositories will be updated.
2. If you want PyCharm to apply the update procedure silently the next time push is rejected using the update method you choose in this dialog, select the Remember the update method choice and silently update in the future option.
After you leave this dialog, the Auto-update if push of the current branch was rejected check box in the [Git settings](#) page of the Settings dialog box will be selected, and the applied update method will become the default one.

To change the update strategy, dissect this option to invoke the [Push Rejected](#) dialog next time push of the current branch is rejected, apply a different

update procedure, and select the Remember the update method choice ... option once again.

Note Note that if you haven't selected an update procedure from the Push Rejected dialog before, and you select the Auto-update if push of the current branch was rejected check box in the [Git settings](#) page of the Settings dialog, PyCharm will synchronize the local branch silently using `git-merge` by default.

3. Select the update method ([rebase](#) or [merge](#)) by clicking the Rebase or Merge button respectively.

Managing Branches

PyCharm supports processing of multiple Git branches using sophisticated checkout and merge strategies, and provides an interface for configuring operations on branches.

To manage branches, use the Branches node and the nested commands. This node is available on:

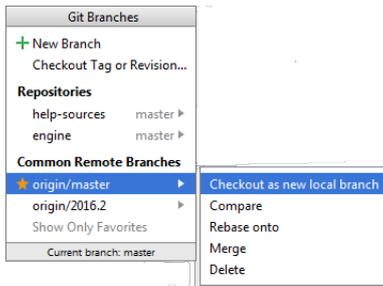
- the main VCS menu
- context menu of the editor
- VCS operations pop-up

In this part:

- [Accessing Git Branches Popup Menu](#)
- [Creating a New Branch](#)
- [Checking out \(Switching Between\) Branches](#)
- [Merging, Deleting, and Comparing Branches](#)
- [Rebasing Branches](#)
- [Resetting Head Commit](#)
- [Applying Changes from a Specific Commit to Other Branches \(Cherry Picking\)](#)
- [Git Branches in Multirooted Projects](#)

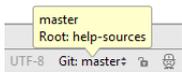
Accessing Git Branches Popup Menu

The Git Branches popup menu contains all commands that are required to manage Git branches, and can be invoked in a number of ways.



To open the Git Branches popup menu, do one of the following:

- From the main menu, choose VCS | Git | Branches.
- Right-click any file in the editor, and in the context menu choose Git | Repository | Branches.
- In the [VCS operations](#) popup menu, choose Branches.
- Click the Git widget in the Status bar:



If you have many branches, you can choose, whether you want to display all of them in the branches popup, or just the favorites. To do this, toggle the Show Only Favorites and the Show x More commands at the bottom of the branches popup.

To mark a branch as a favorite, hover the mouse cursor over the branch name, and click the star outline that appears on the left: . The current branch is marked as a favorite by default.

Creating a New Branch

On this page:

- [To create a new Git branch](#)
- [To check out a new Git branch from a local branch](#)
- [To check out a new local branch from a remote branch](#)

To create a new Git branch

1. Invoke the Branches menu as described in [Accessing Git Branches Popup Menu](#).
2. In the pop-up menu, choose New Branch.
3. In the Create new branch dialog box, specify the branch name. The branch with the specified name will be checked out (corresponds to `git checkout -b`).

To check out a new Git branch from a local branch

1. Invoke the Branches menu as described in [Accessing Git Branches Popup Menu](#).
2. Select a branch in the pop-up list that shows all available local and remote branches, and choose Checkout as new branch from the submenu.
3. Specify the branch name in the Create new branch dialog that opens.

To check out a new local branch from a remote branch

1. Invoke the Branches menu as described in [Accessing Git Branches Popup Menu](#).
2. Select a branch in the pop-up list that shows all available local and remote branches, and choose Checkout as new local branch from the submenu.
3. Specify the name of the new branch in the Checkout new branch from <branch name> dialog that opens.
The branch with the specified name will be checked out and put under version control.

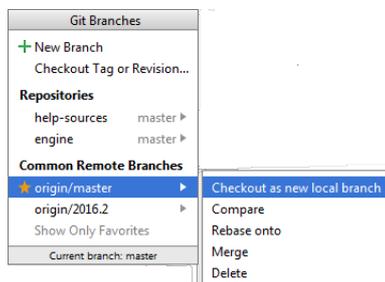
Checking out (Switching Between) Branches

On this page:

- [Checking out an existing branch](#)
- [Checking which branch is currently checked out](#)

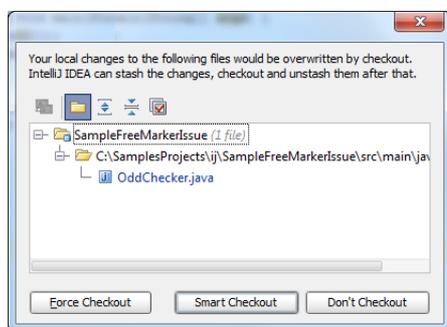
Checking out an existing branch

1. Invoke the Branches menu as described in [Accessing Git Branches Popup Menu](#).
2. Select a branch in the pop-up list that shows all available local and remote branches, and choose Checkout from the submenu.



What happens next depends on whether there are conflicts:

- If your working tree is clean and your local changes do not conflict with the specified branch, this branch will be checked out (a notification will popup in the bottom-left corner of the PyCharm window).
- If you were working with a dirty tree, and your local changes will be overwritten by checkout, PyCharm shows the files that prevent you from checking out the selected branch, and suggests to choose between the `force checkout` and `smart checkout`.



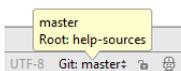
If you select `force checkout`, your local changes will be overwritten (equivalent to the `git checkout -f` command).

If you select `smart checkout`, PyCharm will `stash` local changes, check out the selected branch, and then `unstash` the changes. If a conflict occurs during the unstash operation, you will be prompted to merge the changes.

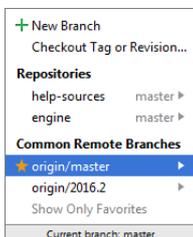
Checking which branch is currently checked out

If you want to check which branch is you are currently working in, do one of the following:

- Look at the widget in the [Status bar](#):



- Invoke the Branches menu as described in [Accessing Git Branches Popup Menu](#). Check the current branch name in the bottom line:



On this page:

- [To merge a branch](#)
- [To delete a branch](#)
- [To compare branches](#)

To merge a branch

1. Invoke the Branches menu as described in [Accessing Git Branches Popup Menu](#).
2. Select a branch in the pop-up list that shows all available local and remote branches, and choose Merge from the submenu. The selected branch will be merged into the branch that is currently [checked out](#).

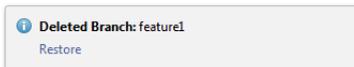
If there are merge conflicts, you will be prompted to resolve them.

If your local changes are going to be overwritten by merge, PyCharm suggests to perform **smart merge** (similar to [smart checkout](#)).

To delete a branch

1. Invoke the Branches menu as described in [Accessing Git Branches Popup Menu](#).
2. Select a branch in the pop-up list that shows all available local and remote branches, and choose Delete from the submenu.

After you have deleted a branch, a notification will be displayed in the bottom right corner from which you can restore the deleted branch:



If the deleted branch contained commits that have not yet been merged to its upstream branch or to the current branch, it will still be deleted immediately (equivalent to the `git branch --D` or `git branch --delete --force` command), but the notification will also contain a link allowing you to view the unmerged commits.

If the deleted branch was tracking a remote branch, you will also be able to remove the remote branch from this notification.

To compare branches

1. Invoke the Branches menu as described in [Accessing Git Branches Popup Menu](#).
2. Select a branch in the pop-up list that shows all available local and remote branches, and choose Compare from the submenu.

PyCharm compares the branch that is [currently checked out](#) with the selected branch.

3. In the dialog that opens, compare the differences in the following two tabs:
 - Log: this tab lists the commits that exist in the current branch, and are missing in the selected branch, and vice versa.
 - Diff: this tab shows the differences between files existing in both branches. Use the [Show Diff](#) command on the context menu of a file to explore the differences between branches.

Rebasing Branches

Git integration with PyCharm supports the Rebase operation and provides an interface that ensures high flexibility in setting the rebase arguments.

The following functionality is supported:

- The **basic use case** which involves [applying a branch on top](#) of the current HEAD of the master after synchronization with the upstream.
- Rebasing a branch entirely or partially to a [specific commit](#) in any branch or tag.
- Running rebase on several local repositories simultaneously.
- Selecting a merge strategy to apply, with the possibility to use no merging strategy at all.
- Running [rebase interactively](#) with control over preserving/squashing merges.
- [Resuming interrupted rebase](#) after merge conflicts are resolved.
- [Cancelling](#) rebase.

To initiate a rebase operation

1. From the main menu, select VCS | Git | Rebase. The [Rebase Branches](#) dialog box opens.
2. From the Git Root drop-down list, select the relevant local repository.
3. From the Branch drop-down list, select the branch you want to rebase. By default, the current branch is selected. If you specify a different branch, it will be checked out.
4. Specify the new base and the commits you want to apply.
5. If necessary, choose a rebase strategy and click Rebase.

The `rebase` command is also available from the [Git Branches popup](#) in the submenu for the selected branch.

To resume an interrupted rebase operation

- From the main menu, choose VCS | Git | Continue Rebasing.
Before resuming the rebase operation, view the log in the [Version Control](#) tool window.

If rebase was initiated and interrupted on two or more local repositories, the Continue Rebasing dialog box is displayed. Select the repository on which you want to resume the rebase operation from the Git Root drop-down list.

To cancel a rebase operation

- From the main menu, choose VCS | Git | Abort Rebasing

If rebase was initiated on two or more local repositories, the Abort Rebasing dialog box is displayed. Select the repository on which you want to cancel the rebase operation from the Git Root drop-down list.

Applying a Branch Entirely on Top of Master

Suppose you have `branch1` based on `master`. While you were working in `branch1`, some updates were committed to `master`.

The diagram below illustrates rebasing `branch1` so that it applies on top of the current HEAD of `master`.



By default, commits `1`, `2`, and `3` are applied one after another in the chronological order. To skip, edit, squash commits or change their order, run the rebase operation in the [interactive mode](#).

To apply a branch entirely on top of the current HEAD of the master

1. [Initiate the rebase](#) procedure.
2. Clear the Preserve Merges check box.
3. Clear the Interactive check box.
4. From the Onto drop-down list, select the master branch.
5. Clear the selection in the From drop-down list, if anything is selected.
6. From the Merge Strategy drop-down list, select Default.
7. Click the Rebase button. The rebase process starts. View the rebase log in the [Version Control](#) tool window, [resolve conflicts](#) that arise, and [resume](#) rebasing.

Interactive Rebase

Rebasing branches interactively provides you with the possibility to apply commits in the necessary order, squash or edit commits before applying, and skip commits that contain extraneous changes.

Interactive mode is available for rebasing branches entirely or partially on top of the HEAD or to any specific commit.

To rebase a branch in the interactive mode

1. [Initiate the rebase](#) procedure.

2. Select the Interactive check box.

Tip To have PyCharm try to recreate merges instead of ignoring them, select the Preserve Merges check box.

Warning! Git does not support squashing commits when the Preserve merges option is enabled.

3. Specify the [new base](#), the [range of commits](#) to apply, and the [merge strategy](#).

4. Click OK. The [Rebasing Commits](#) dialog box opens displaying a list of all the commits within the specified range in the chronological order. For each commit its hash and comment are shown.

Tip To view which files are affected in a commit, select the commit and click the View button.

5. Define the order of processing the commits by selecting the relevant lines and clicking the Move Up and Move Down buttons.

6. Use the Action drop-down list to define how each commit should be processed:

- To apply a commit as is, select the Pick option.
- To update a commit before applying, select the Edit option.
- To ignore a commit, select the Skip option.
- To combine a commit with the previous commit, select the Squash option.

After you start rebasing, you will be asked to supply additional information on the squashed commits.

If the affected commits have different authors, the squashed commit will be attributed to the author of the first commit.

7. Click the Rebase button. The rebase process starts. View the rebase log in the [Version Control](#) tool window, [resolve conflicts](#) that arise, and [resume](#) rebasing.

Rebasing a Branch to a Specific Commit

Suppose you have `branch1` based on `master`. While you were working in `branch1`, some updates were committed to `master`. For some reason, you do not want to rebase `branch1` to the current HEAD of `master`, but want to rebase it to commit `CommitB`. Moreover, you do not want to apply commit `1`, but want to start with commit `2`.

The diagram below illustrates the described rebase operation.



Transplanting a branch to a branch different from its base is also supported. Suppose your `branch2` is based on `branch1`, which, in its turn, is based on `master`.

The diagram below illustrates rebasing `branch2` to a specific commit in `master`.



To rebase a branch to a specific commit

1. [Initiate the rebase](#) procedure.
2. Configure the rebase mode:
 - To [interactively](#) handle the list of commits to apply, select the Interactive check box.
 - To have PyCharm try to recreate merges instead of ignoring them, select the Preserve Merges check box. Note that this option is available only in the interactive mode.
3. Define the contents of the Onto and From drop-down lists. By default, tags and remote branches are not available from them. To expand the lists with tags or remote branches, select the Show Tags and Show Remote Branches check boxes respectively.
4. From the Onto drop-down list, select the required branch and specify the commit to move the base to. Type the desired commit hash or use an expression, for example, of the following structure:

```
<branch>~<number of commits backwards between the latest commit (HEAD) and the required commit>
```

Refer to the Git [commit naming](#) conventions for details.

Tip If you select a tag, no commit specification is required.

5. Click the Validate button and view which files are affected in the specified commit in the Paths affected in commit... dialog box that opens.
6. From the From drop-down list, select the current branch and specify the commit, starting from which you want to apply commits on top of the new base. Type the desired commit hash or use an expression, for example, of the following structure:

```
<branch>~<number of commits backwards between the latest commit (HEAD) and the required commit>
```

Refer to the Git [commit naming](#) conventions for details.

Tip To apply all commits, leave the field empty.

7. Specify the merge procedure to be applied, when necessary:
 - To perform all merges manually, select the Do not use merging strategies check box.
 - To have PyCharm apply one of the available merging methods, select the relevant option from the Merge Strategy drop-down list. See the [Rebase Branches](#) dialog box reference for description of available options.
8. Click the Rebase button and proceed depending on the chosen rebase mode:
 - If you have specified the [interactive](#) mode, configure the list of commits to apply in the [Rebasing Commits](#) dialog box, that opens.
 - If you have specified the automatic mode, view the rebase log in the [Version Control](#) tool window, [resolve conflicts](#) that arise, and [resume](#) rebasing

Resetting Head Commit

Suppose, you notice a small error in a recent commit or a set of commits and want to redo that part without showing the undo in the history. In this case resetting the latest commit in the current branch (HEAD) to an earlier commit is helpful. The Reset HEAD operation sets the current HEAD to the specified commit and optionally resets the index and working tree to match.

To reset the current HEAD to an earlier commit

1. On the main menu, choose VCS | Git | Reset HEAD. The [Reset Head](#) dialog box opens.
2. From the Git Root drop-down list, choose the required local repository.
3. Make sure the Current Branch read-only field shows the branch in which you actually want to reset the HEAD. If necessary, [switch](#) to the correct branch.
4. From the Reset Type drop-down list, select the desired reset strategy to apply:
 - To have the changed files preserved but not marked for commit, select the Mixed option. The index will be reset while the working tree will not and you will get a report of what has not been updated.

Tip This is the default strategy.

- To have only the HEAD pointer moved without updating the index and the working tree, select the Soft option. Your current state with any changes will remain different from the commit you are switching to.
 - To have both the working directory and the index changed to the specified commit, select the Hard option.
5. In the To Commit text box, specify the commit you want to reset the current HEAD to. Type its commit hash or use an expression, for example, of the following structure:

```
<branch>~<number of commits backwards between the latest commit (HEAD) and the required commit>.
```

Refer to the Git [commit naming](#) conventions for details.

6. To check that the specified commit exists and is the one you actually need, click the Validate button. The Paths affected in commit dialog box shows the files that were affected in the specified commit. View the information and click OK.
If you specify a commit that does not exist, PyCharm displays an error message.

Applying Changes from a Specific Commit to Other Branches (Cherry Picking)

Suppose, you have two diverged branches, `master` and `hotfix`. One day, you decide to integrate changes from one specific commit in `master` to `hotfix`. With PyCharm, you can cherry pick between branches right from the [Version Control](#) tool window.

To cherry pick changes between two branches

1. [Switch to the target branch](#) that the changes will be integrated to.
2. Open the [Version Control](#) tool window and switch to the [Log](#) tab.
3. From the Branch drop-down list on the toolbar, select the source branch containing the changes you want to cherry-pick.
Note that the [Commits pane](#) lists all commits performed in the selected branch. To reduce the number of items in the list, you can filter the commits by the user or date. You can also click the Highlight non-picked commits button  to grey out the commits that have already been applied to the current branch.
4. Select the required commit. Use the information in the [Commit Details area](#) if necessary.
5. Click the Cherry-pick button  on the toolbar. PyCharm will display the [Commit Changes](#) dialog with the automatically generated commit message.
6. If you want to review the changes or even modify the code before committing it to the target branch, you can do so in the difference viewer available from this dialog.
7. When done, click Commit to cherry-pick the selected changes.

Note that if you click Cancel in the [Commit Changes](#) dialog, a separate changelist will be created with the selected changes that you can see in the [Local Changes tab](#). You can review these changes and commit them later if necessary.

On this page:

- [Basics](#)
- [Asynchronous branch control](#)
- [Synchronous branch control](#)
- [Rolling back a successful operation if it fails in some of the repositories](#)

Basics

Sometimes you may need to have [several local Git repositories](#) within a single project. Such multirooted project can be set up in the following ways:

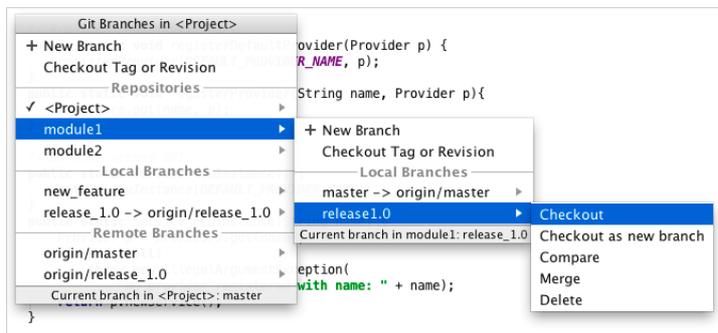
- The repositories are **independent**, they can be included in the project just as nested roots. One of these repositories can be intended for the project development itself while the others store the libraries used.
Such repositories are controlled [asynchronously](#), that is, commit, push, and other operations can be applied to them independently, just as it is done in projects with one Git repository.
- The repositories are **mutually connected**. Such repositories most often need to be controlled synchronously, that is, commit, push, and other operations should be applied to all these repositories at once. For such projects PyCharm also allows you to control branches [synchronously](#). This means that all branch operations on several repositories are performed simultaneously as if it were a single repository. In case an operation fails in at least one of the repositories, PyCharm [rolls it back](#) in the repositories where the operation has passed successfully.

Asynchronous branch control

Asynchronous control is applied to the branches in a multirooted project when all local Git repositories in the project are **independent**. Commit, push, and other operations to the branches in the repositories are applied independently, just as it is done in projects with one Git repository.

To apply a Git operation to a branch asynchronously (in one of the repositories)

1. [Open the Git Branches](#) popup menu.
2. In the Git Branches popup menu that opens, choose the branch to apply the required action to. The popup menu shows a list of branches in the current repository and a list of all the repositories in the project.
The current repository is detected in the following ways:
 - If you have explicitly selected a file or a folder in the Project tool window or elsewhere, PyCharm treats the root under which the file is located as **current repository**.
 - If you used the VCS | Git | Git Branches command on the main menu, PyCharm treats the root of the file currently opened in the editor as **current repository**.



Do one of the following:

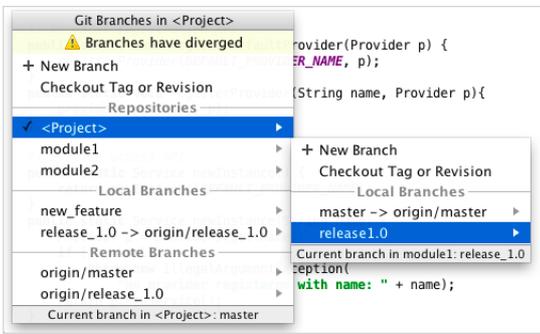
- To update a branch in the current repository, select the branch in the list.
- To access another repository, select it under the Repositories group, and then select the relevant branch in the subordinate popup menu that opens.

3. Click the branch and choose the required [action](#) from the popup menu that opens.

Synchronous branch control

In a multirooted project when several **mutually connected** repositories, it is often helpful to apply **synchronous** branch control, which involves the following:

- All branch operations on several repositories are performed simultaneously as if it were a single repository. This applies to all actions including [checkout](#), [merge](#), and [delete](#). If you invoke the [compare](#) action, PyCharm shows a dialog box where you are asked to choose the repository to compare the branch with.
- The Git Branches popup lists only those local and remote branches that are common for all the repositories in the project.
- Creating a new branch from the popup will create it on all the repositories.
- You may still control branches individually for each repository by using the repository selector in the Git Branches popup menu. If you happen to checkout a branch only in one of the repositories, you'll get the warning that branches have diverged:



To disable synchronous branch control, do one of the following

When you [invoke the Git Branches](#) popup menu for the first time, PyCharm examines the repositories registered in the project. If they all are on branches with same names, PyCharm displays a popup menu that proposes to control branches synchronously.

- Click the [Disable](#) link in the information popup window.
- [Open the Settings dialog box](#), and on the [Git](#) page under the [Version Control](#) node, clear the [Control repositories synchronously](#) check box.

To enable synchronous branch control

- [Open the Settings dialog box](#), and on the [Git](#) page under the [Version Control](#) node, select the [Control repositories synchronously](#) check box.

Rolling back a successful operation if it fails in some of the repositories

If you checkout a branch on several roots and the operation fails for some of the roots but has already succeeded for the others, the branches diverge. For example, the `master` branch is checked out in one repository while the `feature` branch is checked out in the other.

To prevent this, PyCharm offers to rollback the successful checkout operation, so the previous checked out branch is checked out again.



Besides checkout, PyCharm suggests to rollback merge and other branch operations.

Stashing and Unstashing Changes

On this page:

- [Stashing and unstashing](#)
 - [Saving changes to a new stash](#)
 - [Applying a stash](#)
 - [Removing stashes](#)
- [Stashing vs shelving](#)

Stashing and unstashing

Sometimes it may be necessary to revert your working copy to match the HEAD commit but you do not want to lose the work you have already done. This may happen if you learn that there are upstream changes that are possibly relevant to what you are doing, or if you need to make some urgent fixes.

Stashing involves recording the difference between the HEAD commit and the current state of the working directory (stash). Changes to the index can be stashed as well.

Unstashing involves applying a stored stash to a branch.

You can apply a stash to an **existing branch** or create a **new branch** on its basis.

A stash can be applied as many times as you need to any branch you need, just **switch** to the required branch. Keep in mind that:

- Applying a stash after a series of commits results in conflicts that need to be resolved.
- A stash cannot be applied to a "dirty" working copy, that is a working copy to which changes have been made since the latest commit.

You can store stashes in a list as long as you need and **remove** them from the list when necessary.

Saving changes to a new stash

1. From the main menu, choose VCS | Git | Stash Changes. The **Stash** dialog box opens.
2. Select the relevant Git root and make sure that the correct branch is checked out.
3. In the Message text box, describe the changes to be stashed.

Tip To stash the local changes and bring the changes staged in the index to your working tree for examination and testing, select the Keep Index check box.

Applying a stash

To apply a stash to the same branch

1. From the main menu, choose VCS | Git | Unstash Changes. The **Unstash Changes** dialog box opens.
2. Select the Git root where you want to apply a stash and make sure that the correct branch is checked out.
3. In the Stashes list, select the relevant stash.

Tip Examine the stash descriptions and the names of the branches where specific stashes were created to find the stash you need.

4. Click the View button to open the Paths affected in commit dialog box and find out which files are affected in the selected stash.
5. Specify additional unstash options:
 - To have the selected stash removed from the list after it is applied, select the Pop stash check box.
 - To have the stashed index modifications applied, select the Reinstate Index check box.

Warning! This operation may fail if you have conflicts. This happens because conflicts are stored in the index, where you can no longer apply the changes as they were originally.

To create a new branch on the basis of a stash

1. On the main menu, choose VCS | Git | Unstash Changes. The **Unstash Changes** dialog box opens.
2. Select the Git root where you want to apply a stash.
3. In the Stashes list, select the relevant stash.
4. In the As new branch text box, type the name of the new branch to be created.

Removing stashes

1. On the main menu, choose VCS | Git | Unstash Changes. The **Unstash Changes** dialog box opens.
2. In the Stashes list, select the stashes to be removed and click the Drop button.

Tip To remove all stashes from the list, click the Clear button.

Stashing vs shelving

As you might have noticed, Stashing and Unstashing have much in common with **Shelving** and **Unshelving**.

The only difference is in the way patches are generated and applied:

- Patches with stashed changes are generated by Git itself. To apply them later, you do not need PyCharm.
- Patches with shelved changes are generated by PyCharm. Normally, they are also applied through the IDE. Applying shelved changes outside PyCharm is also possible but requires additional steps.

Handling LF and CRLF Line Endings

The Difference Viewer points at discrepancies in line endings (LF-CRLF). For Git repositories, PyCharm displays a warning when you are about to commit CRLFs and offers to set the `core.autocrlf` setting for you.

On this page:

- [Basics](#)
- [Enabling smart handling of LF and CRLF line separators](#)
- [Handling the problems with line separators during commit](#)

Basics

Quite often people working in a team and using the same repository or upstream prefer different operating systems. This may result in problems with line endings, because **Unix, Linux, and macOS** use `LF` and **Windows** uses `CRLF` to denote the end of a line. PyCharm shows the discrepancies in line endings in the [Difference Viewer Dialog](#), so you can fix them manually.

To have Git solve such problems automatically, you need to set the `core.autocrlf` attribute to `true` on Windows and to `input` on Linux and macOS. For more details on the meaning of the `core.autocrlf` attribute, see the article [Mind the End of Your Line](#) or [Dealing with Line Endings](#). You can change the configuration manually by running `git config --global core.autocrlf true` on Windows or `git config --global core.autocrlf input` on Linux and macOS. However, PyCharm can analyze your configuration, warn you if you are about to commit `CRLF` into the repository, and offer to set the `core.autocrlf` setting to `true` or `input` depending on the operating system used.

Enabling smart handling of LF and CRLF line separators

To enable smart handling of LF and CRLF line separators

To enable smart handling of `LF` and `CRLF` line separators

1. [Open the Settings dialog box](#).
2. Under the Version Control node, click Git.
3. On the [Git page](#) that opens, select the Warn if CRLF line separators are about to be committed checkbox.

Handling the problems with line separators during commit

To handle problems with the line separators during commit

1. [Enable smart handling of LF and CRLF line separators](#). After that, PyCharm will show the [Line Separators Warning Dialog](#) every time you attempt to commit a file with `CRLF` separators, unless you have set any related [git attributes](#) on the affected file. In the latter case, PyCharm supposes that you clearly understand what you are doing and excludes the file from analysis.
2. When the [Line Separators Warning Dialog](#) is displayed, do one of the following:
 - To ignore the warning and commit the file with `CRLF` separators, click the Commit As Is button.
 - To have the `core.autocrlf` attribute set to `true` or `input` depending on the operating system used before commit, click the Fix and Commit button. As a result, all the `CRLF` separators will be replaced with `LF` separators and committed into the repository. Note that the reverse operation will not be performed when you download the files into your working directory, that is, no `CRLF` will appear in place of `LF`.
 - To stop the commit procedure, click Cancel.
3. To suppress showing the dialog box in the future, select the Don't warn again check box.

Using Mercurial Integration

With the Mercurial integration enabled, you can perform basic Mercurial operations from inside PyCharm.

Note The information provided in the topics listed below assumes that you are familiar with the basics of Mercurial version control system.

In this section:

- Using Mercurial Integration
 - [Prerequisites](#)
 - [Mercurial support](#)
- [Adding Files To a Local Mercurial Repository](#)
- [Setting Up a Local Mercurial Repository](#)
- [Managing Mercurial Branches and Bookmarks](#)
- [Switching Between Working Directories](#)
- [Pulling Changes from the Upstream \(Pull\)](#)
- [Pushing Changes to the Upstream \(Push\)](#)
- [Tagging Changesets](#)

Prerequisites

- [Mercurial](#) is installed on your computer.
- The location of the Mercurial executable file `hg.exe` is correctly specified on the Mercurial page of the Settings/Preferences dialog box. If you followed the standard installation procedure, the default location is `/opt/local/bin` or `/usr/local/bin` for Linux and macOS and `/Program Files/TortoiseHG` for Windows.

It is recommended that you add the path to the Mercurial executable file to the `PATH` variable. In this case, you can specify only the executable name, the full path to the executable location is not required. For more information about environment variables, see [Path Variables](#).

- Mercurial [integration is enabled](#) for the current project root or directory.

If you want to use a remote repository, create a Mercurial hosting account first. You can access the remote repository through a pair of `ssh keys` or apply the username/password and keyboard interactive authentication methods supported by the Mercurial integration.

Mercurial support

- When Mercurial integration with PyCharm is enabled, the Mercurial item appears on the VCS menu, and on the context menus of the [Editor](#) and [Project](#) views.
- The files in the folders under the Mercurial control are highlighted according to their status. See [File Status Highlights](#) for file status highlighting conventions.
- Modifications results are shown in the [Version Control tool window](#).
- When using Mercurial integration, it is helpful to open the [Version Control](#) tool window. The [Console](#) tab displays the following data:
 - All commands generated based on the settings you specify through the PyCharm user interface.
 - Information messages concerning the results of executing generated Mercurial commands.
 - Error messages.

Adding Files To a Local Mercurial Repository

After a Mercurial repository for a project is [initialized](#), you need to add the project data to it:

- If you have specified Mercurial as the version control system for your project in the Settings dialog box, PyCharm suggests to put each new file under Mercurial control during the file creation.
To have Mercurial ignore some types of files, [configure files to ignore](#).
- You can [add all unversioned](#) files to Mercurial control or [select files to add](#).

To add all currently unversioned files to Mercurial control

1. Switch to the [Version Control](#) tool window.
2. In the [Local Changes](#) tab, navigate to the Unversioned Files node and choose Add to VCS from the context menu.

To add specific file(s) to a local Mercurial repository, do one of the following:

- Switch to the [Version Control](#) tool window, expand the Unversioned Files node, and select the files to be added. From the context menu, choose Add to VCS.
- Switch to the [Project](#) tool window and select the files to be added. From the context menu, choose Mercurial | Add to VCS.

Setting Up a Local Mercurial Repository

Although Mercurial provides high flexibility in arranging data and your work with repositories, the following scenarios are most commonly used for setting up a local Mercurial repository:

- [Clone](#) an existing remote repository and create a new project with the downloaded data.
- [Create a local repository](#) which you can push to a remote location later, if necessary.

To clone a remote Mercurial repository

1. On the main menu, choose VCS | Checkout from Version Control | Mercurial. The Clone Mercurial Repository dialog box opens.
2. In the Mercurial Repository URL text box, type the URL of the remote repository which you want to clone.
3. Click the Test Repository button next to the Mercurial Repository URL text box to check that connection to the remote repository can be established successfully.
4. In the Parent Directory text box, specify the directory where you want PyCharm to create a folder for your local Mercurial repository. Use the Browse button  button, if necessary.
5. In the Directory Name text box, specify the name of the new folder into which the repository will be cloned. Click Clone.
6. Create a new project based on the cloned data by accepting the corresponding suggestion displayed by PyCharm.

To create a local Mercurial repository

1. Open the project you want to store in a repository.
2. On the main menu, choose VCS | Import into Version Control | Create Mercurial Repository. The [Create Mercurial Repository](#) dialog box opens.
3. Specify the location of the new repository.
 - To have the repository created in the project root, choose the Create repository for the whole project option. PyCharm will create the `.hg` directory in the project root folder. This option is selected by default.
 - To have a new repository created in another location, choose the Select where to create repository option and specify the path to the repository location in the text box below. Type the path manually or click the Browse button  and choose the relevant folder in the Select directory for hg init dialog box that opens.

Warning! Mercurial does not support external paths. So if you choose another directory, note that it must contain the tree where the project root resides.

If you choose a directory which is already under Mercurial control, PyCharm opens the Directory Is Under hg dialog box, where you can choose to create a repository in the specified location or to stay in the parent repository.

4. Put the required [files under Mercurial](#) version control. The files appear in the [Version Control](#) tool window under the Default node. Note that if you specify Mercurial as the version control system for a directory in the [Version Control](#) dialog box, PyCharm will suggest to put each new file in this directory under Mercurial control.

With PyCharm, you can use both [named branches](#) and [light-weight branches \(bookmarks\)](#). PyCharm provides interface for creating, merging, and switching between branches and bookmarks, see [Switching Between Working Directories](#). You can also run commands in the embedded Terminal, see [Working with Embedded Local Terminal](#).

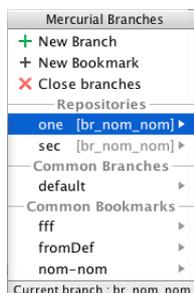
On this page:

- [Opening the Branches pop-up list](#)
- [Creating a named branch](#)
- [Creating a bookmark](#)
- [Closing a branch](#)
- [Merging named branches and bookmarks](#)
- [Merging a named branch or bookmark with another named branch or bookmark](#)
- [Merging a named branch or bookmark with a changeset](#)

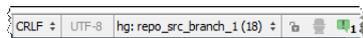
For information about switching between branches and bookmarks, see [Switching Between Working Directories](#).

Opening the Branches pop-up list

Most of the operations with branches and bookmarks are invoked from the Branches pop-up list.

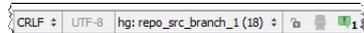


The list shows all the Mercurial repositories under the project root, all the named branches, and all the bookmarks in them. The current repository and the current bookmark are marked with a tick. The name of the current named branch is shown in the dedicated hg area on the Status bar:



To open the Branches pop-up list, do one of the following:

- On the Status bar, click the name of the current named branch in the dedicated hg area.



- On the main menu, choose VCS | Mercurial | Branches.
- On the context menu of the Editor or Version Control tool window, choose Mercurial | Branches.

Creating a named branch

1. In the Branches pop-up list, click New Branch.
2. In the Create New Branch dialog box that opens, specify the name of the new branch.

The new branch immediately becomes active and its name is shown on the Status bar in the hg area.

Creating a bookmark

1. In the Branches pop-up list, click New Bookmark.
2. In the [New Bookmark dialog box](#) that opens, specify the name of the bookmark to be created.
3. Specify, whether you want to switch to the new bookmark immediately or not.
 - To activate the new bookmark and thus enable tracking and updating the light-weight branch the bookmarks identifies, leave the Inactive check box cleared. The new bookmark immediately becomes active and its name is marked with a tick in the Branches pop-up list.
 - To have an inactive bookmark created, that is, to remain in the current light-weight branch (bookmark) or named branch and switch to the new bookmark later, select the Inactive check box.

Closing a branch

According to [Mercurial workflows](#), when you are done with a feature development and do not expect any further changes, you close the corresponding branch. A closed branch is not displayed among active branches, in the [Log view](#), etc. To close a branch, do the following:

1. In the Branches popup, click Close branch. The [Commit changes dialog](#) will be displayed.
2. Click Commit and Close. All changes will be committed and the current branch will be closed.

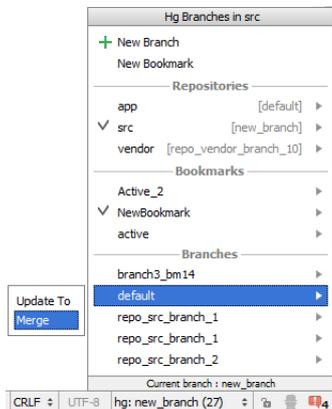
Note that if you have several repositories listed in the Repositories section, the corresponding menu option will toggle to Close branches and the `close` operation will be applied to all of them.

Merging named branches and bookmarks

You can merge a named branch or a bookmark with another named branch, another bookmark, or a specific changeset identified by a tag or a revision number.

– Merging a named branch or a bookmark with another named branch or bookmark means merging with its **head**.

Merging with named branches and bookmarks can be invoked through the menu item VCS | Mercurial | Merge, which opens the Mercurial-specific Merge dialog box of from the Branches pop-up list.



– Merging a named branch or a bookmark with a changeset means merging the branch **head** with the specified changeset. A changeset can be identified either by a revision number or a tag, see <http://mercurial.selenic.com/wiki/Tag?highlight=%28bCategoryGlossaryb%29>.

Merging a named branch or a bookmark with a specific changeset can be invoked **only** through VCS | Mercurial | Merge.

For definitions and Mercurial-specific details regarding the merge operation itself, see <http://www.selenic.com/hg/help/merge>.

By default, **Mercurial** requires that before merge the current working directory should be **clean**, that is, it should not contain any uncommitted changes. Otherwise the merge operation fails and PyCharm shows the corresponding error message. The message also recommends that you clean the current working directory by running the `hg merge <target branch, bookmark, or changeset> -C` to discard the uncommitted changes.

If your current working copy is not clean, you can either commit the changes or shelve them as described in [Shelving and Unshelving Changes](#).

Merging a named branch or bookmark with another named branch or bookmark

Merging a named branch or a bookmark with another named branch or bookmark means merging with its **head**.

1. Make sure, your current working directory is **clean**, that is, it does not contain any uncommitted changes. Commit or shelve the changes, if any.

2. Invoke merge by doing one of the following:

- In the Branches pop-up list, click the name of the branch or bookmark to merge with, then choose Merge on the pop-up menu:
- Choose VCS | Mercurial | Merge on the main menu or Mercurial | Merge on the context menu of the Editor.

In the Merge dialog box that opens:

1. Choose the target repository from the Repository drop-down list which shows all the Mercurial repositories available under the current project root.
2. Choose the Branch or Bookmark option and choose the named branch or bookmark to merge the current working directory with.

3. Resolve conflicts. As soon as a conflict takes place, the Files Merged with Conflicts dialog box opens with a list of conflicting files. Use the controls of the dialog box to resolve the problems:

- To have the version of the current working directory preserved, click Accept Yours.
- To have the version of the branch you are merging with preserved, click Accept Theirs.
- To resolve the conflicts manually, click Merge and use the Conflict Resolution Tool, as described in [Resolving Conflicts](#).

If no conflicts arise during merge, the operation passes silently and the merge log is shown in the Version Control tool window.

Merging a named branch or bookmark with a changeset

Merging a named branch or a bookmark with a changeset means merging the branch **head** with the specified changeset. A changeset can be identified either by a revision number or a tag, see <http://mercurial.selenic.com/wiki/Tag?highlight=%28bCategoryGlossaryb%29>.

1. Make sure, your current working directory is **clean**, that is, it does not contain any uncommitted changes. Commit or shelve the changes, if any.

2. Choose VCS | Mercurial | Merge on the main menu or Mercurial | Merge on the context menu of the Editor.

3. In the Merge dialog box that opens:

1. Choose the target repository from the Repository drop-down list which shows all the Mercurial repositories available under the current project root.
2. Choose the Tag or Revision option and choose the tag or specify the hash or revision number to merge the current working directory with. To copy a hash, open the Log tab of the Version Control tool window, select the relevant branch and revision, and then choose Copy Hash on the context menu of the selection.

4. Resolve conflicts. As soon as a conflict takes place, the Files Merged with Conflicts dialog box opens with a list of conflicting files. Use the controls of the dialog box to resolve the problems:
- To have the version of the current working directory preserved, click Accept Yours.
 - To have the version of the branch you are merging with preserved, click Accept Theirs.
 - To resolve the conflicts manually, click Merge and use the Conflict Resolution Tool, as described in [Resolving Conflicts](#).

If no conflicts arise during merge, the operation passes silently and the merge log is shown in the Version Control tool window.

Switching Between Working Directories

The Mercurial integration with PyCharm provides the possibility to switch update the repository's [working directory](#) to the specified [changeset](#) or a specific [line of development](#). Changesets can be identified by their hashes or by previously assigned [tag identifiers](#).

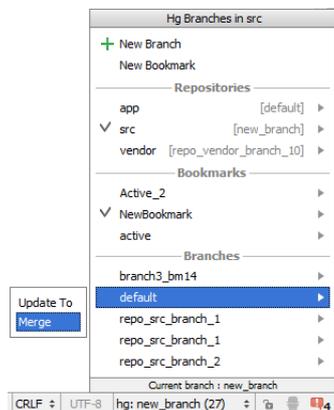
On this page:

- [Opening the Branches pop-up list](#)
- [Switching to another named branch or bookmark](#)
- [Switching to another changeset](#)

You can update a named branch or a bookmark to another named branch, another bookmark, or a specific changeset identified by a tag or a revision number.

- Updating a named branch or a bookmark to another named branch or bookmark means updating to its **head**.

Updating to named branches and bookmarks can be invoked through the menu item VCS | Mercurial | Update to, which opens the Mercurial-specific Switch Working Directory dialog box of from the Branches pop-up list.



- Updating a named branch or a bookmark to a changeset means updating the branch **head** to the specified changeset. A changeset can be identified either by a revision number or a tag, see <http://mercurial.selenic.com/wiki/Tag?highlight=%28\bCategoryGlossary\b%29>.

Updating a named branch or a bookmark to a specific changeset can be invoked **only** through VCS | Mercurial | Update to.

By default, **Mercurial** requires that before update the current working directory should be **clean**, that is, it should not contain any uncommitted changes.

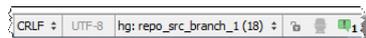
Otherwise the update operation fails and PyCharm shows the corresponding error message. The message also recommends that you clean the current working directory by running the `hg update <target branch, bookmark, or changeset> -C` to discard the uncommitted changes.

If your current working copy is not clean, you can either commit the changes or shelve them as described in [Shelving and Unshelving Changes](#). PyCharm provides the possibility to discard any uncommitted changes when the update operation is already invoked. This option is available only in the Mercurial-specific Switch Working Directory dialog box.

Opening the Branches pop-up list

To open the Branches pop-up list, do one of the following:

- On the Status bar, click the name of the current named branch in the dedicated hg area.



- On the main menu, choose VCS | Mercurial | Branches.
- On the context menu of the Editor or Version Control tool window, choose Mercurial | Branches.

Switching to another named branch or bookmark

Updating a named branch or a bookmark to another named branch or bookmark means updating to its **head**.

1. Make sure, your current working directory is **clean**, that is, it does not contain any uncommitted changes. Commit or shelve the changes, if any. If you invoke update through the Switch Working Directory dialog box, you can also prevent conflicts by having any uncommitted changes discarded.

2. Invoke update by doing one of the following:

- In the Branches pop-up list, click the name of the branch or bookmark to update to, then choose Update to on the pop-up menu:
- Choose VCS | Mercurial | Update to on the main menu or Mercurial | Update to on the context menu of the Editor.

In the Switch Working Directory dialog box that opens:

1. Choose the target repository from the Repository drop-down list which shows all the Mercurial repositories available under the current project root.
2. Choose the Branch or Bookmark option and choose the named branch or bookmark to update the current working directory to.
3. To prevent failures during update if the current working directory is not clean, select the Overwrite locally modified files (no backup) check box. The uncommitted changes will be discarded.

Switching to another changeset

1. Make sure, your current working directory is **clean**, that is, it does not contain any uncommitted changes. Commit or shelve the changes, if any.

If you invoke update through the Switch Working Directory dialog box, you can also prevent conflicts by having any uncommitted changes discarded.

2. Invoke update by doing one of the following:

- In the Branches pop-up list, click the name of the branch or bookmark to update to, then choose Update to on the pop-up menu:
- Choose VCS | Mercurial | Update to on the main menu or Mercurial | Update to on the context menu of the Editor.

In the Switch Working Directory dialog box that opens:

1. Choose the target repository from the Repository drop-down list which shows all the Mercurial repositories available under the current project root.
2. Choose the Branch or Bookmark option and choose the named branch or bookmark to update the current working directory to.
3. To prevent failures during update if the current working directory is not clean, select the Overwrite locally modified files (no backup) check box. The uncommitted changes will be discarded.

3. Resolve conflicts. As soon as a conflict takes place, the Files Merged with Conflicts dialog box opens with a list of conflicting files. Use the controls of the dialog box to resolve the problems:

- To have the version of the current working directory preserved, click Accept Yours.
- To have the version of the branch you are merging with preserved, click Accept Theirs.
- To resolve the conflicts manually, click Merge and use the Conflict Resolution Tool, as described in [Resolving Conflicts](#).

If no conflicts arise during update, the operation passes silently and the update log is shown in the Version Control tool window.

Pulling Changes from the Upstream (Pull)

Refreshing a local Mercurial repository with the changes from the remote repository (Pull) involves retrieving changes and applying them to the local data . The Mercurial integration with PyCharm provides interface for specifying the mandatory Pull settings and for customizing the Pull procedure.

To pull changes from a remote repository

1. On the main menu, choose VCS | Mercurial | Pull Changesets. The **Pull** dialog box opens.
2. Specify the required URL address of the source remote repository.

Pushing Changes to the Upstream (Push)

Do the following:

1. From the main menu, choose VCS | Mercurial | Push . The [Push Commits dialog](#) opens showing all Mercurial repositories (for multi-repository projects) and listing all commits made in the current branch in each repository since the last push.
If you have a project that uses multiple repositories that are not controlled synchronously, only the current repository is selected by default. For details on how to enable synchronous repositories control refer to [Version Control Settings: Mercurial](#) .
2. If necessary, you can modify the path to the remote repository by clicking it. The label turns into a text field where you can type the new path or invoke completion by pressing `Ctrl+Space` .
If there are no remotes in the repository, the Define remote link appears. Click this link and specify the remote name and URL in the dialog that opens.
3. If you want to preview changes before pushing them, select the required commit. The right-hand pane shows the changes included in the selected commit.
You can use the toolbar buttons to [examine the commit details](#).

Note If the author of a commit is different from the current user, this commit is marked with an asterisk.

5. If you want to push active bookmarks with your commits (they are not sent to the remote repositories by default), select the Export Active Bookmarks option.
6. Click the Push button when ready and select which operation you want to perform from the drop-down menu: `push` or `push --force` .

Tip You can also switch to the editing mode by pressing `Enter` or `F2` for the selected element.

Tip You can press `Ctrl+Q` for the selected commit to display extra info, such as the commit author, time, hash and the commit message.

Tip If you select an entire repository, all files from all commits will be listed in the right pane.

If the same file was modified within several commits, it will only be listed once if you select these commits or the entire repository, and if you invoke the [Differences Viewer for Files](#) for this file, all changes will be zipped together.

Using force push

When you run `push` , Mercurial will refuse to complete the operation if the remote repository has changes that you are missing and that you are going to overwrite with your local copy of the repository. Normally, you need to perform `pull` to synchronize with the remote before you update it with your changes.

The `--force push` command disables this check and lets you overwrite the remote repository, thus erasing its history and causing data loss.

A possible situation when you may still need to perform `--force push` is when you rebase a pushed branch and then want to push it to the remote server. In this case, when you try to push, Mercurial will reject your changes because the remote ref is not an ancestor of the local ref. If you perform `pull` in this situation, you will end up with two copies of the branch which you then need to merge.

Warning! Rebasing a pushed branch and modifying its history should be avoided unless absolutely necessary (for example, if you've accidentally pushed some sensitive data).

Warning! Using the `--force` will lead to all new heads being pushed on all branches, which is likely to cause confusion for your team.

If you decide to force push the rebased branch and you are working in a team, make sure that:

- Nobody has pulled your branch and done some local changes to it
- All pending changes have been committed and pushed
- You have the latest changes for that branch

Tagging Changesets

PyCharm supports both **local** and **global** tags. **Local** tags are stored in the file `.hg/localtags` in the repository, **global** tags are stored in the file `.hgtags`.

Currently tagging specific changesets is supported only in the command line mode in the embedded Terminal. To launch the Terminal, hover your mouse pointer over  in the lower left corner of the IDE, then choose Terminal from the menu (see [Working with Embedded Local Terminal](#) for details). After commit, tags appear in the Log tab of the Version Control tool window.

For more information about Mercurial tags, see [Mercurial Documentation: Tag](#).

Tagging a repository

PyCharm provides UI for tagging the current repository which means assigning a tag to its **tip**. The created tag is **global**, is stored in the file `.hgtags`. After commit, the tag appears in the Log tab of the Version Control tool window.

1. Open the Tag dialog box by doing one of the following:
 - On the main menu, choose VCS | Mercurial | Tag Repository.
 - On the context menu of the Editor, choose Mercurial | Tag Repository.
2. In the Tag dialog box that opens, specify the tag name. The name must be unique.

Using Perforce Integration

This feature is supported in the Professional edition only.

With the Perforce integration enabled, you can perform basic Perforce operations from inside PyCharm.

Note The information provided in the topics listed below assumes that you are familiar with the basics of Perforce version control system.

In this section:

- Using Perforce Integration
 - [Prerequisites](#)
 - [Preliminary steps](#)
 - [Perforce support](#)
- [Enabling and Configuring Perforce Integration](#)
- [Handling Modified Without Checkout Files](#)
- [Integrating Perforce Files](#)
- [Resolving Conflicts with Perforce Integration](#)
- [Showing Revision Graph and Time-Lapse View](#)
- [Using Multiple Perforce Depots with P4CONFIG](#)
- [Working Offline](#)
- [Attaching and Detaching Perforce Jobs to Changelists](#)
- [Checking Perforce Project Status](#)

Prerequisites

- A Perforce client is installed on your computer.
- You have an account with the Perforce depot.

To start using Perforce integration, perform the following preliminary steps

1. Create a client spec using your Perforce client.
2. Get source files from the Perforce depot using your Perforce client.
3. As soon as the local working copy is on your computer, [associate your local directory with Perforce](#).

After that you will be able to open source files for edit, and perform the usual Perforce-related tasks using PyCharm.

Perforce support

- When Perforce integration with PyCharm is enabled, the Perforce item appears on the VCS menu, and on the context menus of the [Editor](#) and [Project](#) views.
- The files in the folders under the Perforce control are highlighted according to their status. See [File Status Highlights](#) for file status highlighting conventions.
- Modifications results are shown in the [Version Control tool window](#).
- When using Perforce integration, it is helpful to open the [Version Control](#) tool window. The [Console](#) tab displays the following data:
 - All commands generated based on the settings you specify through the PyCharm user interface.
 - Information messages concerning the results of executing generated Perforce commands.
 - Error messages.

Enabling and Configuring Perforce Integration

The Perforce Integration is disabled by default. If you want to perform Perforce-related operations right from PyCharm, enable the integration at the IDE level and associate the project root or specific directories with Perforce. The general procedure is described in the section [Enabling Version Control](#).

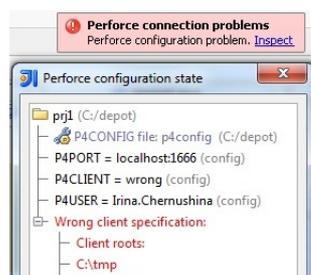
In this section:

- [Enabling Perforce integration for a project or directory](#)
- [Configuring Perforce integration settings](#)

To enable Perforce integration for a project or directory

1. Open the [Settings dialog box](#), and click [Version Control](#).
2. In the [Version Control](#) page, that opens, point to the desired root.
This can be <Project Root> or a [content root](#).
3. From the VCS drop-down list, choose Perforce.

If you specify a wrong client workspace, and your project roots do not match with the workspace roots, PyCharm displays a pop-up window with a warning.



Click [Inspect](#) to view and fix the discrepancies.

To configure Perforce integration

1. Open the [Settings dialog box](#), and then open the [Perforce](#) page under the [Version Control](#) node.
2. To establish live connection to the Perforce server, select the [Perforce is online](#) check box.
3. Specify which credentials you want to use for connecting to the Perforce server.
 - To use the connection settings from your `P4CONFIG` files, choose the [Use P4CONFIG](#) or [default connection](#) option.
If you are using `P4CONFIG` files for configuration, PyCharm shows what config files it has found and what other default settings are used. This way you can be sure that your `P4CONFIG` files are detected and taken into account.
 - To configure connection manually, choose the [Use connection parameters](#) option and specify the following settings in the corresponding text boxes:
 1. The **Port** the Perforce client will listen to.
 2. The **Client name**.
 3. Your **User name** and **Password** to authenticate to the server.
4. To use ticket-based authentication, select the [Login authentication](#) check box. Otherwise, password-based authentication will be used. In either cases PyCharm uses the login name and password specified in the dialog box or stored in the `P4CONFIG` files.
5. To attempt to log on the Perforce server without authentication, select the [Try to login silently](#) check box.
6. To have PyCharm create a `P4.output` file and store the output of Perforce commands in it, select the [Dump Perforce Commands to:](#) check box.
7. Specify the path to the Perforce executable file. Click [Test Connection](#) to make sure your settings ensure successful connection.
8. In the [Timeout](#) field, specify the lapse of time in seconds during which PyCharm waits for response from the server. If the server does not respond in due time, the user is prompted to disable integration.
9. To enable displaying the branch history of a specified file, including all file branch points, edits, and merges, select the [Show branching history](#) check box.
10. To have PyCharm point at committed changes that are also integrated to other changelists and provide information on the target changelists that received the content in question, select the [Show integrated changelists in committed changes](#) check box.
11. To get the user interface for attaching and detaching Perforce jobs to changelists, select the [Enable Perforce Jobs Support](#) check box.

Handling Modified Without Checkout Files

If you are going to modify or delete a file under Perforce version control, the read-only status of such file should be removed. PyCharm takes care of automatically making files writable. However, you can change read-only status manually, which may happen in a number of ways; for example:

- [With the Clear Read-Only Status dialog enabled](#), you make a file writable using file system.
- When a read-only file is opened in the editor, you double-click lock icon  in the status bar.
- You remove read-only attribute externally, using file properties.

In these cases, the file gets status Modified without checkout and appears in the [Local Changes tab of the Version Control tool window](#).

To resolve 'modified without checkout' files

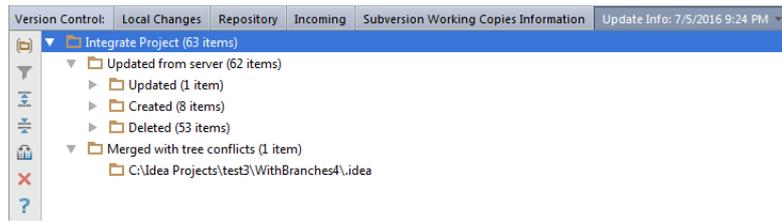
1. In the [Local Changes tab of the Version Control tool window](#), expand Modified without Checkout node, and select the desired file.
2. On the context menu of the file, choose Check Out. The file becomes writable, and moves to the active changelist.

Integrating Perforce Files

You can merge changes between the branches into your local working copy, using the branch specification, or a changelist.

The Integrate Project command is available for both Perforce and Subversion integrations.

Integration results display in the Integrate Info tab of the Version Control tool window. Context menu of a file enables you to compare versions, view history and annotations, browse changes and more.



To integrate a Perforce branch into a project

1. On the main menu, choose VCS | Integrate Project.
2. In the Integrate dialog box, select the Perforce tab (if both Perforce and Subversion integrations are used in this project).
3. Select the sources to be merged, and the desired revision.
4. Define VCS-specific merge options.
5. Click OK.

Resolving Conflicts with Perforce Integration

As described in the section [Resolving Conflicts](#), the conflicts might occur in course of team work. Perforce integration makes use of the following commands:

- Resolve enables you to resolve a conflict to a specific file.
- Resolve All applies to all files in a changelist that have merge status.

To resolve conflicts for the files under Perforce version control

1. In the [Local Changes tab of the Version Control tool window](#), select a conflicting file, or a whole conflict node.
2. On the context menu of the selection, choose Resolve or Resolve All. Files Merged with Conflicts dialog box appears.
3. If you want to accept server version and overwrite your local changes, click Accept Theirs. If you want to force your changes to the repository, click Accept Yours. Clicking Merge opens the merge tool, where you can [accept or discard](#) each change individually.
4. Once the conflicts are successfully resolved, [commit](#) your local version to the repository.

Showing Revision Graph and Time-Lapse View

Using Perforce integration, you can view the Revision Graph or Time-lapse View for a file without leaving the IDE, provided that `p4v.exe` is installed on your computer.

To show Revision Graph or Time-lapse View for a file

1. Select the desired file in any navigation tool window, or open it in the editor.
2. On the main Version Control menu, or on the context menu of the selection, choose Perforce, and then select Revision Graph, or Time-Lapse View on the submenu.
3. With the first invocation, enter password in the login dialog box.

Using Multiple Perforce Depots with P4CONFIG

If your project contains directories that are stored in the different Perforce depots, you might need to switch between them. PyCharm uses P4CONFIG to automatically switch to the respective depot as you use a Perforce-versioned directory.

P4CONFIG is an environment variable that contains the name of P4CONFIG file without a path. If a certain directory is associated with Perforce, PyCharm seeks for P4CONFIG file in this directory and its parents; if the file is not found, it is sought in the bin directory of PyCharm installation. When a P4CONFIG file is found, PyCharm uses the settings contained therein, to connect to the respective Perforce depot.

A sample P4CONFIG file might consist of such lines:

```
P4CLIENT=MyClient  
P4USER=MySelf  
P4PORT=ida:3456
```

To use multiple Perforce depots in a project, follow these general steps

1. Create a P4CONFIG file in each directory associated with Perforce.
2. Create environment variable P4CONFIG that contains file name without a path.

Working Offline

In this section:

- [Offline mode basics](#)
- [Going offline](#)
- [Going online](#)

Offline mode basics

The Perforce plugin keeps a log of VCS operations performed while offline, and replays the log when the user comes back online. The log of operations is stored in the `.iws` file and persists between PyCharm restarts.

While offline, you can perform the following operations, which will be automatically replayed in online mode:

- Edit
- Add/Copy
- Delete
- Move/Rename
- Revert
- Move to another changelist
- View Committed/Incoming changes (displaying cached information only).

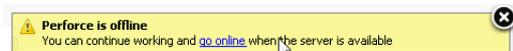
The following operations are not supported in offline mode: update, commit, integrate, tracking of the unversioned, locally deleted and *modified without checkout* files (unversioned files are shown as *unchanged*), and any other operations that require server connection.

Tip The performance of PyCharm Perforce integration in offline mode is considerably better than in online mode (because no server calls are required), so you might want to use offline mode even though connection to the Perforce server is successful.

To go to offline mode, do one of the following

- **Automatically**, when the Perforce server becomes unavailable. PyCharm switches to the offline mode automatically, and displays an offline notification in a pop-up window. To enable this behaviour, select the Switch to offline mode automatically if Perforce is unavailable check box in the [Perforce](#) page of the [Settings](#) dialog box.
- Manually at anytime, by choosing VCS | Perforce and select Work Offline on the main menu.

When offline mode is activated, the following notification balloon appears:



This balloon fades after a while; **Perforce is offline** message appears at the bottom of the [Local Changes](#) tab of the Version Control tool window.

To return to online mode, do one of the following

- Choose VCS | Perforce and clear Work Offline.
- In the offline notification balloon, click the Go online link.
- In the [Local Changes](#) tab of the Version Control tool window, click the Go online link:



Attaching and Detaching Perforce Jobs to Changelists

The Perforce integration with PyCharm provides you with a user interface for attaching and detaching Perforce [jobs](#) to changelists.

Note that the integration does not support creation of Perforce jobs.

To get access to working with Perforce jobs, select the Enable Perforce Jobs Support check box on the [Perforce](#) page of the [Settings](#) dialog box.

From this topic you will learn how to:

- Attach jobs to changelists:
 - [At any stage](#) of your work from the [Local](#) tab.
 - [During commit](#) from the [Commit Changes](#) dialog box.
- Find the desired job to attach to a changelist using:
 - The [standard search](#) functionality, which enables you to specify numerous search criteria and thus to flexibly configure the search procedure.
 - The [quick search](#) functionality, which enables you to have the found job linked to the changelist automatically.

Tip This functionality is helpful when you need to attach only one job to a changelist and you either know the exact name of the desired job or at least can specify a search pattern for the name.

- [Detach](#) jobs from changelists.

To find and link a job at any stage of your work

1. Open the [Local](#) tab of the [Version Control](#) tool window.
2. Select the changelist you want to link a job to.
3. From the context menu of the changelist, choose Edit Associated Jobs. The [Edit Jobs Linked to Changelist](#) dialog box opens.
4. Find the desired job. Do one of the following:
 - To use the [standard search](#) functionality, click the  button.
 - To use the [quick search](#) functionality, click the [+](#) button.
5. View the details of the found job and click OK.

To find and link a job during commit

1. In the [Local Changes](#) tab of the [Version Control](#) tool window, select the changelist you want to link a job to and open the [Commit Changes](#) dialog box.
2. Find the desired job using the controls in the Jobs area. Do one of the following:
 - To use the [standard search](#) functionality, click the  button.
 - To use the [quick search](#) functionality, click the [+](#) button.
3. View the details of the found job and continue the [commit](#) procedure.

To find a job using the standard search functionality

1. In the [Edit Jobs Linked to Changelist](#) dialog box or in the Jobs area of the [Commit Changes](#) dialog box, click the  button. Which dialog box you are currently in depends on whether you are linking jobs [during commit](#) or at any [other stage](#) of work.
2. In the [Link Job to Changelist](#) dialog box that opens, specify the desired search parameters.

Tip At least one of the fields should be filled in.

3. Click the Search button. The jobs that match the specified criteria are shown in the Search Results list. To view the details of a job, select it in the list.
4. Select the desired job and click OK. The dialog box closes and you return to the dialog box where you started the search:
 - The [Commit Changes](#) dialog box, if you are linking jobs [during commit](#).
 - The [Edit Jobs Linked to Changelist](#) dialog box, if you are linking jobs at any [other stage](#) stage of work.

To quickly find and link one job

1. Open the [Edit Jobs Linked to Changelist](#) dialog box or switch to the Jobs area of the [Commit Changes](#) dialog box. The dialog box to be used depends on whether you are linking jobs [during commit](#) or at any [other stage](#) of work.
2. In the text box, type the desired job name search pattern and click the [+](#) button. The job is found and automatically linked to the current changelist. If no job matching the specified pattern is found, the corresponding information message is displayed.

Warning The details of jobs that are found and linked through the quick search functionality are available only in the [Edit Jobs Linked to Changelist](#) dialog box.

To detach a job from a changelist

– In the Edit Jobs Linked to Changelist dialog box or in the Jobs area of the Commit Changes dialog box, select the desired job and click the  button.

The dialog box to be used depends on whether you are detaching jobs [during commit](#) or at any [other stage](#) of work.

Checking Perforce Project Status

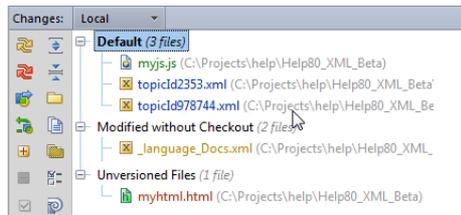
Besides indicating the [current file status](#) relative to the repository, PyCharm integration with Perforce provides you with the accumulated view of the project files' statuses.

In this section:

- [Viewing the statuses of project files](#)
- [Refreshing file status](#)

To view differences between the current state of the project files and the repository

1. Open the required project.
2. On the main menu, choose VCS | Refresh File Status.
3. Switch to the Version Control tool window, tab **Local**.



The status of each file is indicated by the [color](#) in which the path to the file is displayed.

To refresh the statuses of project files

PyCharm provides two refresh modes for statuses of files under Perforce control.

- **Standard Refresh** takes into consideration only the changes made through the PyCharm integration with Perforce. This improves performance because does not require connecting to the server. However, this approach does not let you know about the changes made outside PyCharm, for example, right through the `p4v client` application.
- **Force Refresh** considers all the changes made to project, both from PyCharm and from any other application, for example, right through `p4v client`.

1. Switch to the Version Control tool window, tab **Local**.
2. Do one of the following:
 - To run **Standard Refresh**, click the Refresh toolbar button  or press `Ctrl+F5`.
 - To run **Force Refresh**, click the Force Refresh toolbar button .

Using Subversion Integration

With the Subversion integration enabled, you can perform basic Subversion operations from inside PyCharm.

PyCharm currently supports integration with Subversion 1.9 and below.

PyCharm comes bundled with the Subversion plugin. If you are using SVN 1.7 or below, this plugin is enough for Subversion integration. If you are using SVN 1.8 or higher, you also need to [download](#) and install the command line client on your machine. In this case, make sure the Use command line client option is selected in the [Subversion settings page](#).

Subversion support

- When Subversion integration with PyCharm is enabled, the Subversion item appears on the VCS menu, and on the context menus of the [Editor](#) and [Project](#) views.
- The files in the folders under the Subversion control are highlighted according to their status. See [File Status Highlights](#) for file status highlighting conventions.
- Modifications results are shown in the [Version Control tool window](#).
- When using Subversion integration, it is helpful to open the [Version Control](#) tool window. The [Console](#) tab displays the following data:
 - All commands generated based on the settings you specify through the PyCharm user interface.
 - Information messages concerning the results of executing generated Subversion commands.
 - Error messages.

Authenticating to Subversion

The Subversion server does not require user authentication on every request. When you use Subversion integration in PyCharm, you only need to answer the authentication challenge of the server if it is required by the authentication and authorization policies. Upon successful authentication, your credentials are saved on disk, in `~/.subversion/auth/` on Unix systems or `<USER> HOME>/.subversion_IDEA` on Windows and macOS.

When an authentication challenge comes from the server, the credentials are sought for in the disk cache; if the appropriate credentials are not found, or fail to authenticate, you are prompted to specify your login and password.

If necessary, you can opt to delete all credentials stored in the cache for the http, svn and ssh+svn protocols.

To delete credentials from disk

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing `File | Settings for Windows and Linux or PyCharm | Preferences for macOS`, and click `Version Control`.
2. Open the [Subversion settings page](#) and click the `Clear Auth Cache` button.

Browsing Subversion Repository

Prior to checking files out, you can browse the contents of a Subversion repository. The Subversion Repository browser enables you to add or discard repository locations, view the history of files and folders, check out files and folders, navigate to the source code, browse changes, create branches or tags, etc.

Browsing the contents of a Subversion repository is always available, even when Subversion is not enabled in your project. All you need is a valid user account.

To browse the contents of a Subversion repository

1. On the main menu, choose **VCS | Browse VCS Repository | Browse Subversion Repository**.
2. The **SVN Repositories** tool window will open.
3. If no repositories have been specified so far, click **+** in the toolbar and specify the repository URL in the New Repository Location dialog that opens.
4. Browse the repository. If authentication is required, enter your credentials in the Authentication Required dialog that opens when you first try to expand the top node.

Checking Out Files from Subversion Repository

By checking out files from a Subversion repository, you obtain a [local working copy of the repository](#), which you can edit. After making the necessary changes, you can publish the results by [committing](#), or [checking in](#) your changes to the repository.

To check out files from a Subversion repository, do the following:

1. On the main menu, choose VCS | Checkout from Version Control | Subversion.
2. In the [Check Out From Subversion](#) dialog box, expand the desired repository location and select the element you want to check out.
3. Click the Checkout button.
4. In the [dialog that opens](#), specify the destination directory where the local copy of the repository files will be created, and click OK.
If you are checking out sources for an existing project, the destination folder should be below the project [content root](#).
5. In the [SVN Checkout Options](#) dialog box, specify the following settings:
 - Revision to be checked out (HEAD or a selected revision).
 - Whether you need to check out the nested directories.
 - Whether you need to include the external locations.

Click OK.

Tip This action is also available from the [SVN Repositories tool window](#). Right-click a directory and choose the required command from the context menu.

PyCharm suggests to create a project based on the sources checked out from version control.

If you accept the suggestion, open new project, as described in the section [Opening Multiple Projects](#).

Cleaning Up Local Working Copy

The Cleanup command in Subversion can be helpful in the following situations:

- Your local working copy is in an inconsistent state because a Subversion command was interrupted.
- The timestamp of a file has changed while its content remains intact.

To clean up the local working copy, do one of the following:

- Select the desired file or directory in the [Project](#) tool window and choose Subversion | Cleanup from the context menu of the selection.
- Open the desired file in the editor and choose VCS | Subversion | Cleanup from the main menu.
- Select the desired file or directory in the [Local Changes](#) tab of the [Version Control](#) tool window and choose Subversion | Cleanup from the context menu of the selection.

Comparing With Branch

In addition to the common file versions comparison options, the Subversion integration with PyCharm provides a special command that enables you to compare a file from your local working copy with its version in the selected branch.

To compare a file with its version in a specified branch, do the following:

1. Select the desired file in the [Project tool window](#), or open it in the editor.
2. From the main VCS menu, or on the context menu of the selection, choose Subversion | Compare with Branch.
3. In the Compare with Branch pop-up, select the desired branch. The Compare with Branch dialog in the form of the [Differences Viewer for Files](#) appears.

Tip This action is also available in the SVN Repositories browser. Right-click the desired directory and choose the corresponding command from the context menu.

Configuring the Format of the Local Working Copy

A working copy is a directory that contains a collection of files which you can use as your private work area, as well as some extra files, created and maintained by Subversion. For example, each directory in your working copy contains an administrative directory named `.svn`.

You can have local working copies created with Subversion 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, or 1.9. PyCharm handles all these formats, giving you a choice to upgrade to the new format or preserve the legacy one.

Switching to another Subversion format is always associated with a specific working copy (directory) under the Subversion control. In other words, one cannot upgrade to a newer Subversion format at the IDE level, but only to a directory under Subversion control.

Tip To check VCS association, go to [File | Settings | Version Control](#) where all mappings between the project directories (or the entire project) and VCSs are listed.

To change the format of the local working copy, do the following:

1. Open the [Version Control](#) tool window by doing one of the following:

- On the main menu, choose [View | Tool Windows | Version Control](#).
- Press `Alt+9`.

2. Switch to the [Subversion Working Copies Information](#) tab.

This tab is only available, when the current project sources are entirely or partially under Subversion control.

3. Scroll to the information on the required directory and click the [Change](#) link.

Note Note that Subversion 1.9 can be used with the local working copy version 1.8, so in this case the [Change](#) link will not appear.

4. In the [Convert Working Copy Format](#) dialog box, that opens, select the desired format option.

Configuring HTTP Proxy

Subversion stores the http proxy settings in the `servers` file in the user's runtime configuration area (`~/ .subversion/auth/` on Unix systems or `<USER HOME>/Application Data/Subversion/auth/` on Windows).

You can configure the Subversion proxy settings in two ways:

- [Edit the servers file](#) manually.
- Configure proxy settings directly from PyCharm. Do the following:
 1. In the [Settings](#) dialog, go to Version Control | Subversion and open the Network tab.
 2. Click the Edit Network Options button and specify the proxy settings in the [Edit Subversion Options Related to Network Layers](#) dialog box that opens.

Configuring Subversion Repository Location

Subversion repository locations are global PyCharm properties. It means that the configured repository locations will be available no matter if a project is open or not, which is useful if you need to check out an entire project from Subversion. You can define multiple Subversion repository locations for future use.

To configure a Subversion repository location, do the following:

1. Open the [SVN Repositories tool window](#) by choosing `VCS | Browse VCS Repository | Browse Subversion Repository` from the main menu.
2. In the SVN Repositories tool window choose `New | Repository Location` from the context menu, or click the `+` button on the toolbar.
3. In the New Repository Location dialog, specify the repository URL.

Configuring Subversion Branches

PyCharm allows you to compose a list of parent folders of the branches you work with. This list will be displayed every time you perform any operation with branches, for example, when you synchronize your local working copy, compare branches, etc.

Branches are configured in the [Configure Subversion Branches](#) dialog box.

To configure Subversion branches, do the following:

1. Access the [Configure Subversion Branches dialog](#): in the [Version Control](#) tool window, switch to the [Subversion Working Copies Information](#) tab, and then click the [Configure Branches](#) link.
2. In the Trunk location field, specify the URL of your repository trunk. Type the address or click the [Browse](#) button  and select the trunk location in the [Select Repository Location](#) dialog that opens. This dialog shows a tree of all branches and tags under the [repository root](#).
3. In the Branch locations area, make up a list of folders where the branches you need in your work are stored. Use the [+](#) and [-](#) buttons to add/remove branches to/from the list.

Creating Branches and Tags

PyCharm allows you to create branches or tags on the basis of your local working copies, or their repository versions.

To create a branch or a tag in a Subversion repository, do the following:

1. From the main menu, choose **VCS | Subversion | Branch or Tag**. Alternatively, select the source folder in the [SVN Repositories tool window](#) and choose the **Branch or Tag** command from the context menu.

2. In the [Create Branch or Tag dialog](#) that opens, in the **Copy From** section, specify the source folder that will be copied to a branch or a tag. You can use your local working copy, or a repository location.

If a repository location is selected as the source:

- Click to fill the **Repository Location** field with the path to the project location.
- Specify the revision to base the new branch on. This can be the **HEAD** revision, or a revision with the specified number. If the **Specified** option is selected, type the revision number, or click  and find the desired revision in the **Subversion Changes Browser**.

3. In the **Copy To** section, specify the destination where the branch or the tag will be created. If you use the base URL, specify the name of the new branch or tag. If you opt to create a branch or tag in another repository location, type its URL, or click  and select the destination from the [Select Repository Location](#) dialog.

4. Optionally, enter a comment and click **OK**.

Exporting Information From Subversion Repository

You might need to obtain a *clean* local copy of the Subversion working tree without the `.svn` catalogs. Instead of checking files out and then manually deleting the administrative directories, you can use the Export command available in the Subversion repository browser.

To export a directory from a Subversion repository, do the following:

1. In the main menu, select VCS | Browse VCS Repository | Browse Subversion Repository to open the the [SVN Repositories](#) tool window.
2. Right-click a directory you want to export and choose Export from the context menu.
3. In the Select Path dialog that opens, specify the destination directory and click OK.
4. In the SVN Export Options dialog that opens, check the Export and Destination paths and specify the following options:
 - Depth: use this drop-down list to specify the range of recursion into Subversion subdirectories. The available options are:
 - working copy: select this option to get files/directories from the repository subtrees that have not been checked out yet.
 - empty: select this option to involve only the current file.
 - files: select this option to involve files from the current folder.
 - immediates: select this option to involve direct children of the current file.
 - infinity: select this option to enable full recursion.
 - Replace existing files: select this option to replace files in the destination directory with the exported sources.
 - Include external locations: select this option to include external references into the export.
 - Override 'native' EOLs with:: use this drop-down list if you want to override the `svn:eol-style=native` property. This is useful if team members sharing the same repository use different operating systems, which may result in problems with line endings. The following options for line separators are available:
 - None: this option is selected by default, and keeps the `svn:eol-style=native` property unchanged.
 - LF: select this option if you are using unix
 - CRLF: select this option if you are using Windows
 - CR: select this option if you are using macOS

Importing a Local Directory to Subversion Repository

You can import an entire directory to your Subversion repository provided that you have access rights. This is helpful for putting a whole project under version control.

Import to the repository is always available, even if Subversion integration is not enabled for your project.

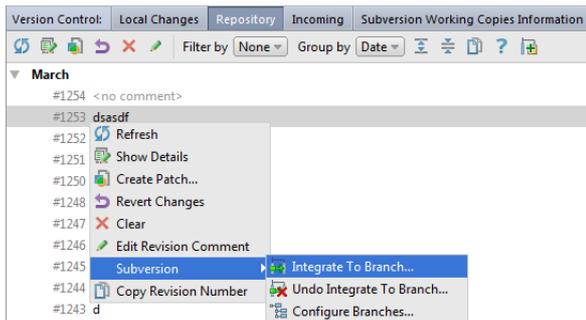
To import a directory to a Subversion repository, do the following:

1. From the main menu, choose VCS | Import into Version Control | Import into Subversion.
2. In the [Import into Subversion](#) dialog that opens, select the target Subversion repository. If the desired target repository location is not in the list, click the **+** button to add it (for more details, see [Configuring Subversion Repository Location](#)). Click Import.
3. In the Select Path dialog that opens, specify the directory you want to import and click OK.
4. In the SVN Import Options dialog that opens, check the Import to and Import from paths and specify the following options:
 - Depth: use this drop-down list to specify the range of recursion into Subversion subdirectories. The available options are:
 - working copy: select this option to get files/directories from the repository subtrees that have not been checked out yet.
 - empty: select this option to involve only the current file.
 - files: select this option to involve files from the current folder.
 - immediates: select this option to involve direct children of the current file.
 - infinity: select this option to enable full recursion.
 - Include ignored resources: select this option to include files that have been added to the Subversion ignore list and are not subject to version control into the import.
5. Optionally, enter a commit message, or select one from the Recent Messages drop-down list and click OK.

Integrating Changes to Branch

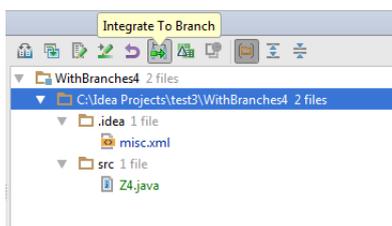
PyCharm allows you to integrate changes into a selected branch, and commit the results of such integration to the repository. Do the following:

1. In the **Version Control** tool window, switch to the **Repository** tab.
2. Right-click the changelist you want to integrate, and select **Subversion | Integrate to Branch** from the context menu:



If you want to integrate separate files instead of the whole changelist, select them in the **Changed Files** pane and choose **Subversion | Integrate To Branch** from the context menu.

If you are using SVN 1.5 or higher both on the server and in your local working copy, select the relevant changelist/file(s) in the **Changelists** or **Changed Files** pane and click  on the toolbar.



The **Integrate to Branch** dialog opens displaying the URL addresses of the source and target branches and a list of available local working copies.

3. From the **Integrate into working copy** list, select the path to the local working copy into which the selected changelist will be integrated.

To add a path to the list, click the **+** button.

To remove a path from the list, click the **-** button.

Note Make sure the specified working copy directory is under Subversion version control!

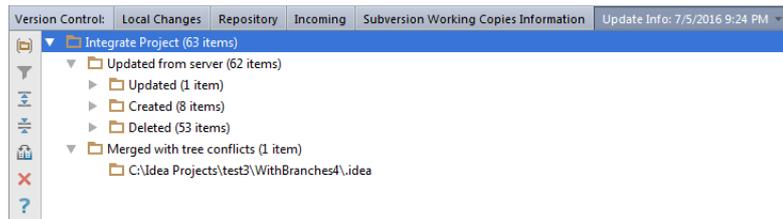
4. To preview the merge result by enabling the `--dry-run` switch of the `svn` command, select the **Try merge, but make no changes** check box. If this check box is not selected, sources are merged silently.
5. Click **OK**. The **Commit Changes** dialog opens.
6. Review the summary, specify the necessary options, and **run commit**.

Integrating SVN Projects or Directories

Integrating projects or directories in Subversion means merging the differences between the two specified revisions into your working copy.

The Integrate Project command is available for both Subversion and Perforce.

Integration results are displayed in the Update Info tab of the [Version Control tool window](#). The context menu of a file allows you to compare versions, view file history and annotations, browse changes, etc.



To integrate different sources into one Subversion project, do the following:

1. From the main menu, choose VCS | Integrate Project. The [Integrate Project](#) dialog opens.
2. If both Subversion and Perforce are used as version control systems in your project, select the Subversion tab.
3. In the Source 1 and Source 2 fields, specify the sources to be merged and select the revision. If you check the Specified option, you can click the Browse button  and select a revision from the Changes Browser.
4. If necessary, select the following merge options and click OK:
 - Use ancestry: if this option is selected, **ancestry** of files will be noticed (this corresponds to the `svn merge` command). If unchecked, any relations between files and directories will be ignored (corresponds to `svn diff`).
 - Try merge but make no changes: select this option to preview merge results by enabling the `--dry--run` option of the SVN command. If unchecked, sources will be merged silently.
 - Depth: use this drop-down list to specify the range of recursion into Subversion subdirectories. The available options are:
 - working copy: select this option to get files/directories from the repository subtrees that have not been checked out yet.
 - empty: select this option to involve only the current file.
 - files: select this option to involve files from the current folder.
 - immediates: select this option to involve direct children of the current file.
 - infinity: select this option to enable full recursion.

Locking and Unlocking Files and Folders

Though Subversion integration allows you to successfully modify and merge files changed by different team members, sometimes it makes sense to lock files (for example, images) to avoid overwriting changes.

To lock a file:

1. Select the file you want to lock in the [Project tool window](#) or open it in the editor.
2. From the main menu, select VCS | Subversion | Lock from the main menu, or Subversion | Lock from the context menu of the selection.
3. Enter a lock comment in the [Lock File](#) dialog that opens.

To unlock a file, select the file you want to unlock, or open it in the editor, and choose VCS | Subversion | Unlock from the main menu, or Subversion | Unlock from the context menu of the selection.

Tip To forcibly break a lock set by somebody else, select the Steal Existing Lock check box.

Resolving Text Conflicts

If a conflict occurs in a file under the Subversion version control, conflict markers are added to the conflicting file, and three auxiliary unversioned files are created in your local working copy:

- filename.mine: the copy of your local file without conflict markers.
- filename.rOld: the base revision you have last synchronized to.
- filename.rNew: the latest version on the server.

Conflicting files are marked with red in the [Local Changes](#) tab of the [Version Control](#) tool window. In the [Update Info](#) tab, they are grouped in the Merged with conflicts list and are also marked with red.

With PyCharm, you can resolve conflicts in two ways:

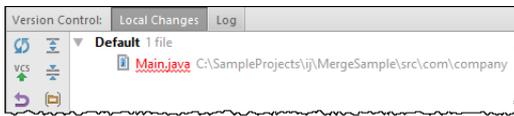
- Semi-automatically, using a merge tool.
- Manually in the editor. After that, you need to manually mark the processed files as conflicts-free.

On this page:

- [To resolve a text conflict using the merge tool](#)
- [To resolve a text conflict manually](#)
- [To mark a file as resolved](#)

To resolve a text conflict using the merge tool

1. In the [Local Changes](#) tab of the [Version Control](#) tool window, select the conflicting file:



2. On the main VCS menu, or on the context menu of the selection, choose Subversion | Resolve Text Conflict. The Files Merged with Conflicts dialog appears.
3. If you want to accept the server version and overwrite your local changes, click Accept Theirs. If you want to force your changes to the repository, click Accept Yours. Clicking Merge opens the merge tool, where you can [accept or discard](#) each change individually. As a result, the file is automatically marked as resolved, and auxiliary files are deleted.

Tip You can click the column header to sort the conflicting files by name.

4. Once the conflicts have been successfully resolved, commit your local version to the repository.

To resolve a text conflict manually

Open the conflicting file in the editor, and do one of the following:

- Edit the contents within the conflict markers as required.
- Copy one of the auxiliary files on top of your working file.

To mark a file as resolved

1. Do one of the following:

- Select the file in the [Project](#) tree or in the [Local Changes](#) tab of the [Version Control](#) tool window, and choose Subversion, and then choose Mark Resolved on the context menu of the selection.
- With the conflicting file opened in the editor, right-click the mouse anywhere in the editor tab. On the context menu choose Subversion, and then choose Mark Resolved.
- On the main menu, choose VCS | Subversion | Mark Resolved..

2. In the [Mark Resolved](#) dialog box that opens select the file in question.

3. Click the Mark Resolved button.

Sharing Directory

PyCharm supports the Subversion sharing directories feature. The **Share Directory** command is applied to the content roots. When such content root is shared, a new directory is created in the specified location of the Subversion repository, checked out to the local directory you want to share, and the contents of the local directory is added to the repository. As a result, the local directory contains a valid working copy of the repository.

To share a directory via the Subversion repository, do the following:

1. In the [Project tool window](#), select an unversioned directory that is the content root of a module associated with Subversion.
2. From the main menu, select Subversion | Share Directory.
3. Specify the target repository location where the directory shall be shared.

Viewing and Fast Processing of Changelists

If you are using Subversion 1.5 or higher on the server and in your local working copy, you can take advantage of the extended Merge Info functionality implemented through the [Merge Info](#) pane of the [Version Control tool window](#), the [Repository](#) tab.

With this functionality, instead of browsing all changelists within a certain period, you can define a set of changelists to display. This is done by specifying a pair of branches in the repository (source and target), whereupon PyCharm shows only the changelists from the source branch that have been integrated into the target branch.

Additionally, you can specify various filtering options to minimize the number of extraneous changelists.

Finally, integration and managing integration status are also available directly from the Merge Info pane.

The extended browsing functionality includes:

- [Defining the Set of Changelists to Display](#)
- [Filtering Out Extraneous Changelists](#)
- [Integrating Files and Changelists from the Version Control Tool Window](#)
- [Viewing and Managing Integration Status](#)

Tip Before enabling the extended Merge Info functionality, make sure you are using SVN Server 1.5 or higher.

If you are using SVN 1.4 or lower, to enable the Merge Info functionality, you need to [change the format of your local working copy](#) first.

Defining the Set of Changelists to Display

With the extended [Merge Info](#) functionality, you can limit the set of changelists to be displayed in the [Changelists](#) pane to changelists that have been integrated from one specific branch to another. These branches are referred to as "source" and "target" respectively.

To define the set of branches you want to display, do the following:

1. Open the [Version Control tool window](#) and switch to the [Repository tab](#).
2. To open the Merge Info pane, click the Highlight Integrated button  on the toolbar.
3. In the From field, specify the URL address of the source branch.
4. In the To field, specify the path to the target branch. If necessary, use the Browse button  to open the [Select Branch](#) dialog.
5. Specify the path to the local [working copy](#) to which you are going to apply patches created based on the selected changelists. If necessary, use the Browse button  to open the Configure Working Copy Paths dialog box.

Filtering Out Extraneous Changelists

With the extended [Merge Info](#) functionality, you can limit the number of changelists displayed in the Changelists pane by applying the following filters:

Tip Filtering out integrated/not integrated changelists is available only when integration status is highlighted.

Otherwise the Filter out integrated button  and Filter out not integrated button  are disabled. To enable integration status highlighting, click the Highlight Integrated button  on the toolbar.

- To display only the changelists that have not been integrated into the working copy, click the Filter out integrated button  on the toolbar.
- To display only the changelists that have been integrated into the working copy, click the Filter out not integrated button  on the toolbar.
- To hide the changelists that are [managed in another VCS](#) or are located under another root, click the Filter out others button  on the toolbar.
- To display only the changelists that were committed by a specific user, select User in the Filter by drop-down list. Then select the required user name.
- To display only the changelists applied to a specific module or folder, select Structure in the Filter by drop-down list. Then select the required location.
- To group changelists by users who committed them, or by commit dates, select the corresponding option in the Group by drop-down list.

Integrating Files and Changelists from the Version Control Tool Window

You can integrate changelists or files into your local working copy directly from the Version Control tool window.

Do the following:

1. In the Changelists pane, select the required changelist. If necessary, you can select several changelists at a time.
2. Do one of the following:
 - To integrate an entire changelist, click the Integrate to Branch button  on the Merge Info pane toolbar.
 - To integrate a particular file from the selected changelist, select the file in the Changed Files pane and click the Integrate to Branch button  on the Merge Info pane toolbar.
 - To revert the last integration of the selected changelist into the working copy, click the Undo Integrate to Branch button  on the toolbar.

Viewing and Managing Integration Status

With the extended [Merge Info](#) functionality, you can view and update changelists' integration status.

Subversion stores the information on whether a changelist has been integrated into the local working copy or not. Based on this data, PyCharm informs you on the integration status of a specific changelist by displaying one of the following icons next to it:

-  integrated
-  not integrated
-  integration status unknown
-  common history

To have the integration status displayed, click the Highlight Integrated button  on the Merge Info pane toolbar.

You can change the integration status of a changelist without actually integrating it into the working copy or reverting the previous integration. This will affect only the administrative data.

To toggle the integration status of a changelist, do one of the following:

- To mark a changelist as Integrated, select it and click the Mark As Merged button  on the Merge Info pane toolbar.
- To mark a changelist as Not integrated, select it and click the Mark As Not Merged button  on the Merge Info pane toolbar.

Subversion integration enables you to work with Subversion-specific properties without leaving PyCharm.

Once defined, the properties of a file or a directory are displayed in the SVN Properties view. In this view you can explore and change the existing properties and their values, or create new ones using the toolbar buttons or context menu commands.

This section describes how to:

- [View properties of a file or directory from within PyCharm](#)
- [Create a new property, or change the value of an existing property](#)
- [Set up a keyword property](#)
- [Delete a property](#)
- [Resolve property conflicts](#)

To view the properties of a file or directory

1. In the [Project tool window](#), select the desired file or directory under the SVN version control.
2. From the main VCS menu, or from the context menu of the selection, choose Subversion | Edit Properties. The SVN Properties view will open showing the properties of the selected file:
3. Use the toolbar buttons or the context menu commands to create, edit or delete properties, as described in the procedures below.

Tip If you want the SVN Properties view to preserve its contents as you navigate through your project or edit files, make sure that the Follow Selection button  is not pressed; otherwise the view will show the properties for the currently selected or edited file.

To create a new property, or set the value for an existing property

1. Open the [SVN Properties view](#).

Tip Use the Set property command on the Subversion menu to define a single property.

2. Click the add button  on the toolbar of the SVN Properties view, or choose the Add property command on the context menu. The [Set Property](#) dialog box appears.
3. In the Property name field, type the name of the new property, or select one from the drop-down list.
4. Choose the Set property value option, and specify the desired value in the text area below.
5. To apply the changes to all subdirectories of the selected directory, select the Update properties recursively check box.
6. Click OK.

To set up the svn:keywords property

- In the SVN Properties view for a file, click .
- In the SVN Keywords dialog box, check the keywords to be included in the property.
- Click OK.

To delete a property

1. Select the property you want to delete.
2. Click  on the toolbar.

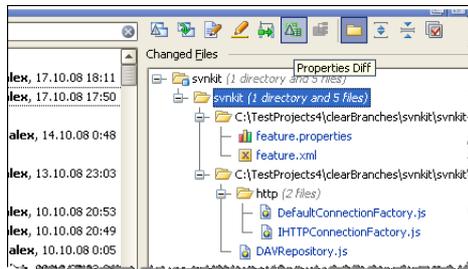
You can also delete a property from the Set Property dialog. To do this:

1. In the Property name field, select the property to be deleted.
2. Select the Delete property radio-button.
3. If you want this property to be deleted from all files and subdirectories of the selected directory, select the Update properties recursively option.

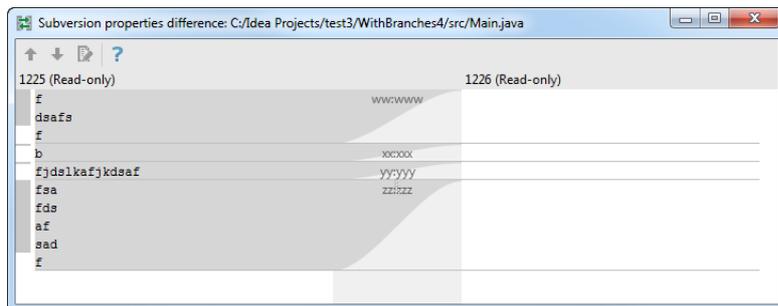
With PyCharm, you can view differences in properties between a file in the [local copy](#) and in the [repository](#) or between [two revisions](#) of a file in the local copy.

To view property difference between the local copy and the repository version

1. Open the Version Control tool window and switch to the [Repository](#) tab.
2. In the [Changed Files](#) pane, select the file for which you want to view property differences.
3. Choose Properties Diff on the context menu of the selection or click  on the toolbar.

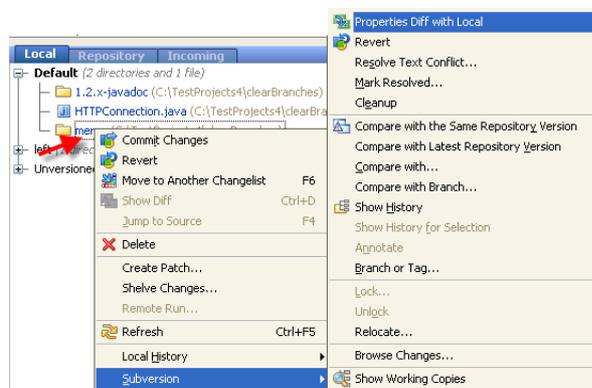


4. In the Subversion properties difference viewer, explore the differences:

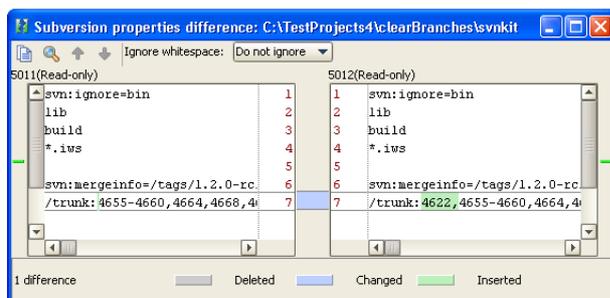


To view property difference between two revisions in the local copy

1. Open the Version Control tool window and switch to the [Local](#) tab.
2. Select the file for which you want to view property differences between revisions and choose Subversion | Properties Diff with Local in the context menu.



3. In the Subversion properties difference viewer explore the differences:



Resolving Property Conflicts

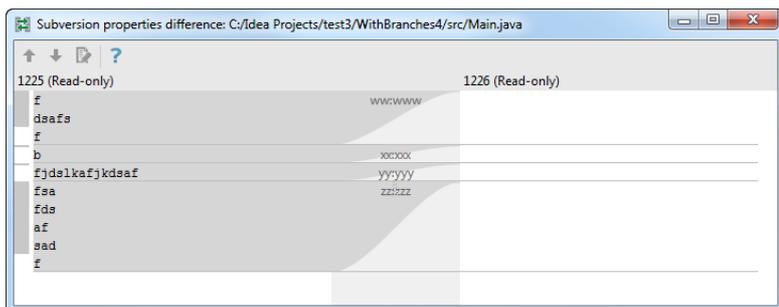
A **property conflict** is reported during synchronization with the server when PyCharm detects differences between the properties of a local file or folder and their server version. PyCharm does not attempt to resolve property conflicts automatically, and displays the conflicting files and folders under the Merged with property conflicts node in the [Update Info](#) tab of the [Version Control](#) tool window. You have to resolve property conflicts manually and then tell PyCharm to treat the corresponding files and folders as conflict-free.

On this page:

- [To resolve a property conflict](#)
- [To mark a file as resolved](#)

To resolve a property conflict

1. Open the [Version Control tool window](#) and switch to the [Repository](#) tab.
2. In the [Changed Files](#) pane, select the conflicting file.
3. Choose Properties Diff from the context menu of the selection, or click  on the toolbar.
4. Explore the differences in the Subversion properties difference viewer:



5. [Update the properties](#) so the conflict is resolved.

To mark a file as resolved

- In the [Update Info](#) tab of the [Version Control](#) tool window, select the fixed file under the Merged with property conflicts node. As you can see, the file is still displayed in red as conflicting.
- On the context menu of the selection, choose Subversion, and then choose Mark Resolved.
- When the dialog box is closed, the Local Changes tab of the Version Control tool window shows the affected files as updated and available for checking in to the server.
- [Check in](#) the resolved files.

Diagnosing Problems with Subversion Integration

If any problem occurs with Subversion integration, feel free to [contact our support](#). To facilitate detecting, locating, and resolving your issue, provide detailed information regarding your Subversion integration with PyCharm. This topic lists the necessary information and explains how you can retrieve it.

The following data is usually required to diagnose Subversion problems:

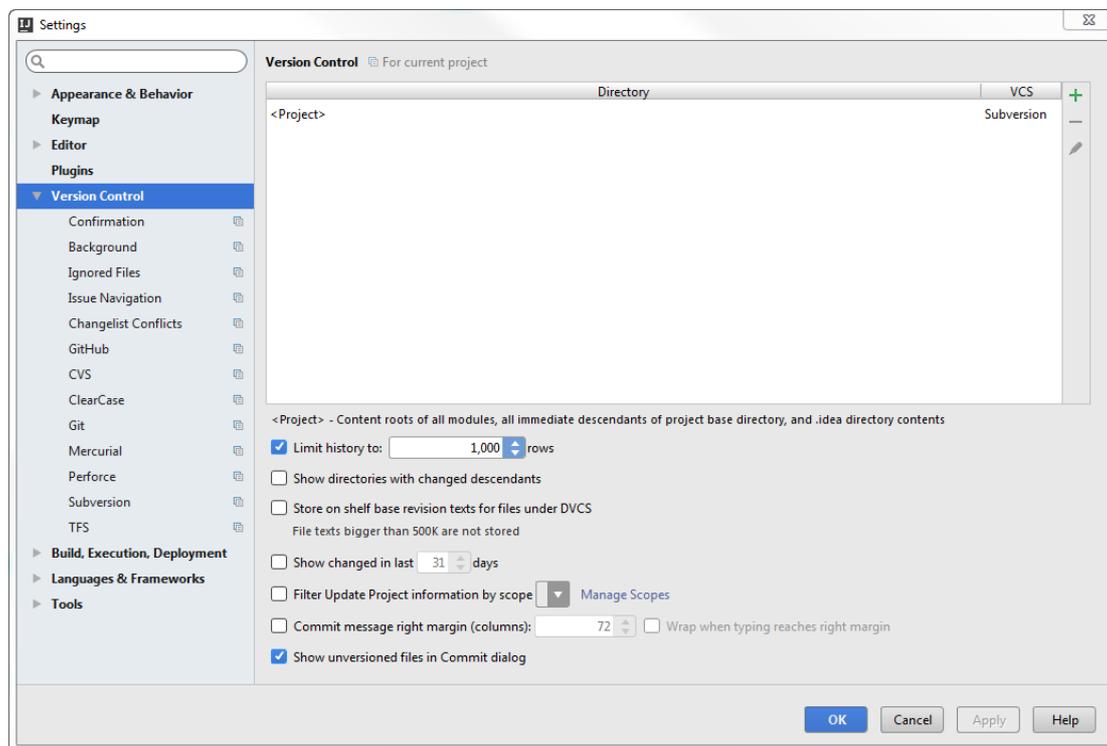
- The PyCharm version and the build number.
- The operating system used.

On this page:

- [General VCS settings](#)
- [Subversion settings](#)
- [Local working copy format](#)
- [Parent folders of the branches used](#)
- [Enabling svnkit logging](#)

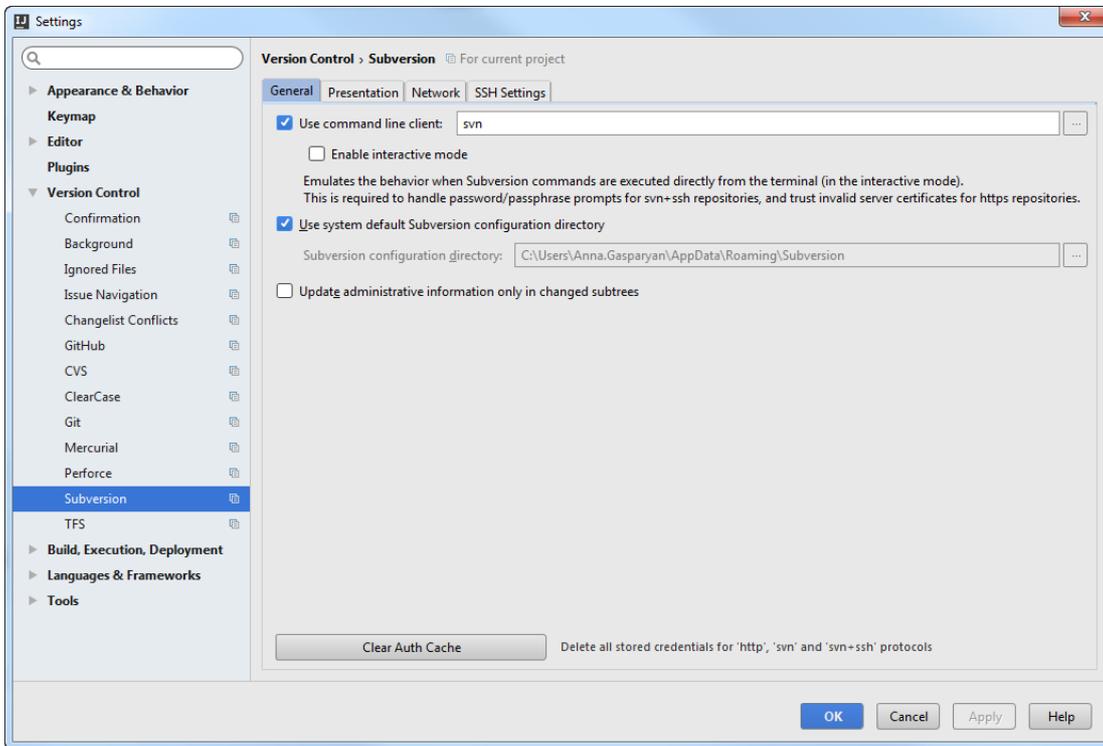
General VCS settings

The general version control settings applied to your project are specified on the [Version Control](#) page of the [Settings/Preferences dialog](#). Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Version Control.



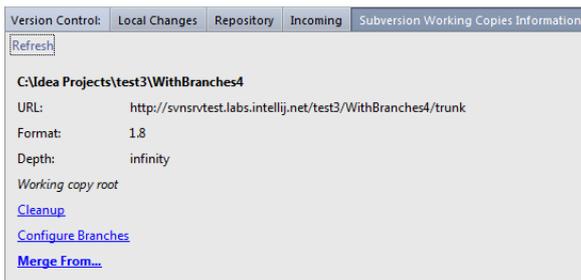
Subversion settings

Subversion-specific settings are configured on the [Subversion](#) page, under the [Version Control](#) node of the [Settings/Preferences dialog](#). Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Subversion under Version Control.



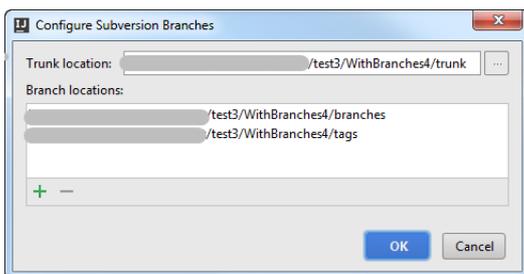
Local working copy format

The working copy format is the Subversion format in compliance with which the working copy was created. To view the working copy format, choose View | Tool Windows | Version Control on the main menu. In the Version Control tool window that opens, switch to the Subversion Working Copies Information tab.



Parent folders of the branches used

To view the configuration of branches, in the Subversion Working Copies Information tab of the Version Control tool window, click the Configure Branches link.



Enabling svnkit logging

If data saved in the PyCharm logs is not sufficient to solve the problem and the problem is related to the communication protocol or authentication, enable **svnkit logging**, reproduce the problem, and attach the `idea.log`.

To enable svnkit logging, add `-Djavasvn.log=true` to one of the following files depending on the operating system used:

- Windows: `PyCharm.exe.vmoptions`
- Linux: `PyCharm.vmoptions`
- macOS: <http://stackoverflow.com/a/13581526/72788>

Refer to [Tuning PyCharm](#) to learn about PyCharm `idea.properties` and `vmoptions` files locations.

PyCharm supports integration with the [GitHub](#) remote storage.

To use GitHub integration, perform these general steps

- [Enable the GitHub bundled plugin](#) to get access to GitHub integration.
- [Register your GitHub account](#) in PyCharm.
- [Clone repositories](#) from GitHub.
- [Publish your projects](#) on GitHub.
- [Share code snippets](#) through Git gists.

Registering GitHub Account in PyCharm

To retrieve data from GitHub repositories and share your projects in right from PyCharm, you need to [register](#) your GitHub account credentials in the IDE. You can also [create an account](#) on the GitHub free hosting without leaving PyCharm.

In either case, PyCharm remembers the login and password so you do not need to specify them while [retrieving](#) or [uploading](#) data.

To register GitHub account credentials

1. [Open the IDE Settings](#) and click GitHub.
2. On the [GitHub](#) page that opens, select the type of authentication that you want to use from the Auth Type drop-down list. The following options are available:
 - Password. If this option is selected and you have [two-factor authentication](#) enabled in your GitHub account settings, you will be asked to enter an authentication code each time PyCharm requires you to log in to your GitHub account.
 - Token (recommended by GitHub for authentication from third-party applications, as it does not require PyCharm to remember your password).
3. Specify the credentials depending on the selected authentication type and click OK.

To create a GitHub account from PyCharm

1. [Open the IDE Settings](#) and click GitHub.
2. On the [GitHub](#) page that opens, click the Sign up link.
3. On the Sign up for GitHub page that opens in the browser, choose the account plan (free or paid) and specify the requested information. When you are through with creating the account, the GitHub Welcome Page is displayed.
Free accounts do not support private repositories. Private repositories are available only for you, for other users even READ access is prohibited.
4. Return to the [GitHub](#) page and specify your GitHub account credentials in the Login and Password text boxes and click OK.

Besides the standard procedure for cloning a remote repository, PyCharm provides facilities to clone a repository located on the [GitHub](#) remote storage.

Tip Make sure, you have [registered your GitHub account credentials](#) in PyCharm.

To clone a repository from the GitHub storage

1. Choose VCS | Checkout from Version Control | GitHub on the main menu. PyCharm establishes connection with GitHub using the [login and password you registered](#). Upon establishing connection, the [Select Git Hub Repository to Clone](#) dialog box opens.
2. From the Repository drop-down list, select the source repository to clone the data from.
3. To view the details and description of the selected repository in the browser, click the Details on <repository name> here link.
4. In the Folder text box, specify the directory where the local repository for cloned sources will be set up. Type the path to the directory manually or click the Browse button  and choose the desired directory in the Select project destination folder dialog box that opens.
5. In the Project name text box, specify the name of the project to be created based on the cloned sources.
6. Click the Clone button to start cloning the sources from the specified remote repository.

Viewing the GitHub Version of a File

You can view the GitHub copy of a file right from the IDE. PyCharm detects which branch is currently active and opens the GitHub copy of the file in the corresponding branch.

To view the remote copy of a file

1. Open the file in the editor or select it in the [Project](#) tool window.
2. On the context menu, choose Open on GitHub. The GitHub copy of the file in the remote version of the current branch opens.

Note If the command is invoked on a folder, the page with the information on the corresponding remote folder is displayed.

 Make sure that you have [registered your GitHub account credentials](#) in PyCharm.

To publish your project sources on GitHub

1. On the main menu, choose VCS | Import into Version Control | Share Project on GitHub.
 - If you [have registered login and password](#), PyCharm establishes connection with GitHub using these credentials.
 - If you have not registered your GitHub credentials in PyCharm, the [Login to GitHub Dialog](#) opens. Specify your GitHub login and password or create an account there.
2. Upon establishing connection, the [Share Project on GitHub](#) dialog box opens. Specify the name of the repository to store your project sources in. By default, PyCharm suggests the name of the current project. Provide a brief description of the project functionality. Click the Share button. PyCharm initiates creation of the new repository on the GitHub and [uploads the project sources](#) to it.

Creating Gists

To share your code, you can create [gists](#) right from PyCharm. A gist can contain a code snippet, an entire file, several files, and even folders.

You can also use gists to save and share **console output** during running, debugging, testing, etc.

On this page:

- [Enabling creation of gists](#)
- [Saving code in a gist](#)
- [Saving console output in a gist](#)

Enabling creation of gists

1. Make sure the **GitHub** plugin is enabled. The plugin is bundled with PyCharm and activated by default. If the plugin is not activated, enable it on the [Plugins](#) page of the [Settings / Preferences Dialog](#) as described in [Enabling and Disabling Plugins](#).
2. Create an account on the GitHub remote storage, and [register this GitHub account](#) in PyCharm.

Saving code in a gist

1. Select the code to be shared by doing one of the following:
 - To have a piece of code from a single file shared, open the file in the editor and select the required code snippet. If no selection is made, the gist will contain the entire file.
 - To have the code from a file shared, select the file name in the Project view.
 - To create a gist from a folder, select the folder in the Project view.
 - To create a gist from several files or folders, select them in the Project view.
2. On the context menu of the selection, choose Create gist.
3. In the [Create Gist](#) dialog box, that opens, provide a brief description of the gist to be created.
4. Configure the gist's visibility.
 - By default, the new gist will be **public**, that is, visible for all registered users with your login displayed as owner. To accept this default behaviour, just click OK.
 - To create a **private** gist that will be available for you only, select the Private check box.
 - To make the gist available for all registered users without displaying your login, select the Anonymous check box.
5. Specify whether you want PyCharm to open the new gist in a browser or not:
 - To have the gist opened in the [default PyCharm browser](#) so you can edit it right now, select the Open in browser check box. When you click OK, the <https://gist.github.com/> page, that opens, shows the new gist with the selection or the entire contents of the current file. By default, the gist is named after the file the gist originates from. To update the code snippet, if necessary, click Edit.
 - If you do not want to open the gist right now, clear the Open in browser check box. PyCharm displays the URL address of the gist so you you can access it later.

Saving console output in a gist

1. During a run, debug, or test session, or when running commands in the command line mode, switch to the relevant tool window or tab that presents the output of the application or command.
2. Click the right mouse button anywhere in the tab and choose Create Gist on the context menu.
3. In the [Create Gist](#) dialog box, that opens, provide a brief description of the gist to be created and [configure the gist's visibility](#).
4. Specify whether you want PyCharm to [open the new gist in a browser or not](#).

Creating a Pull Request

Creating [pull requests](#) is a nice way to share your code with the community and to follow the collaborative development. With it you can tell others about the changes you've made and ask for comments, review, or just share the knowledge. With all this going on, a pull request appears in the original repository only after approval. A pull request can be prepared right from PyCharm without switching to the browser.

To create a pull request:

1. On the main menu, choose VCS | Git | Create Pull Request. The Create Pull Request dialog box opens.
2. In the Base Form field, specify the repository to apply the changes to. PyCharm attempts to retrieve all the relevant repositories and shows them in the drop-down list. Choose the repository from the drop-down list or click Select Other Fork and from the Form Owner drop-down list choose the name of the user who is the owner of the target repository.
3. In the Base Branch field, specify the branch to apply the changes to. Click Show Diff to view the list of commits to be included in the pull request. To view the details of a commit, select it and switch to the Log tab which shows a list of files included in the selected commit list. See also [Merging, Deleting, and Comparing Branches](#).
4. In the Title field, specify the name for your request.
5. In the Description field, optionally provide a brief description of the changes to be applied through the request.

Managing Tasks and Context

In this section:

- Managing Tasks and Context
 - [Basics](#)
 - [Prerequisites](#)
 - [Supported issue tracking systems](#)
- [Enabling Integration with an Issue Tracking System](#)
- [Deleting Tasks](#)
- [Opening and Creating Tasks](#)
- [Switching Between Tasks](#)
- [Viewing Description of a Task](#)
- [Saving and Clearing Contexts](#)
- [Switching Between Contexts](#)

Basics

PyCharm provides facilities to set up your workflow according to the issue tracking procedure accepted in your team.

You can bind your account in an issue tracker to your project, and work on it in the discourse of tasks and contexts.

- A **task** is an activity performed in PyCharm. Each task is identified by a task name.

Normally, a task correlates with an issue in your issue tracking system. This correlation is set by using the desired issue ID as the task name. When you switch between tasks, PyCharm cleans your workspace, creates a changelist for the task, and loads a stack trace, if any.

Alternatively, you can define a task yourself so it reflects an activity that is not registered in your issue tracker.

Tasks are listed in the drop-down list on the toolbar. This drop-down list is available only when you have at least one opened issue from the tracker or a self-defined task. A tracker issue has the light-violet background colour until it is opened in PyCharm. After a tracker issue is opened in PyCharm its background colour changes to white. Self-defined tasks always have the white background colour.

- A **context** is a set of files opened in the editor while working on a task or independently from it. You can switch between contexts by switching between tasks associated with them. Alternatively, you can save and clear contexts independently from any tasks.

Prerequisites

Before you start working with tasks and contexts, make sure that the Task Management plugin is enabled. The plugin is bundled with PyCharm and is activated by default. If the plugin is not activated, enable it on the [Plugins](#) page of the [Settings / Preferences Dialog](#) as described in [Enabling and Disabling Plugins](#).

Supported issue tracking systems

Currently PyCharm supports integration with the following issue tracking systems:

- [Jira](#)
- [YouTrack](#)
- [Lighthouse](#)
- [PivotalTracker](#)
- [Redmine](#)
- [Trac](#)
- [FogBugz](#)
- [Mantis](#)
- [Generic server](#)
- [Assembla](#)
- [Sprint.ly](#)
- [GitHub](#)

Enabling Integration with an Issue Tracking System

On this page:

- [Introduction](#)
- [Enabling issue tracker's integration](#)

Introduction

To access your issues in the tracking system right from PyCharm, you need to integrate your issue tracker with PyCharm. Once opened in PyCharm, an issue is considered a newly created **PyCharm task** and added to the drop-down list on the toolbar, whereupon you can switch to it from another task. In the terms of **PyCharm task**, opening an issue is referred to as [creating a task](#).

Enabling integration between PyCharm and an issue tracking system allows you to work on your project in the discourse of tasks and contexts and thus set up your workflow in accordance with the process established in your team.

The Tasks drop-down list is available only when you have at least one opened issue from the tracker or a self-defined task.

Enabling issue tracker's integration

To enable integration with an issue tracking system

1. Access the [Servers](#) dialog box. Do one of the following:
 - [Open the Settings dialog box](#). Below the Tasks node, click Servers.
 - In the [Open Task](#) dialog box, click .
 - On the main menu, choose Tools | Tasks&Contexts | Configure Servers
2. In the Servers dialog box, specify the following:
 - The URL address of your issue tracking server.
 - Your account credentials on the server in question. These credentials will be different for the different issue tracking systems.
 - Specify whether you want to access the server via proxy and specify the proxy settings.
 - To allow access to the specified server for other members of your team, select the Share URL check box.
 - To check whether the specified settings ensure successful connection to the server, click the Test button.

See [reference page](#) for the detailed description of controls.

3. Configure synchronization between PyCharm and your issue tracking system. To do so, [Open the Settings dialog box](#), and click Tasks. In the [Tasks](#) page, configure interaction between PyCharm and your tracker. Do one of the following:
 - To have PyCharm synchronize with the issue tracking system in the background on a regular basis, select the Enable issue cache check box and specify the synchronization frequency and the cache size.
No matter whether you actually request on information from your issue tracker or not, PyCharm will connect to your issue tracking system according to the specified frequency and refresh the cached issues. The advantage of this approach is that when you need to switch to a task, the up-to-date information is already at your disposal so you do not need to wait till PyCharm establishes connection with the tracker and retrieves the information.

Tip This configuration is especially recommended when working with rather "slow" issue tracking systems.

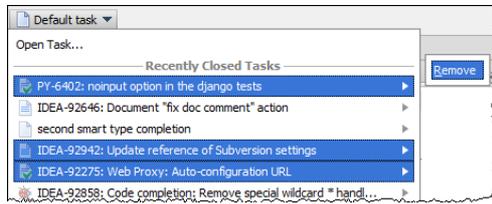
- To have PyCharm connect to the issue tracking system only when you actually need information on issues, clear the Enable issue cache check box.

Note If a server is not trusted, PyCharm shows a dialog box suggesting you to accept the server, or reject it. If you accept the server as trusted, PyCharm writes its certificate to the trust store. On the next connect to the server, this dialog box will not be shown.

Deleting Tasks

To delete a task

1. Click the tasks combo on the main toolbar.
2. Select one or more tasks you want to delete.
Use **Shift** (for adjacent items) or **Ctrl** (for non-adjacent items) keys for multiple selection.
3. Click the right-arrow button, and choose Remove from the list:



Opening and Creating Tasks

On this page:

- [Basics](#)
- [Opening tracker tasks and creating local tasks](#)
- [Tips and Tricks](#)

Basics

PyCharm distinguishes among **tracker tasks** and **local tasks**.

A **tracker task** is an issue in your tracker system until it is opened in PyCharm whereupon it becomes a **local task**.

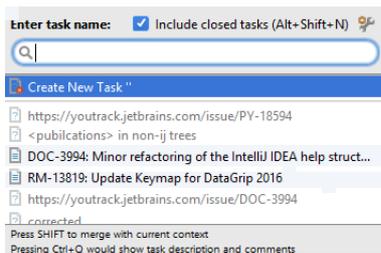
In addition to this, any activity in PyCharm can be configured as a **local task**. This action is referred to as **task creation**.

Opening tracker tasks and creating local tasks

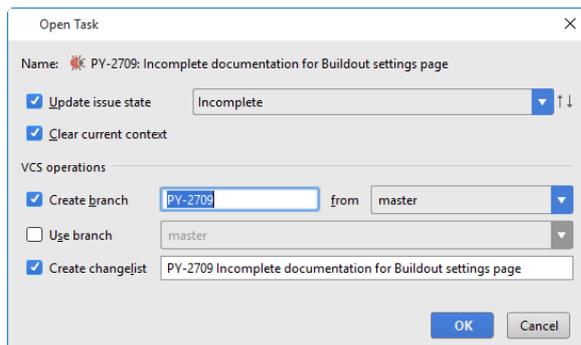
When you open a **tracker task** (an issue in your tracking system) for the first time, a corresponding **local task** is created.

To open tracker tasks and create local tasks, follow these steps

1. Do one of the following:
 - On the main menu, choose Tools | Tasks&Contexts | Open Task.
 - Press `Shift+Alt+N`
 - Click the tasks combo on the main toolbar.
2. To have the possibility to open already closed tasks, select the Include closed tasks check box. A **closed task** is one of the following:
 - A **tracker task** with the status **closed** in the issue tracker.
 - A **local task** that is not associated with a changelist provided that the entire project or the affected directory is under version control (see [Associating a Project Root with a Version Control System](#) and [Associating a Directory with a Specific Version Control System](#)).
3. In the Enter task name pop-up window, choose Create New Task, or just type the task name.



- If you choose Create new task <description>, a new local task with the specified description will be created. In this case, the IDE will switch to this new task.
- If you select a task from an issue tracker, then the Open Task dialog box opens.



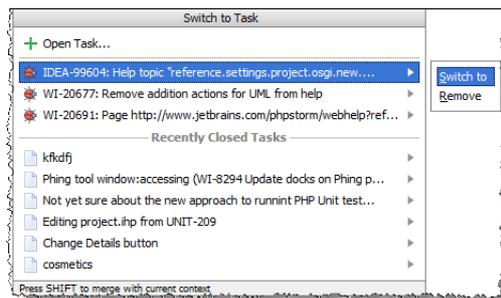
In this dialog, you can specify whether you want PyCharm to clear the current context, create a changelist for the new task, change the issue status in the tracking system, create a new VCS branch or use the existing one. Refer to the [dialog description](#) for details.

Tips and Tricks

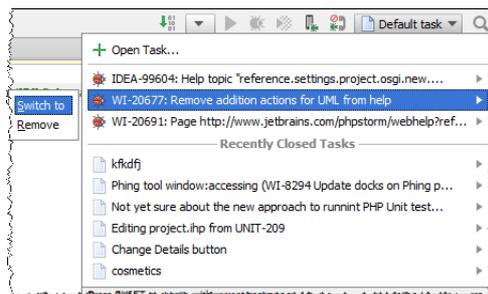
- To show task description right in the suggestion list, press `Ctrl+Q`.
- To configure access to your issue tracking system, click , and specify your account settings in the [Servers](#) dialog box that opens.
- To include closed tasks in the suggestion list, press `Shift+Alt+N` once more, or select the corresponding check box.

To switch to another task, do one of the following

- Choose Tools | Tasks & Contexts | Switch Task on the main menu. Then in the Switch to Task pop-up window, select the desired task from the list.



- Click the tasks combo in the main toolbar, select the desired task from the list, and then click Switch to on the submenu.



Viewing Description of a Task

On this page:

- [Introduction](#)
- [Viewing task descriptions](#)
- [Opening tasks in browser](#)

Introduction

When you are [choosing a task to switch to](#), the list in the Switch to task pop-up window shows only IDs of tasks and their summaries. This information is not always sufficient, because it reflects neither the steps that lead to the problem nor the related discussion.

You can learn more about your current task in two ways:

- [View the description of a task](#) in a pop-up window.
- [Open a task in the browser](#) and view both the issue description and all the comments on it.

Viewing task descriptions

To view the description of the current task in the pop-up window

- Choose Tools | Tasks&Contexts | Show '<task ID>' Description on the main menu.

Opening tasks in browser

To open the current task in browser

- Choose Tools | Tasks&Contexts | Open '<task ID>' in Browser on the main menu.

Saving and Clearing Contexts

With PyCharm, you can save and clear [contexts](#) without associating them with specific [tasks](#).

To manage the current context independently from a task, do one of the following:

- To save the current context, choose Tools | Tasks&Contexts | Save Context on the main menu.
- To clear the current context without loading another one, choose Tools | Tasks&Contexts | Clear Context on the main menu .

Switching Between Contexts

With PyCharm, you can switch between [contexts](#) that are not associated with specific [tasks](#).

To switch to another context

1. Choose Tools | Tasks&Contexts | Load Context on the main menu or press `Shift+Alt+L`.
2. In the Load Context pop-up window, select the desired context from the list.

Managing Plugins

In this section:

- [Plugins](#)
- [Enabling and Disabling Plugins](#)
- [Installing, Updating and Uninstalling Repository Plugins](#)
- [Installing a Plugin from the Disk](#)
- [Managing Enterprise Plugin Repositories](#)
- [Adding Plugins to Enterprise Repositories](#)

Plugins

On this page:

- [Basics](#)
- [Categories of plugins](#)
- [Plugin repositories](#)

Basics

Plugins are extensions to PyCharm core functionality. They provide the IDE integration with version control systems (VCS) and application servers, add support for various development technologies, frameworks and programming languages, and so on.

The more plugins are installed and enabled, the more features you have available. On the other hand, disabling unnecessary plugins may increase the IDE performance, especially on "less powerful" computers.

Certain plugins are independent, certain aren't. Dependent plugins require other plugins to be enabled.

Categories of plugins

In relation to PyCharm, plugins may be attributed to one of the following categories:

- Plugins bundled with the IDE. These plugins are installed and enabled by default. You can disable unnecessary bundled plugins, but you cannot uninstall them. See [Enabling and Disabling Plugins](#).
- Repository plugins, that is, plugins stored in [plugin repositories](#) (e.g., the JetBrains Plugin Repository). To be able to use the repository plugins, you should download and install them. See [Installing, Updating and Uninstalling Repository Plugins](#).

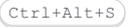
Plugin repositories

PyCharm provides access to the JetBrains Plugin Repository at <http://plugins.jetbrains.com> and to the dedicated PyCharm Plugin Repository. The PyCharm Plugin Repository resides at <http://plugins.jetbrains.com/?pycharm> and stores numerous plugins created by the PyCharm community members.

You can set up your own, enterprise plugin repositories, for example, to store plugins that you want to reserve for your company's internal use only. (A plugin repository corresponds to one or more Web servers.)

You can register such repositories in PyCharm and then work with them the same way as with the JetBrains Plugin Repository. See [Managing Enterprise Plugin Repositories](#) and [Adding Plugins to Enterprise Repositories](#).

Enabling and Disabling Plugins

1. Open the [Settings / Preferences dialog](#) (e.g. )
2. In the left-hand pane, select Plugins.
3. On the [Plugins page](#) that opens in the right-hand part of the dialog, do one of the following:
 - To enable a plugin, select the check box to the right of its name.
 - To disable a plugin, clear the corresponding check box.
4. If the plugin of interest is not present in the list (which may be the case for a [repository plugin](#)), [download and install](#) the plugin.
5. Restart PyCharm for the changes to take effect.

Installing, Updating and Uninstalling Repository Plugins

On this page:

- [Introduction](#)
- [Downloading and installing repository plugins](#)
- [Updating repository plugins](#)
- [Uninstalling repository plugins](#)

Introduction

To be able to use [repository plugins](#), you have to download and install such plugins first. After that, you get the ability to update these plugins when appropriate.

When a repository plugin becomes unnecessary, you can disable or uninstall it.

Downloading and installing repository plugins

To download and install a repository plugin

1. [Open the Settings dialog](#) (e.g. `Ctrl+Alt+S`).
2. In the left-hand pane, select Plugins.
3. On the [Plugins page](#) that opens in the right-hand part of the dialog, click the Install JetBrains plugin or the Browse repositories button.
4. In the dialog that opens (the [Browse JetBrains Plugins dialog](#) or the [Browse Repositories dialog](#)), right-click the required plugin and select Download and Install.
Note that when looking for the plugin of interest, you can filter the plugin list and also to perform a search.
5. Confirm your intention to download and install the selected plugin.
6. Click Close.
7. Click OK in the Settings dialog and restart PyCharm for the changes to take effect.

Note that the plugin you have installed is automatically enabled. When necessary, you can disable it as described in [Enabling and Disabling Plugins](#).

Updating repository plugins

To update a repository plugin

1. [Open the Settings dialog](#) (e.g. `Ctrl+Alt+S`).
2. In the left-hand pane, select Plugins.
The [Plugins page](#) opens in the right-hand part of the dialog.
3. Right-click any of the plugins and select Reload List of Plugins.
The names of repository plugins that have newer versions available are shown blue.
4. Right-click the necessary plugin and select Update Plugin.
5. Click OK in the Settings dialog and restart PyCharm for the changes to take effect.

Uninstalling repository plugins

To uninstall a repository plugin

1. [Open the Settings dialog](#) (e.g. `Ctrl+Alt+S`).
2. In the left-hand pane, select Plugins.
3. On the [Plugins page](#) that opens in the right-hand part of the dialog, right-click the repository plugin to be uninstalled and select Uninstall.
4. Confirm your intention to uninstall the selected plugin.
5. Click OK in the Settings dialog and restart PyCharm for the changes to take effect.

Installing a Plugin from the Disk

Warning! Before installation, make sure that the desired plugin has already been downloaded and saved on your computer as `.zip` or `.jar` file.

To install a plugin from the disk

1. Open the Settings/Preferences dialog box and select Plugins on the left pane.
2. On the right pane of the dialog, click the Install plugin from disk button.
3. In the [dialog](#) that opens, select the desired plugin. You can quickly locate and select the necessary file if you drag-and-drop the corresponding item from your file browser (Explorer, Finder, etc.) into the area where the tree is shown. Click OK to proceed.
4. Click Apply button of the Settings/Preferences dialog.
5. Following the system prompt that appears, restart PyCharm to activate the installed plugin, or postpone it, at your choice.

Managing Enterprise Plugin Repositories

To be able to use [plugin repositories](#) other than the JetBrains Plugin Repository (e.g., your enterprise plugin repositories), you should specify the [URLs](#) of such repositories in PyCharm.

To manage the list of enterprise plugin repositories

1. In the Settings dialog, click [Plugins](#).
2. Click Browse repositories.
3. In the [Browse Repositories dialog](#) that opens, click Manage repositories.
4. Use the [Custom Plugin Repositories dialog](#) that opens, to manage the list of URLs for custom (enterprise) plugin repositories:
 - To add a repository URL, click **+** ([Alt+Insert](#)). In the Add Repository dialog, specify the repository URL and click OK. (You can use the Check Now button to make sure that the specified URL is correct: PyCharm will try to connect to the repository.)
 - To edit a repository URL, select the URL and click **↩** ([Enter](#)). In the Edit Repository dialog, edit the URL and click OK.
 - To remove a URL from the list, select the URL and click **-** ([Alt+Delete](#)).

Click OK in the Custom Plugin Repositories dialog.

5. Click Close in the Browse Repositories dialog.
6. Click OK in the Settings dialog.

As an alternative, you can specify the list of URLs for the enterprise plugin repositories in one of the following files:

- `idea.properties`
In this case, the following line should be added to this file:

```
idea.plugin.hosts=[URL1];[URL2];...[URLn]
```

- `PyCharm.exe.vmoptions`
In this case, the following line should be added to this file:

```
-Didea.plugin.hosts=[URL1];[URL2];...[URLn]
```

Both files reside in the `bin` directory under the PyCharm installation folder.

Note that a repository added via this property is not visible in the Manage Repositories dialog.

Adding Plugins to Enterprise Repositories

To be available to PyCharm users, all [enterprise repository plugins](#) must be listed in the file `updatePlugins.xml` along with their URLs and version numbers. This file must be available at the URL specified for the corresponding repository (see [Managing Enterprise Plugin Repositories](#)).

The plugins themselves are identified by their individual URLs and thus may be located on different Web servers.

- [Adding a plugin to an enterprise plugin repository](#)
- [DTD for updatePlugins.xml](#)

To add a plugin to an enterprise plugin repository

1. Upload your plugin JAR onto a Web server.
2. Add the plugin definition to `updatePlugins.xml`. If this file doesn't yet exist, create it at the location corresponding to the repository URL. The plugin definition in `updatePlugins.xml` may look similar to this:

```
<plugins>
...
  <plugin id="MyPlugin" url="http://plugins.example.com:8080/myPlugin.jar" version="1.0"/>
...
</plugins>
```

3. To publish a new version of the same plugin, upload the corresponding plugin JAR to the repository, and change the value of the `version` attribute in the plugin definition.

DTD for updatePlugins.xml

The file `updatePlugins.xml` must correspond to the following [Document Type Definition](#) (DTD):

```
<!DOCTYPE plugins [
  <!ELEMENT
    plugins(plugin)*>
  <!ELEMENT
    plugin (#PCDATA)>
  <!ATTLIST
    plugin id CDATA #REQUIRED url DATA #REQUIRED version CDATA #REQUIRED]>
```

Comparing Files and Folders

PyCharm helps explore differences in the various situations: differences between files, directories, revisions of the same file under version control or in the local history, database objects, local and remote files.

All these operations are performed in a similar way. In this section we'll consider the most basic operations:

- [Comparing Files](#)
- [Comparing Folders](#)

Comparing Files

On this page:

- [Introduction](#)
- [Comparing two files](#)
- [Comparing a file in the editor with the Clipboard contents](#)
- [Comparing a file with the editor contents](#)

Introduction

PyCharm enables you to compare arbitrary files in project (including image files), selected file with the editor, or compare a file in the editor with the Clipboard contents. All comparisons are performed in the [Differences viewer](#).

Comparing two files

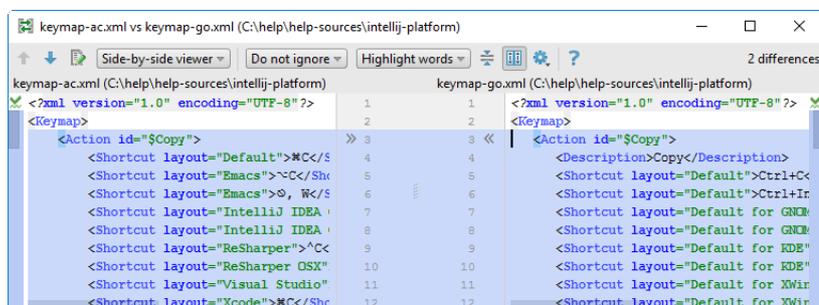
1. Press and keep holding `Ctrl` for Windows and Linux or `⌘` for macOS and click the two files to compare in the Project tool window.
2. On the context menu of the selection, choose Compare Files , or press `Ctrl+D` . The [Differences Viewer for Files](#) opens, with the differences being color-highlighted.

Tip It is enough to select a single file in the Project tool window. In this case the context menu command is Compare File with Editor, and the Differences Viewer shows the contents of the selected file on the left pane, and the contents of the active editor tab on the right pane.

3. View the differences and apply them, if necessary, using the **chevron** buttons `»»` <<

Note that keeping `Ctrl` (for Windows and Linux) or `⌘` (for macOS) pressed turns the chevron buttons `»»` to `»`. Click these buttons to append changes.

Keeping `Shift` pressed turns the chevron buttons `»»` to `x`. Click this button to revert changes.



Comparing a file in the editor with the Clipboard contents

1. Open the desired file in the editor.
2. Right-click the editor pane and choose Compare with Clipboard on the context menu.
3. View and manage differences in the [Differences Viewer for Files](#).

Comparing a file with the editor contents

1. Right-click the desired file in the Project tool window.
2. Choose Compare File with Editor on the context menu.
3. View and manage differences in the [Differences Viewer for Files](#).

Comparing Folders

On this page:

- [Basics](#)
- [Opening the Difference Viewer](#)
- [Comparing two folders in the Difference Viewer](#)
- [Synchronizing contents of folders](#)

Basics

PyCharm provides a dedicated [Differences Viewer for Folders](#) for comparing files in two folders against the file size, content, or timestamp. The Differences Viewer shows the contents of the selected directories in the left and right panes of the Item List. The contents of the selected file are shown in the lower pane, with the differences being color-highlighted.

Besides exploring differences, the tool also provides interface for synchronizing the contents of folders.

Opening the Difference Viewer

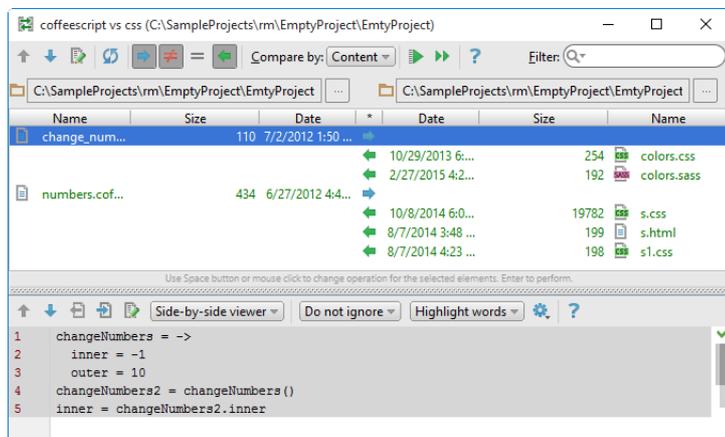
Do one of the following:

- Keeping the **Ctrl** key pressed, click two directories in the Project tool window, and choose Compare Directories on the context menu of the selection, or press **Ctrl+D**.
- Select a directory in the Project tool window, choose Compare with on the context menu of the selection, or press **Ctrl+D**, and then select the second directory in the [dialog that opens](#).

Tip You can also open the difference viewer without running PyCharm. This is done through the following command: `<path to PyCharm executable file> diff <path_1> <path_2>` where `path_1` and `path_2` are paths to the folders in question.

Comparing two folders in the Difference Viewer

1. Configure the layout of the Items List. Use the toolbar buttons to narrow down or widen the set of items to show. For example, show or hide files that exist in just one of the directories, equal files, or different files, etc.
2. Specify the parameter for comparison. In the Compare by drop-down list, select one of the possible options (contents, size, or time stamp).
3. Filter the folders' contents. To do that, type filtering string in the Filter text field, and press **Enter** to apply it. Using the asterisk ***** wildcard to represent any number of characters is welcome.
4. To switch to another pair of folders to compare, update the fully qualified paths to them. Click the Browse button  next to the Paths read-only fields and choose the required folders in the [dialog box that opens](#).
5. Explore the detected differences between files in the Differences Pane.



Synchronizing contents of folders

1. For each pair of items, in the * field specify the action to apply. Click the icon in the field until the required action is set.

IconAction

-  Copy the item in the left side to the right side, possibly overwriting the contents of the corresponding target item, if it already exists.
-  Copy the item in the right side to the left side, possibly overwriting the contents of the corresponding target item, if it already exists.
-  The items are treated identical with regard to the selected criterion of comparison. No action will be performed by default.
-  The items differ with regard to the selected criterion of comparison. No action will be performed by default. Explore the differences in the [Differences Pane](#) and change the intended action by clicking the icon.
-  The item is present only in one of the folders and will be removed.

2. Do one of the following:

- To synchronize the currently selected item, click the Synchronize Selected button  on the toolbar.
- To synchronize all the items, click the Synchronize All button  on the toolbar.

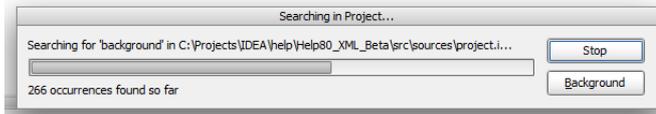
Working with Background Tasks

On this page:

- [Introduction](#)
- [Viewing background tasks](#)

Introduction

When lengthy tasks are running, for example, search and replace, VCS update, etc., PyCharm displays the progress bar. You can bring execution of such tasks to the background by clicking the Background button.

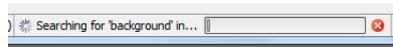


However, when a task of a certain type is brought to the background, you can still view its progress.

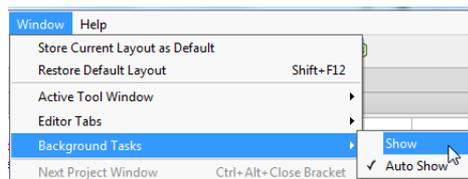
Viewing background tasks

To view tasks running in the background, do one of the following

- View the Status bar:



- In the [Status bar](#), click .
- On the main menu, choose Window | Background Tasks | Show.



As a result, the Background Tasks manager opens, showing all the tasks that are currently running in the background. You can review their progress, and if necessary, cancel the unnecessary tasks by clicking .



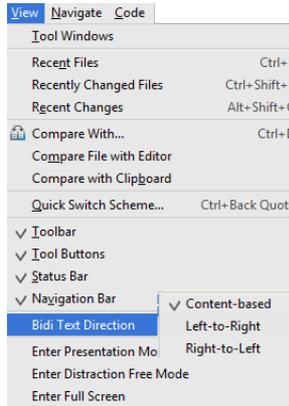
When the Auto Show command is checked, the Background Tasks manager opens automatically every time a task is launched in the background.

Text Direction

PyCharm enables you to choose the base direction to render the strings and tokens, containing bidirectional text, like a mix of English and Hebrew or Arabic.

To choose the direction of rendering strings

1. On the menu View, point to the node BiDi Text Direction.
2. Choose the required direction:



By default, the direction of rendering is content-based, which means that text direction is defined by the text with which the string begins. For example, if the string starts with English, then the base direction of text is LTR.

However, it is also possible to always use either LTR or RTL as the base direction.

So doing, the string literals change their view:

Left to Right		Right to Left	
1	key1 = text	1	key1 = text
2	value = عنوان URL	2	value = URL عنوان

Working with Consoles

PyCharm enables you to use interactive consoles, thus making it possible to stay within the IDE, without the necessity to switch to the shell.

- [Running Console](#)
- [Configuring Output Encoding](#)
- [Configuring Color Scheme for Consoles](#)
- [IPython](#)
- [Loading Code from Editor into Console](#)
- [Using Consoles](#)
- [Working with Embedded Local Terminal](#)

Running Console

On this page:

- [Introduction](#)
- [Launching console](#)

Introduction

The consoles built in PyCharm completely correspond to the shell consoles.

Besides the standard functionality, these consoles feature:

- Code completion.
- Syntax check with inspections.
- Automated insertion of paired brackets, quotes and braces.
- Scrolling through the history of commands using the arrow keys.
- Quick documentation lookup `Ctrl+Q`

Launching console

To launch an interactive console

- On the main menu, choose any console-related command from the Tools menu .

Tip When the current project is a Django project, the console that starts is Django Console. However, if the current project is pure Python project, then PyCharm starts Python Console.

Configuring Output Encoding

PyCharm creates files using the IDE encoding defined in the [File Encodings](#) page of the Settings dialog, which can be either system default, or the one selected from list of available encodings. Output in the consoles is also treated in this encoding.

It is possible that encoding used in the console output is different from the IDE default. To have PyCharm properly parse text in the console, you have to do some additional editing.

To set up encoding for the console output, depending on your operating system:

– In Windows and Linux:

Open for editing `PYCHARM_HOME/bin/pycharm.exe.vmoptions`

or

`PYCHARM_HOME/bin/pycharm.vmoptions`

respectively, and add the following line at the bottom:

```
-Dconsole.encoding=<encoding name>
```

For example:

```
-Dconsole.encoding=UTF-8
```

– In macOS: Open `Info.plist` located in `/Applications/RubyMine.app/Contents`, locate the tag `<key>VMOptions</key>`, and modify it as follows:

```
<key>VMOptions</key>
<string>-Xms16m -Xmx512m -XX:MaxPermSize=120m
  -Xbootclasspath/p:../lib/boot.jar -ea
  -Dconsole.encoding=<encoding name>
</string>
```

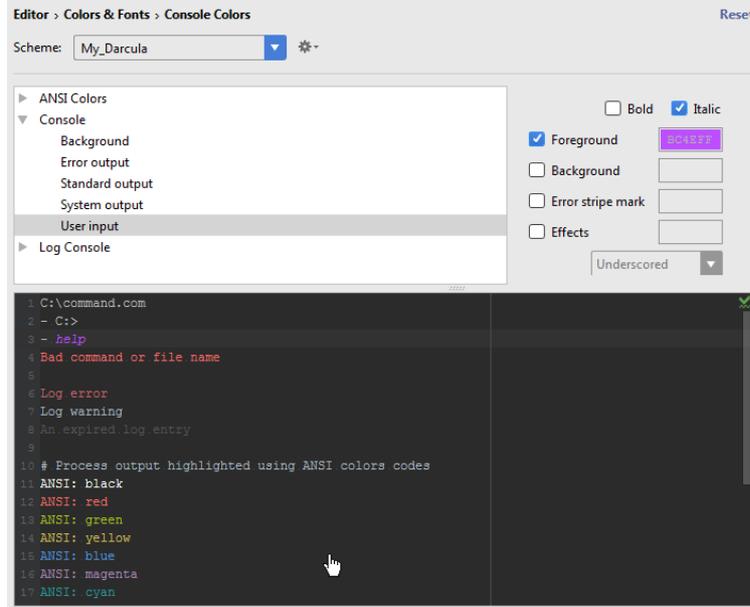
Configuring Color Scheme for Consoles

PyCharm enables you to define your habitual color scheme for the various types of consoles. So doing, you can individually configure all sorts of console output and user input.

Color scheme includes numerous colors for background, user input, system output, and error output.

To configure color and font scheme for consoles

1. Make sure you are working with an editable scheme.
2. Open the Settings/Preferences dialog, and under **Colors&Fonts**, scroll through the list of components, and select the ones related to consoles:
 - Console Colors
 - Console Font
3. In the right-hand pane, click the desired component in the list, and change color settings and font type:



IPython

PyCharm supports usage of [IPython](#) magic commands. Prior to start working, consider the following prerequisites:

- IPython is [downloaded and installed](#) on your computer.
- The Python interpreter, for which IPython has been installed, is the [default interpreter for the current project](#).

To use the magic commands of IPython, follow these general steps

1. On the main menu, choose Tools | Run Python Console. If IPython has been properly installed, PyCharm will recognize it automatically, and will report about the version used.
2. In the lower part of the console, start typing the magic commands, and press `Enter` to execute them. Refer to the section [Using Consoles](#) for the list of available actions.

Loading Code from Editor into Console

To run source code from the editor in console

1. Open file in the editor, and select a fragment of code to be executed.
2. On the context menu of the selection, choose Execute selection in console, or press `Enter`.

Note With no selection, the command changes to Execute line in console. Choose this command on the context menu, or press same keyboard shortcut. The line at caret loads into the Python console, and runs.

In an interactive console, you can

- Type commands in the lower pane of the console, and press `Enter` to execute them. Results are displayed in the upper pane.
- Use [basic code completion](#) `Ctrl+Space`.
- Use Up and Down arrow keys to scroll through the history of commands, and execute the required ones.
- [Load](#) source code from the editor into console.
- Use context menu of the upper pane to copy all output to the clipboard, compare with the current contents of the clipboard, or remove all output from the console.
- Use the toolbar buttons to control your session in the console.
- Configure color scheme of the console to meet your preferences. Refer to the section [Configuring Color Scheme for Consoles](#) for details.

Working with Embedded Local Terminal

On this page:

- [Prerequisites](#)
- [Overview](#)
- [Configuring embedded local terminal](#)
- [Running embedded local terminal](#)
- [Actions available in the embedded local terminal](#)
- [Example](#)

Prerequisites

Before you start working with terminal, make sure that the Terminal plugin is enabled. The plugin is bundled with PyCharm and is activated by default. If the plugin is not activated, enable it on the [Plugins](#) page of the [Settings / Preferences Dialog](#) as described in [Enabling and Disabling Plugins](#).

Overview

PyCharm features a local terminal that makes it possible to access the command line. Depending on your platform, you can work with command line prompt, Far, `powershell`, `bash`, etc. Using the terminal, you can execute any command without leaving the IDE.

Configuring embedded local terminal

The terminal settings are configurable on several pages of the Settings/Preferences dialog.

To configure the embedded local terminal options

1. In the Settings/Preferences dialog box, open the [Terminal](#) page, and specify the following:
 - The desired shell that will run by default, the name of a session tab
 - Name of the tab a new session will be opened in, possibility to copy to clipboard etc.
 - Ability to override the PyCharm keymap.
2. The following settings are inherited by the embedded local terminal from the IDE settings/preferences:
 - On the [Keymap](#) page, you can configure the `Ctrl+C` and `Ctrl+V` shortcuts.
 - On the editor's [Appearance](#) page - anti-aliasing and caret blinking.

Note that the setting Use block caret is not inherited in the terminal - its caret is always block.

- Under the editor's [Color and Fonts](#) settings, you can change the following options:
 - On the Console Fonts page - line spacing, and console fonts.
 - On the Console Colors page - console colors.
 - On the General page - selection foreground and background colors.

Running embedded local terminal

To run the console, do one of the following

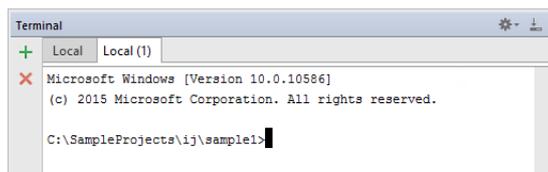
- Press `Alt+F12`.
- Click the Terminal tool window button  [Terminal](#).
- Hover your mouse pointer over  in the lower left corner of the IDE, then choose Terminal from the menu, as described in the section [Tool window quick access](#).

Actions available in the embedded local terminal

In the embedded local terminal, you can do the following:

- To create a new session:
 - Click  on the toolbar of the terminal.
 - Right-click a session tab, and then choose Create new session on the context menu.

A new session is presented in a separate tab:



- Rename a tab. Right-click a tab, and choose Rename tab on the context menu.
- Close an active session that currently has the focus. This can be done in a number of ways:
 - Click  on the terminal toolbar.
 - Right-click a session tab, and then choose Close session on the context menu.
- Use up and down arrows on your keyboard to browse through the history of entered commands.
- Toggle between the embedded local terminal and editor by pressing `Alt+F12` (View | Tool Windows | Terminal).

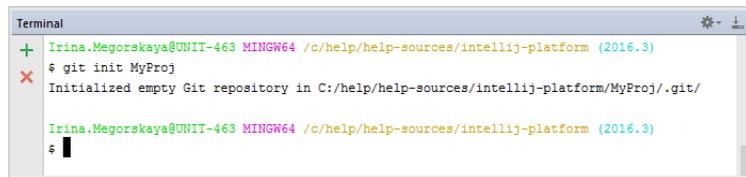
Example

Open the [Terminal](#) page of the Settings/Preferences dialog, and configure the Shell path field as follows:

```
"[path to the git installation]\bin\sh.exe" -login -i
```

Do not forget the quotes around the command!

Now, when you open the new terminal in PyCharm, it will recognize `git` commands:



```
Terminal
+ Irina.Megorskaya@UNII-463 MINGW64 /c/help/help-sources/intellij-platform (2016.3)
$ git init MyProj
X Initialized empty Git repository in C:/help/help-sources/intellij-platform/MyProj/.git/

Irina.Megorskaya@UNII-463 MINGW64 /c/help/help-sources/intellij-platform (2016.3)
$
```

Working with Diagrams

This feature is supported in the Professional edition only.

In this part:

- Working with Diagrams
 - [Prerequisites](#)
 - [Basics](#)
 - [Features](#)
- [Configuring Default Settings for Diagrams](#)
- [Viewing Diagram](#)
- [Adding Node Elements to Diagram](#)
- [Deleting Node Elements from Diagram](#)
- [Viewing Changes as Diagram](#)
- [Viewing Class Hierarchy as a Class Diagram](#)
- [Viewing Members in Diagram](#)
- [Navigating to Source from Diagram](#)
- [Navigating Through a Diagram Using Structure View](#)

Prerequisites

Before you start working with UML diagrams, make sure that the UML Support plugin is enabled. The plugin is bundled with PyCharm and is activated by default. If the plugin is not activated, enable it on the [Plugins](#) page of the [Settings / Preferences Dialog](#) as described in [Enabling and Disabling Plugins](#).

Basics

UML model in PyCharm is represented by a Class diagram in standard notation.

PyCharm enables using UML class diagrams to analyze Python classes, and the structure of the databases and tables. Besides that, you can explore changes committed to VCS.

Features

In PyCharm, Class diagram features:

- Ability to view UML model as a diagram in a [separate editor tab](#), in a [pop-up window](#), or as a [preview](#).
- Ability to invoke UML class diagram from the tool windows, the History tab of the Version Control tool window, and Navigation bar.
In the editor, one can view class diagram for the whole class.
- Navigation from a diagram element to the [underlying source code](#).
- Navigation to [class, file or symbol by name](#) and to the [last edit location](#).
- Viewing information at the tooltip, and [quick documentation lookup](#).
- [Viewing changed classes](#) as a UML Class diagram.
- Quick hierarchy view in a [UML Class diagram pop-up window](#).
- Ability to [find usages](#) of a node element or member.

Configuring Default Settings for Diagrams

Default options for the UML Class diagram help define the elements to be shown in diagram, visibility level of the node elements and members, layout, and more. These settings apply to any newly created UML Class diagram. Once a UML Class diagram is open, you can change its display settings as required, using the diagram toolbar and context menu.

Refer to the section [Diagrams](#) for the detailed description of controls.

To configure default settings for diagrams

1. Open [Settings](#), and under the Tools node, click Diagrams.
2. In the [Diagrams](#) page of the Settings dialog box, choose node you want to configure settings for, and then select the check boxes to define the elements to be shown or hidden in diagram, for example, class members, or visibility level.
3. Select the check boxes that define the Class diagram appearance and behavior: default layout, behavior after layout, and the possibility to view colored links.
4. Apply changes.

Viewing Diagram

On this page:

- [Prerequisites](#)
- [Basics](#)
- [Prerequisites](#)
- [Opening a UML class diagram](#)
- [Tips and tricks](#)

Prerequisites

Before you start working with UML diagrams, make sure that the UML Support plugin is enabled. The plugin is bundled with PyCharm and is activated by default. If the plugin is not activated, enable it on the [Plugins](#) page of the [Settings / Preferences Dialog](#) as described in [Enabling and Disabling Plugins](#).

Basics

You can invoke UML class diagram from different places:

- From the various tool windows.
- From the Navigation bar.
- From the Structure tool window.
- From the editor.

PyCharm displays UML diagrams in two modes:

- In a pop-up window.
- In a separate editor tab.

PyCharm makes it possible to choose diagram type:

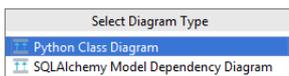
- Python class diagram
- Django model dependency diagram (for the Django projects).
- Google App Engine model dependency diagram (for the Google App Engine projects).

Prerequisites

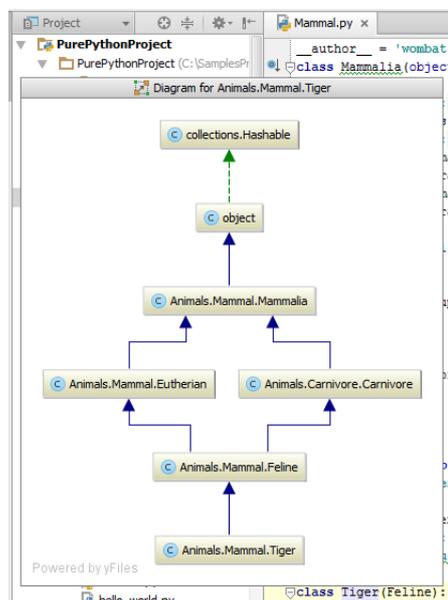
Before you start working with UML diagrams, make sure that the UML Support plugin is enabled. The plugin is bundled with PyCharm and is activated by default. If the plugin is not activated, enable it on the [Plugins](#) page of the [Settings / Preferences Dialog](#) as described in [Enabling and Disabling Plugins](#).

Opening a UML class diagram

1. Select the desired item, or open it in the editor.
2. Do one of the following:
 - On the context menu of the selection, click **Diagram**, and on the submenu, select the way you want to view the model: **Show Diagram** or **Show Diagram Pop-up**.
 - Press `Ctrl+Shift+Alt+U` or `Ctrl+Alt+U`.
3. From the pop-up list, choose the diagram type:

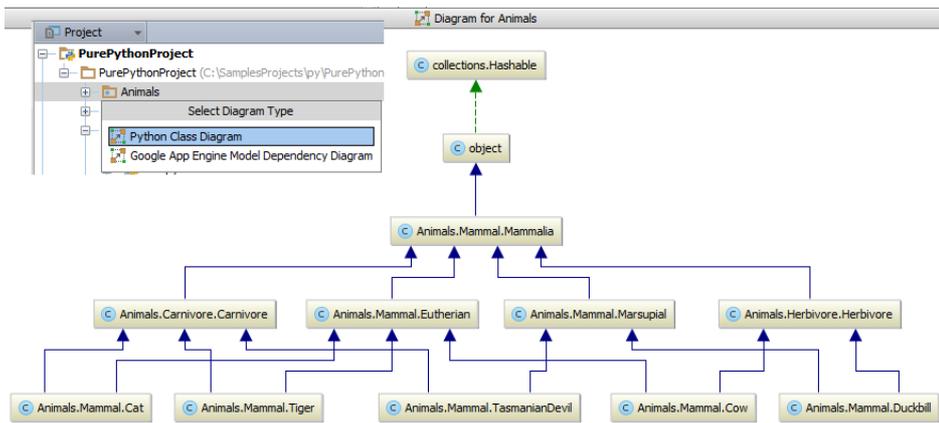


The diagram is displayed in the editor tab or in the pop-up window:



Tips and tricks

- You can open a UML class diagram without using your pointing device. Consider such a workflow: press `Alt+Home`, then press `Ctrl+Alt+U`.
- It is possible to view a UML class diagram of a Python package. Just select a Python package in the Project tool window, and press `Ctrl+Alt+U`:



Adding Node Elements to Diagram

You can add the existing node elements to the background of the UML Class diagram, using the context menu action, or drag-and-drop technique.

If there are relationships between the node elements in model, and the added element, these relationships will be displayed in diagram.

On this page:

- [Adding node elements](#)
- [Adding notes](#)

To add a node element to a UML Class diagram

1. In the Enter class name to add dialog box, start typing the desired class name. PyCharm automatically displays suggestion list with matching names.

Note that you can include classes outside the scope of your project, by selecting the Include non-project classes check box.

2. Select the desired class from the suggestion list, and press `Enter`.

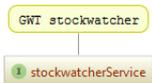
To add a note to a node element

1. Right-click a node element, which you would like to comment.

2. On the context menu, choose New | Note.

3. In the text box, type the desired text. Note that `Enter` starts a new line in the text box. To complete a note, click OK, or press

`Ctrl+Enter`.



Deleting Node Elements from Diagram

To remove elements from view

1. In the diagram, select one or more elements to be deleted.
2. Press .

Viewing Changes as Diagram

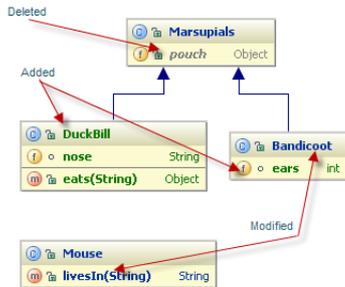
For the modules under version control, you can usually view the list of changed files in the Changes tool window. If you want to evaluate how your changes affect the model, use UML Class diagram. This view presents the complete picture of changes, including relationships between the modified classes.

For this purpose, PyCharm provides the action [Show changed classes](#), which is available from the editor, Project tool window, and the Local Changes tab of the Version Control tool window.

Moreover, PyCharm helps you analyze how the changes affect a model across revisions, and provides the action [Compare all classes from revision on UML](#).

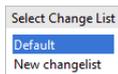
Changes in Class diagram are color-coded:

- Green for added elements.
- Gray for deleted elements.
- Blue for modified elements.



To view changes in UML Class diagram

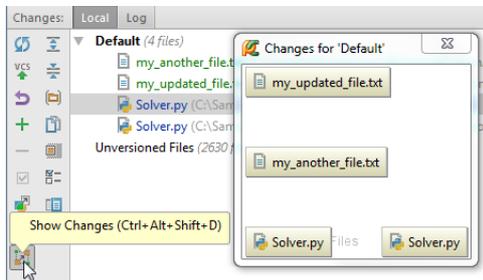
1. In the [Local Changes](#) tab of the Version Control tool window, select the desired changelist from the suggested popup:



Note that if there is only one changelist, the popup won't show up.

2. Do one of the following:

- In the toolbar of the [Local](#) tab, click , or press `Ctrl+Shift+Alt+D`.
- On the context menu of the editor or the Project tool window, choose [Diagrams | Show Changes](#), or press `Ctrl+Shift+Alt+D`.



The diagram opens in a pop-up window. Double-click a node to view changes in a Differences viewer.

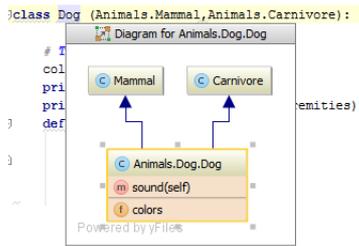
To view changes in revisions as UML Class diagram

1. In the [History](#) tab of the [Version Control tool window](#), select the desired revision.
2. Click , or press `Ctrl+Shift+D`. The diagram opens in a pop-up window.

Viewing Class Hierarchy as a Class Diagram

PyCharm helps you view the complete hierarchy of the selected type.

So doing, you can modify the contents of the UML Class diagram, to show ancestor or descendant classes, types used in method signatures, and perform all actions that are available for UML Class diagrams.



To view class hierarchy as a UML Class diagram

1. Open the desired class in the editor, and place the caret at the type you want to see hierarchy for.
2. Do one of the following:
 - On the context menu, point to Diagrams, and then choose Show Diagram, or Show Diagram Popup
 - Press `Ctrl+Shift+Alt+U` or `Ctrl+Alt+U`

Tip You can view type hierarchy for any class selected in the Project tool window:



Viewing Members in Diagram

By default, the diagrams show node elements only. However, you can show members too.

To show members in diagram

1. Press `Ctrl+Shift+Alt+U` to open a diagram.
2. Do one of the following:
 - On the diagram toolbar, click the [buttons](#) that correspond to the members you want to show.
 - Right-click the diagram background, and on the context menu, point to the Show Categories command, and then choose the category to be shown or hidden.

Note As you choose a Show Categories command, the state of the corresponding toolbar button changes accordingly.

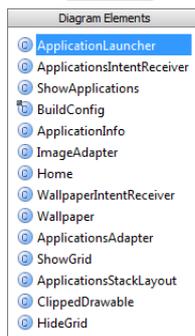
Navigating to Source from Diagram

In a diagram, you can use all the regular procedures that enable navigating to the underlying source code:

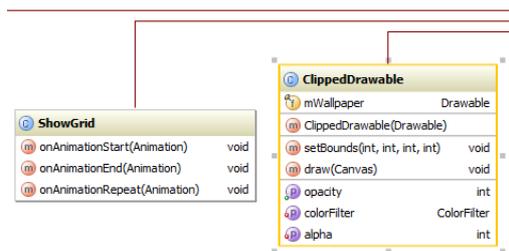
- Jump to source (**F4**). So doing, if the Jump to Source command has been invoked on a class node, the caret is placed at the class declaration.
- View source (**Ctrl+Enter**)
- Navigate by name (**Ctrl+N**, **Ctrl+Shift+N**, or **Ctrl+Shift+Alt+N**)

To navigate through a diagram, follow these general steps

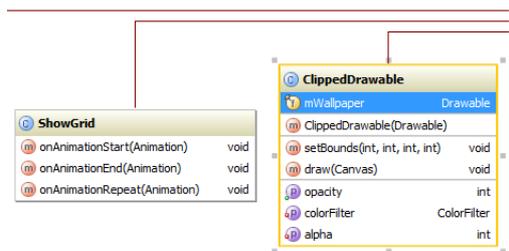
1. Press **Ctrl+F12**. The menu of diagram elements pops up.



2. Use the arrow keys to select the desired diagram element, and press **Enter**. The selected diagram node element becomes active.



3. Press **Enter** once more. The first member of an element gets the focus.



4. The following actions are available:

- Pressing **Enter** toggles the focus between a diagram element and its members.
- When a diagram element has the focus, use the arrow keys to jump between elements.
- When a member has the focus, use the vertical arrow keys to navigate through an element to the desired member.
- Having selected an element or a member, [navigate to the underlying source code](#).

Language and Framework Specific Guidelines

This part provides descriptions of the actions required to fulfil certain tasks using the frameworks integrated into PyCharm:

- [Python](#)
- [Django](#)
- [Buildout](#)
- [CoffeeScript Support](#)
- [Cython Support](#)
- [Databases and SQL](#)
- [Docker](#)
- [Documenting Source Code in PyCharm](#)
- [Flask](#)
- [Google App Engine](#)
- [IntelliLang](#)
- [IPython/Jupyter Notebook Support](#)
- [JavaScript-Specific Guidelines](#)
- [Markup Languages and Style Sheets](#)
- [Matplotlib Support](#)
- [Node.js](#)
- [Pyramid](#)
- [Puppet](#)
- [Templates](#)
- [TextMate](#)
- [Testing Frameworks](#)
- [TypeScript Support](#)
- [Using Bower Package Manager](#)
- [Using File Watchers](#)
- [Using ReactJS in JavaScript and TypeScript](#)
- [Vagrant: Working with Reproducible Development Environments](#)
- [Web2Py](#)

Python

This section provides descriptions of the Python-specific procedures that are used in projects of all supported [types](#), and the procedures that pertain to the empty projects only.

In this section:

- Python
 - [Prerequisite](#)
 - [Python Support](#)
- [Creating Empty Project](#)
- [Configuring Python Interpreter](#)
 - [Configuring Available Python Interpreters](#)
 - [Configuring Python Interpreter for a Project](#)
 - [Changing Name of a Python Interpreter or Virtual Environment](#)
- [Installing, Uninstalling and Upgrading Packages](#)
- [Managing Dependencies](#)
 - [Creating and Running setup.py](#)
 - [Creating Requirement Files](#)
 - [Populating Dependencies Management Files](#)
 - [Resolving Unsatisfied Dependencies](#)
- [Resolving References](#)
- [Using Python Stubs and Typeshed](#)
- [Cleaning .pyc Files](#)
- [Profiler](#)

Prerequisite

At least one [Python](#) interpreter is properly installed on your machine.

Python Support

PyCharm supports Python from version 2.4 up to the version 3.6.

PyCharm provides support for Python 3.5 and (since 2016.3) Python 3.6, with the backing of the following:

- [PEP-0484 -- Type Hints](#)
- [PEP 0448 -- Additional Unpacking Generalizations](#)
- [PEP 0492 -- Coroutines with async and await syntax](#)
- [PEP 526 -- Syntax for variable annotations](#)
- [PEP 498 -- Literal String Interpolation](#)
- [PEP 515 -- Underscores in Numeric Literals](#)
- [PEP 525 -- Asynchronous Generators](#)
- [PEP 530 -- Asynchronous Comprehensions](#)
- and more.

Python support in PyCharm includes:

- Dedicated [project types](#) .
 - Ability to [configure](#) local and remote interpreters and virtual environments. and virtual environments.
 - [Python console](#).
 - Run/debug configurations for [Python](#), and [Python remote debug](#).
 - [Code inspections](#).
 - [Intention actions](#).
 - [Code completion](#) and resolve.
 - Built-in code formatter and separate set of Python [code style settings](#).
 - [Find usages](#) in Python code.
 - [Testing frameworks](#).
 - [Quick documentation](#).
 - Recognizing Python [documentation comments](#).
 - [Documentation generators](#)
 - Configuring [Python debugger](#).
- Note that the debugger contains speedup modules, which use [Cython](#) and are generated with a few changes in the regular files to cythonize the files. The Cython speedups are available for CPython versions 2.7, 3.4 and 3.5.
- On **Windows** the compiled Cython extensions are bundled to PyCharm.
 - On **Linux** and **Mac OS**, Cython extensions should be compiled manually in one in two possible ways:
 - by clicking the link that appears in the warning after the first debugger launch.
 - by running the command from the warning manually in the terminal. A separate Cython extension should be compiled for each version of Python interpreter.

If someone doesn't want to use Cython extensions for some reasons, the environment variable `PYDEVD_USE_CYTHON=NO` should be passed.

- [UML Class diagrams for Python classes.](#)

Creating Empty Project

[Empty projects](#) are intended for pure Python programming.

To create an empty project

1. Do one of the following:
 - On the main menu, choose File | New Project
 - On the [Welcome screen](#), click Create New Project

New Project dialog box opens.

2. In the New Project dialog box, do the following:
 - In the Project type section, click Pure Python .
 - Specify the project location and interpreter .
3. Click the button Create.
4. Configure [project structure](#).

Configuring Python Interpreter

In this section:

- [Configuring Python Interpreter](#)
 - [Basics](#)
- [Configuring Available Python Interpreters](#)
- [Configuring Python Interpreter for a Project](#)
- [Changing Name of a Python Interpreter or Virtual Environment](#)

Basics

In PyCharm, you can define several Python interpreters. So doing, you can choose the one to be used in your project, from the list of the interpreters available on your machine.

PyCharm supports:

- Standard [Python interpreters](#)
- [IronPython](#)
- [PyPy](#)
- [Jython](#)
- [CPython](#)

Python interpreters can be configured on the following levels:

- Current project: selected Python interpreter will be used for the current project.
- New project: selected Python interpreter will be used for the new project instead of the default one.

Configuring Available Python Interpreters

In this section:

- Configuring Available Python Interpreters
 - [Overview](#)
 - [Viewing the list of available interpreters](#)
 - [Configuring the list of available interpreters](#)
 - [Project Interpreters dialog box](#)
 - [Auto-detecting interpreters](#)
 - [Removing interpreters from the list](#)
- [Configuring Local Python Interpreters](#)
- [Configuring Remote Python Interpreters](#)
 - [Configuring Remote Interpreters via Deployment Configuration](#)
 - [Configuring Remote Interpreters via Docker](#)
 - [Configuring Remote Interpreters via Docker Compose](#)
 - [Configuring Remote Interpreters via SSH](#)
 - [Configuring Remote Interpreters via Vagrant](#)
 - [Configuring Remote Interpreters via WSL](#)
- [Creating Virtual Environment](#)
- [Conda Support. Creating Conda Environment](#)
- [Adding Existing Virtual Environment](#)
- [Installing, Uninstalling and Reloading Interpreter Paths](#)

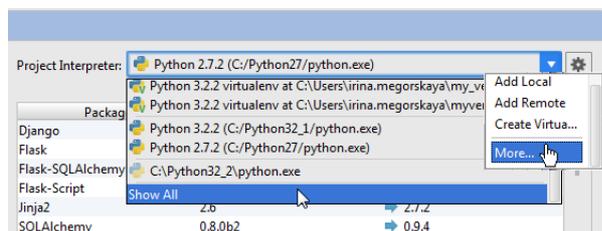
Overview

The list of Python interpreters, available for the various projects, can include interpreters installed locally or remotely, and the various [virtual environments](#).

Viewing the list of available interpreters

To view the list of available interpreters, do one of the following

- In the [Project Interpreter](#) page, click , and then choose More.
- In the [Project Interpreter](#) page, expand the drop-down list of interpreters, and then choose Show all:



Configuring the list of available interpreters

TIP If you want to add [Jython](#) as an interpreter, you have to install it first. Next, [add Jython as the local interpreter](#), specifying the `<installation folder>\bin\jython`.

To configure the list of available interpreters, follow these general steps

1. [Open the Settings/Preferences dialog box](#), and open the [Project Interpreter](#) page.
2. In the list of interpreter types, choose the desired option:
 - [Configure local interpreter](#)
 - [Configure remote interpreter](#)
 - [Configure virtual environment](#)

Choose More to open the [Project Interpreters](#) dialog box, where it is possible to configure the list of the Python interpreters, available on your computer.



Project Interpreters dialog box

The Project Interpreters dialog features the toolbar with the following buttons:

Icon	Tooltip and shortcut	Description
------	----------------------	-------------

	Add	Click this button to choose the type of interpreter to be added: local , remote interpreter,
---	-----	--

or [virtual environment](#).

		
	Remove	Click this button to remove the selected interpreter from the list of available interpreters.
		
	Edit	Click this button to change the name and path of the selected interpreter .
		
	Show virtual environments associated with the other projects	If this button is not pressed, PyCharm shows the virtual environments associated with the current project only.
	Show paths for the selected interpreter	Click this button to show the list of paths for the selected interpreter.

Auto-detecting interpreters

It is not necessary to configure all the interpreters or virtual environments. PyCharm can automatically detect them in certain locations. These automatically detected interpreters are denoted in the list by semi-transparent icons:

- interpreter: 
- virtual environment: 

The ability to automatically detect interpreters depends on the platform. PyCharm looks for the interpreters and virtual environments in the following locations:

- Windows:
 - `C:\PythonXX`
 - `C:\Program Files\PythonXX`
 - `PATH`
 - `WORKON_HOME`
 - `.virtualenv`
- *NIX:
 - `/usr/local/bin/pythonX`
 - `/usr/bin/pythonX`
 - `PATH`
- macOS:
 - `/Library/Frameworks/Python.framework/Version`
 - `/System/Library/Frameworks/Python.framework/Version`
 - `PATH`

Removing interpreters from the list

To remove an interpreter from the list of available interpreters

1. In the list of available interpreters, select the one to be deleted.
2. Click .

To configure a local Python interpreter

1. In the [Project Interpreter](#) page, click .
2. In the drop-down list, choose Add local.



3. In the [Select Python Interpreter](#) dialog box that opens, choose the desired Python executable, located inside the virtual environment folder, and click OK.

Configuring Remote Python Interpreters

This feature is supported in the Professional edition only.

In this section:

- [Prerequisite](#)
- [Configuring remote Python interpreter](#)
- [Important notes](#)

Prerequisite

Before you start working with remote interpreters, make sure that the SSH Remote Run plugin is enabled. The plugin is bundled with PyCharm and is activated by default. If the plugin is not activated, enable it on the [Plugins](#) page of the [Settings / Preferences Dialog](#) as described in [Enabling and Disabling Plugins](#).

Configuring remote Python interpreter

PyCharm provides full integration with the Python interpreters running on remote hosts.

Note that the name of a remote interpreter is automatically prepended with the prefix `Remote`.

To configure a remote Python interpreter, follow these general steps

1. In the [Project Interpreter](#) page of the project settings, click .
2. In the drop-down list, choose Add Remote.



3. Depending on the selection in the dialog box that opens, follow one of the procedures:
 - [Configuring Remote Interpreters via SSH](#)
 - [Configuring Remote Interpreters via Vagrant](#)
 - [Configuring Remote Interpreters via Deployment Configuration](#)
 - [Configuring Remote Interpreters via Docker](#)
 - [Configuring Remote Interpreters via Docker Compose](#)
 - [Configuring Remote Interpreters via WSL](#)

Important notes

When a remote Python interpreter is added, at first the PyCharm helpers are copied to the remote host. PyCharm helpers are needed to run remotely the packaging tasks, debugger, tests and other PyCharm features. Next, the skeletons for binary libraries are generated and copied locally. Also all the Python library sources are collected from the Python paths on a remote host and copied locally along with the generated skeletons. Storing skeletons and all Python library sources locally is required for resolve and completion to work correctly in PyCharm.

PyCharm checks remote helpers version on every remote run, so if you update your PyCharm version, the new helpers will be uploaded automatically and you don't need to recreate remote interpreter.

SFTP support is required for copying helpers to the server.

This feature is supported in the Professional edition only.

In this section:

- [Prerequisite](#)
- [Defining remote Python interpreter via deployment configuration](#)

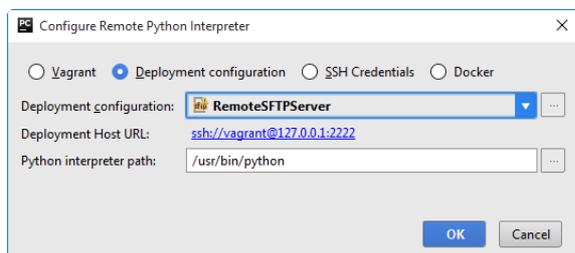
Prerequisite

Make sure you have at least one PyCharm-wide **server access configuration** of the type SFTP to establish access to the target host. To make a configuration available in all PyCharm projects, clear the Visible only for this project check box in the **Connection Tab**. See [Creating a Remote Server Configuration](#) for details.

Defining remote Python interpreter via deployment configuration

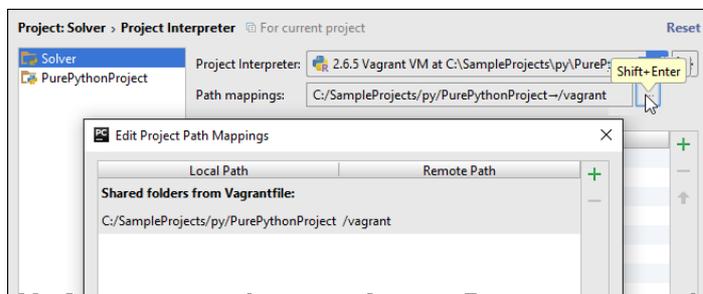
To define a remote Python interpreter via deployment configuration

1. In the **Project Interpreter** page of the **project settings**, click .
2. In the drop-down list, choose **Add Remote**.
3. In the dialog box **Configure Remote Python Interpreter**, click the radio-button **Deployment configuration**:



4. In the **Configure Remote Python Interpreter** dialog box, select the desired SFTP server from the **Deployment configuration** drop-down list. Note that the deployment host URL is displayed for the selected SFTP server automatically. The **Python interpreter path** field displays the path to the desired Python executable. You can accept default, or specify a different one.
5. Click **OK** in the **Configure Remote Python Interpreter** dialog box. The configured remote interpreter is added to the list.
6. Back in the **Project Interpreter** page, if necessary, configure the path mappings:

1. Click :



2. In the **dialog box that opens** add (+) or delete (-) path mappings as desired.

This feature is supported in the Professional edition only.

In this section:

- [Prerequisites](#)
- [Limitation](#)
- [Configuring remote interpreter using Docker](#)

Prerequisites

- Before you start working with Docker, make sure that the Docker integration plugin is enabled. The plugin is bundled with PyCharm and is activated by default. If the plugin is not activated, enable it on the [Plugins](#) page of the [Settings / Preferences Dialog](#) as described in [Enabling and Disabling Plugins](#).
- Before you start working with Docker, make sure that the Ruby Docker plugin is enabled. The plugin is bundled with PyCharm and is activated by default. If the plugin is not activated, enable it on the [Plugins](#) page of the [Settings / Preferences Dialog](#) as described in [Enabling and Disabling Plugins](#).

Limitation

As of this writing, on Windows and MacOS platforms running, debugging and profiling applications is only possible for the projects residing in the user home directory. If necessary, configure other directories in the VirtualBox settings.

Docker Machine on Linux does not share user home folder, and any other folders as well. Thus, for running, debugging and profiling applications, the user should add shared folders with the project sources to the VirtualBox Docker virtual machine manually. These shared folders on VM should be mapped one to one to the folders on the host Linux machine, i.e. (host) `/home/user <=> /home/user` (VM) or (host) `/home/my/project <=> /home/my/project` (VM)

Configuring remote interpreter using Docker

To set up a Python interpreter inside a Docker container for your project, follow these steps:

1. In the [Project Interpreter](#) page of the [Settings/Preferences dialog box](#), click , and then choose Add Remote.
2. From the Configure Remote Python Interpreter dialog box that opens, choose the option Docker:



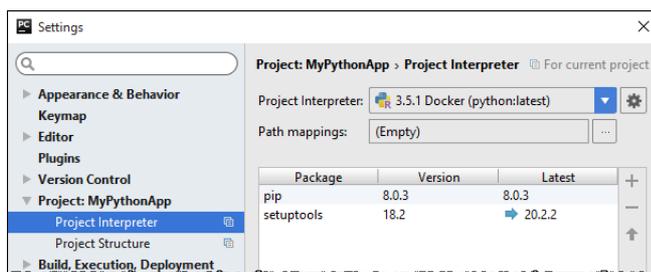
3. If the required Docker machine is not available, click New, and in the dialog box that opens, specify the machine name, API URL, and the certificates folder.

If you want to import credentials of an existing Docker machine, select the corresponding check box.

The configured remote interpreter is set as the [project interpreter](#). Now you can run, debug and profile your application using Python inside a Docker container.

Note that for the project interpreters created via Docker, the field Path mappings is always empty, and [packages toolbar](#) is not available.

The buttons on this toolbar are disabled for the [Docker interpreters](#):



All the packages should be already installed in the Docker image. If some packages are missing, then you will have to create a new Docker image, as described on the page [Quickstart Guide: Compose and Django](#).

Configuring Remote Interpreters via Docker Compose
This feature is supported in the Professional edition only.

In this section:

- [Prerequisite](#)
- [Limitation](#)
- [Using Docker Compose to configure a remote interpreter](#)

Prerequisite

Before you start working with Docker Compose, make sure that the Docker Compose integration plugin is enabled. The plugin is bundled with PyCharm and is activated by default. If the plugin is not activated, enable it on the [Plugins](#) page of the [Settings / Preferences Dialog](#) as described in [Enabling and Disabling Plugins](#).

Limitation

As of this writing, Docker Compose integration in PyCharm is not supported on Windows.

Using Docker Compose to configure a remote interpreter

To configure a remote Python interpreter, follow these steps:

1. In the [Project Interpreters](#) page of the Settings/Preferences dialog box, click .
2. From the drop-down list, choose Add Remote .
The dialog box Configure Remote Python Interpreter opens.
3. In this dialog box that opens, do the following:
 1. Click the radio button Docker Compose.
 2. Choose the desired server from the drop-down list, or click New.
 3. In the Configuration file(s) field, specify the required configuration files, by clicking .
Use the toolbar , , ,  to make up the list of configuration files.

Note that if the path points to a specific file (for example, `docker-compose.yml`), then this configuration will be used. If the path points to a directory, then the configuration is defined by the file `docker-compose.yml` and (if exists) `docker-compose.override.yml`.
 4. In the field Environment variables, specify the environment variables in the `docker-compose.yml` file. Click the browse button  next to this field, and specify the required environment variables using the toolbar buttons , , , .
 5. Specify the service that represent your project.
 6. The Python interpreter path field displays the path to the desired Python executable. You can accept default, or specify a different one.
4. Click OK in the Configure Remote Python Interpreter dialog box. The configured remote interpreter is added to the list.

From this point, [autocompletion](#), [code inspections](#), as well as other features, will be driven by the interpreter from the Docker container derived from the service description in `docker-compose.yml` file.

Configuring Remote Interpreters via SSH

This feature is supported in the Professional edition only.

In this section:

- [Prerequisites](#)
- [Configuring remote Python interpreter via SSH credentials](#)

Prerequisites

- A `ssh` server should run on a remote host, since PyCharm runs remote interpreter via `ssh-session`.
- If you want to copy your sources to a remote computer, create a deployment configuration, as described in the section [Creating a Remote Server Configuration](#).

Configuring remote Python interpreter via SSH credentials

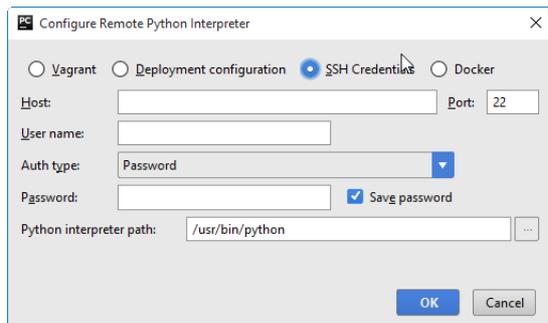
To configure a remote Python interpreter via SSH credentials

1. In the [Project Interpreter](#) page of the [Settings/Preferences dialog box](#), click .
2. From the drop-down list, choose Add Remote.

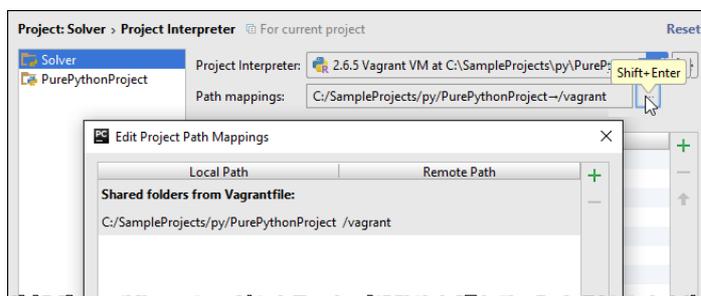


The dialog box Configure Remote Python Interpreter opens.

3. In the dialog box Configure Remote Python Interpreter, click the radio-button SSH credentials:



4. In the Configure Remote Python Interpreter dialog box, fill in the server information (host, port, etc.)
5. The Python interpreter path field displays the path to the desired Python executable. You can accept default, or specify a different one.
6. Click OK in the Configure Remote Python Interpreter dialog box. The configured remote interpreter is added to the list.
7. Back in the [Project Interpreter](#) page, if necessary, configure the path mappings:
 1. Click 



2. In the [dialog box that opens](#) add (+) or delete (-) path mappings as desired.

This feature is supported in the Professional edition only.

In this section:

- [Prerequisites](#)
- [Configuring remote Python interpreter via Vagrant](#)

Prerequisites

Before you start working with Vagrant, make sure that the Vagrant plugin is enabled. The plugin is bundled with PyCharm and is activated by default. If the plugin is not activated, enable it on the [Plugins](#) page of the [Settings / Preferences Dialog](#) as described in [Enabling and Disabling Plugins](#).

Besides that, make sure that the following prerequisites are met (outside of PyCharm):

- [Oracle's VirtualBox](#) is installed on your computer.
- [Vagrant](#) is installed on your computer, and all the necessary infrastructure is created.
- The parent folders of the following executable files are added to the system **PATH** variable:
 - `vagrant.bat` or `vagrant` from your Vagrant installation. This should be done automatically by the installer.
 - `VBoxManage.exe` or `VBoxManage` from your Oracle's VirtualBox installation.
- The required virtual boxes are created.

Configuring remote Python interpreter via Vagrant

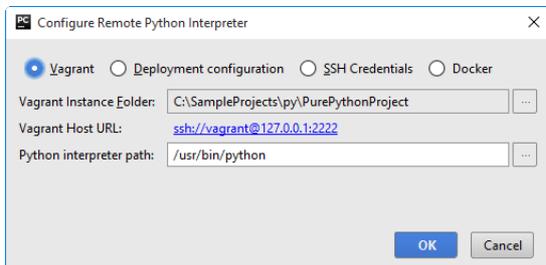
To configure a remote Python interpreter

1. In the [Project Interpreters](#) page of the Settings/Preferences dialog box, click .
2. From the drop-down list, choose Add Remote .



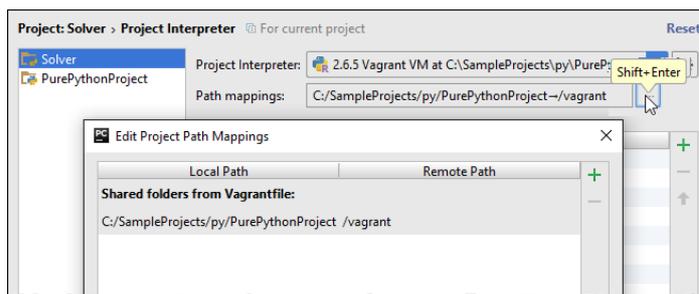
The dialog box Configure Remote Python Interpreter opens.

3. In this dialog box, click the radio button Vagrant:



In this case, the remote server settings are taken from the vagrant configuration file, defined in the page [Vagrant](#).

4. Click the browse button  next to the field Vagrant instance folder, and specify the desired Vagrant instance folder. This results in showing the link to Vagrant host URL.
5. The Python interpreter path field displays the path to the desired Python executable. You can accept default, or specify a different one.
6. Click OK in the Configure Remote Python Interpreter dialog box. The configured remote interpreter is added to the list.
7. Back in the [Project Interpreter](#) page, if necessary, configure the path mappings:
 1. Click .



2. In the [dialog box that opens](#) add (+) or delete (-) path mappings as desired.

Configuring Remote Interpreters via WSL

On this page:

- [Overview](#)
- [Prerequisites](#)
- [Running SSH on WSL](#)
- [Configuring WSL as remote interpreter](#)

Overview

PyCharm can be configured to use WSL as remote interpreter, but without deployment, since each drive on Windows is mapped to the appropriate folder in `/mnt/<DRIVE_NAME>` in WSL. So, only mappings need to be configured.

Use `127.0.0.1` as the host name, and login and password entered after first `lxrun /install`. Also, set `C:\` to `/mnt/c/` in the mappings.

Prerequisites

Make sure that the following prerequisites are met:

- Win10 build `14361` or later is installed. It is also possible to upgrade the current Insider Preview.
- WSL is installed (like `lxrun /install` or `lxrun /update`)

Running SSH on WSL

1. Run `bash.exe`.
 2. Update to latest version `sudo apt-get update` or `sudo apt-get upgrade`.
 3. Open `/etc/ssh/sshd_config` file for editing:
 - Enable password authentication (unless you want to use the public keys): set `PasswordAuthentication` to `yes`.
 - Since `chroot` is not implemented in WSL, you also need to set `UsePrivilegeSeparation` to `no`.
- Save changes and close `sshd_config`.
4. Type `sudo $(sudo which sshd) -d` to run OpenSSH in the foreground. You should see something like "Server listening on <server> port <number>"
 5. From another `bash.exe` session, try `ssh 127.0.0.1`.
 6. If you see a message about ECDSA encryption algorithm fingerprint, answer `y`. Password prompt appears, which means that the server works correctly.
 7. Turn it off with `Ctrl+C`, and type the command `sudo service ssh start` to start the server in daemon mode. Looks like upstart is broken in the current WSL, so you would need to run `bash.exe`, `start sshd` and keep console window opened, since WSL stops when the last client disconnects. You may create `wsl_ssh.bat` file like `bash.exe -c "sudo service ssh start &&& sleep 999d"` and use it to launch ssh.

Now that SSH is running, you can configure the remote interpreter.

Tip Modern Windows installations may have built-in SSH server that has to be stopped, because it prevents `OpenSSH` from listening to port `22`. For example, from the elevated powershell one may run:

```
Get-Service -Name ssh* | Stop-Service -Force
Get-Service -Name ssh* | Set-Service -StartupType Disabled
```

- If you get a message "Could not load host key: /etc/ssh/ssh_host_rsa_key", you may need to regenerate keys. In bash prompt type

```
sudo ssh-keygen -A
```

Configuring WSL as remote interpreter

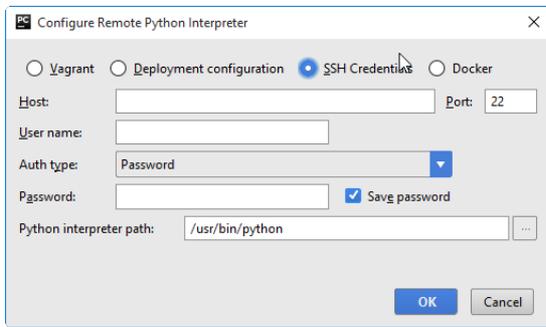
To configure WSL as the remote interpreter, follow these steps

1. In the [Project Interpreter](#) page of the [Settings/Preferences dialog box](#), click .
2. From the drop-down list, choose Add Remote.



The dialog box Configure Remote Python Interpreter opens.

3. In the dialog box Configure Remote Python Interpreter, click the radio-button SSH credentials:

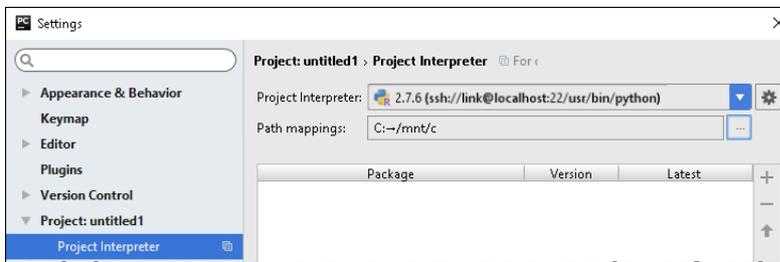


4. In the Configure Remote Python Interpreter dialog box, fill in the server information (host= `localhost` or `127.0.0.1` , port = `22` , user name and password used on first launch of `lxrun /install .`)
5. The Python interpreter path field displays the path to the desired Python executable. You can accept default, or specify a different one.
6. Click OK in the Configure Remote Python Interpreter dialog box. The configured remote interpreter is added to the list.

Later, you can use this interpreter as the [project interpreter](#). The only difference lays with [path mappings](#).

To configure path mappings, follow these steps

1. Next to the Path mappings field, click . [Edit Project Path Mappings Dialog](#) opens.
2. In this dialog box, map local paths to remote ones (for example, map `C:\` to `/mnt/c/`):



Several mappings are delimited with semicolons.

3. Click OK when ready.

Creating Virtual Environment

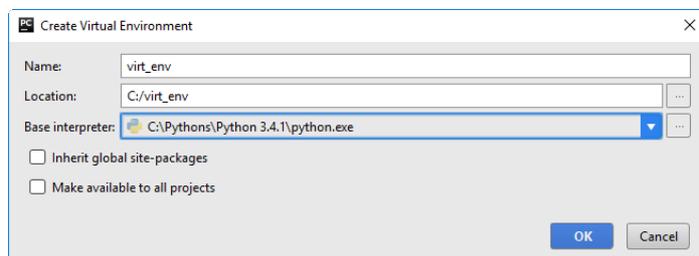
PyCharm makes it possible to create a virtual environment using the [virtualenv](#) tool. So doing, PyCharm tightly integrates with `virtualenv`, and enables configuring virtual environments right in the IDE.

`virtualenv` tool comes bundled with PyCharm, so the user doesn't need to install it.

To create a virtual environment

1. Open the [Settings/Preferences dialog box](#), and then open the [Project Interpreters](#) page.
2. Click  next to the Project Interpreter field, and choose the option `Create VirtualEnv`.
Create Virtual Environment dialog box opens.
3. In the Create Virtual Environment dialog box, do the following:
 - In the Name field, type the name of the new virtual environment, or accept the suggested default name.
 - In the Location field, specify the target directory, where the new virtual environment will be created, or accept the suggested default location.
 - From Base interpreter drop-down list, select one of the [configured Python interpreters](#), which will be used as the base for the new virtual environment.
If the desired base interpreter is missing in the drop-down list, you can locate it manually by clicking .
 - If you want the `site-packages` of the base interpreter to be visible from the virtual environment, select the check box `Inherit global site-packages`. If you leave this check box cleared, the new virtual environment will be completely isolated.
 - You can make this virtual environment available to all projects, by selecting the check box `Make available to all projects`.

Click OK to apply changes and close the dialog box.



Tip You can create as many virtual environments as required. To easily tell them from each other, use different names.

In this section:

- [Prerequisite](#)
- [Creating Conda environment](#)

Prerequisite

Make sure that [Anaconda](#) or [Miniconda](#) is downloaded and installed on your computer.

Whether you install Anaconda or Miniconda, [depends](#) on you needs.

Creating Conda environment

To create a Conda environment

1. [Open the Settings/Preferences dialog box](#), and then open the [Project Interpreters](#) page.
2. Click  next to the Project Interpreter field, and choose the option Create Conda Env. Create Conda Environment dialog box opens.
3. In the Create Conda Environment dialog box, do the following:
 - In the Name field, type the name of the new Conda environment, or accept the suggested default name.
 - In the Location field, specify the target directory, where the new Conda environment will be created, or accept the suggested default location.
 - From Python version drop-down list, select one of the configured Python interpreters, which will be used as the base for the new Conda environment.
 - Make this virtual environment available to all projects by selecting the check box Make available to all projects.

Click OK to apply changes and close the dialog box.

Adding Existing Virtual Environment

Once a virtual environment is created, it can be added to the list of available interpreters, as a local interpreter.

To add an existing virtual environment to the list of available interpreters

1. In the [Project Interpreter](#) page, click .
2. In the drop-down list, choose Add local.



3. In the [Select Python Interpreter](#) dialog box that opens, choose the desired Python executable, located inside the virtual environment folder, and click OK.

Installing, Uninstalling and Reloading Interpreter Paths

PyCharm helps view and manage the interpreter paths:

- [Viewing interpreter paths](#)
- [Adding interpreter paths](#)
- [Removing interpreter paths](#)
- [Reloading interpreter paths](#)

To view the interpreter paths

1. In the [Project Interpreter](#) page of the project settings, click  button.
2. In the drop-down list, click More. The available interpreters show up in the [Project Interpreters](#) dialog.
3. Select the desired interpreter.
4. In the toolbar of the [Project Interpreters](#) dialog box, click the button . The existing paths of the selected interpreter show up in the [Interpreter Paths](#) dialog box.

To add an interpreter path

1. In the toolbar of the [Interpreter Paths](#) dialog box, click .
2. Choose the desired path in the [Select Path](#) dialog.

To delete interpreter paths

1. Select the paths to be deleted.
2. In the toolbar of the [Interpreter Paths](#) dialog box, click .
The removed paths remain in the list with the note "removed by user".

To reload interpreter paths

If an interpreter has been updated, it is a good idea to refresh its paths.

- In the toolbar of the [Interpreter Paths](#) dialog box, click .

Configuring Python Interpreter for a Project

In this section:

- [Introduction](#)
- [Selecting Python interpreter for a project](#)
- [Working on the same project on different platforms](#)

Introduction

PyCharm helps assign a Python interpreter for a project.

So doing, each one of the projects, opened in the same window, can have an interpreter of its own, selected from the list of [available interpreters](#).

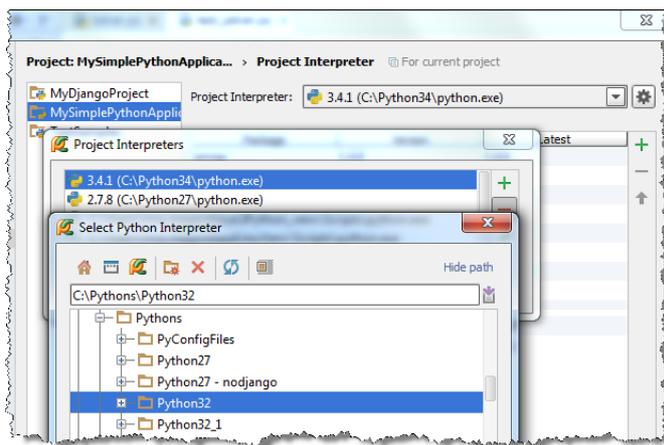
Note that PyCharm stores only the interpreter name in the project settings.

Selecting Python interpreter for a project

To configure Python SDK for the current project, follow these steps:

1. Open the Settings/Preferences dialog box, and click [Project Interpreter](#) page.
2. In the Projects pane, choose the desired project.
3. Choose SDK from the list of available Python interpreters and virtual environments.
This list includes:
 - Python interpreters, which reside in standard locations.
 - Virtual environments, which reside under the project folder, or under the folder specified as an environment variable `WORKON_HOME`.
 - Other Python interpreters, installed locally or remotely.
4. If the desired interpreter is not in the list, click , and configure the desired interpreter as described in the section [Configuring Available Python Interpreters](#).

For example, look at the following image:



5. Apply changes.

Tip An interpreter can be made the project default when it is added.

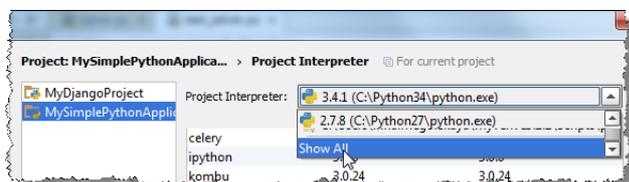
Working on the same project on different platforms

You can work on the same PyCharm project on different platforms (for example, on Windows at work, and on MacOS at home). This can be easily done, if you rename the project interpreter.

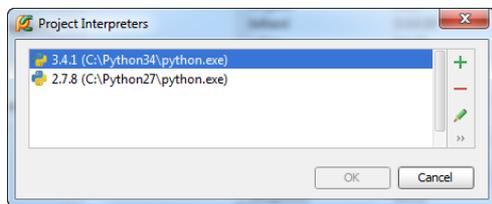
The reason is that PyCharm stores the interpreter **name** with the project, but not the interpreter path.

To rename an interpreter, follow these steps

1. In the Settings/Preferences dialog, click the page [Project Interpreter](#), and select the desired project.
2. Click the drop-down list Project Interpreter, and choose Show All.

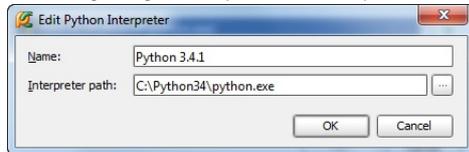


The dialog box Project Interpreters opens.



3. Choose the desired interpreter and click . The dialog box Edit Python Interpreter appears.

4. In this dialog, change the interpreter name as required.

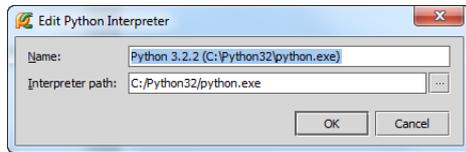


Note that only this name is stored with the project settings, making it possible to use different interpreters on each platform, without changing the project settings every time you switch to another platform.

With PyCharm, one can easily discern numerous Python interpreters and virtual environments by their names, rather than by the long paths to the executables.

To change visible name of a Python interpreter

1. In the [Project Interpreters](#) page, select one of the configured interpreters or virtual environments.
2. Click .
3. In the Edit Python Interpreter dialog box that opens, type the desired interpreter name.



The Python interpreter name specified in the Name field, becomes visible in the list of available interpreters.

If necessary, change the path to the Python executable.

Installing, Uninstalling and Upgrading Packages

In this section:

- [Basics](#)
- [Installing packages](#)
- [Uninstalling packages](#)
- [Upgrading packages](#)

Basics

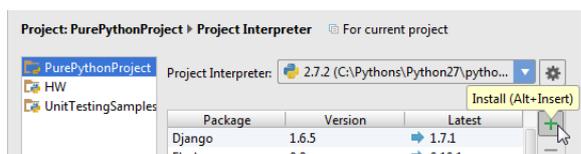
PyCharm provides a dedicated tool for installing, uninstalling, and upgrading Python packages. So doing, if a packaging tool is missing, PyCharm suggests to install it.

PyCharm smartly tracks the status of packages and recognizes outdated versions by showing the number of the currently installed package version (column Version), and the latest available version (column Latest). When a newer version of a package is detected, PyCharm marks it with the arrow sign ➡.

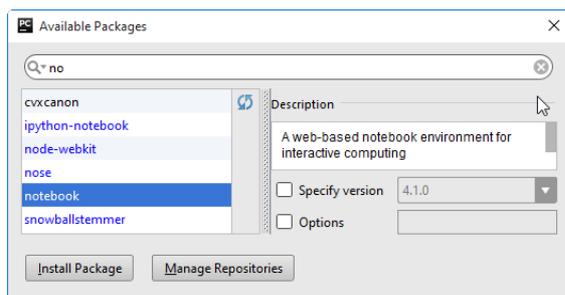
Installing packages

To install a package

1. In the [Project Interpreter](#) page of the project settings, select the desired Python interpreter or virtual environment.



2. Click **+**.
3. In the [Available Packages](#) dialog box that opens, select the desired package from the list. If necessary, use the Search field, where you can enter any string. So doing, the list of packages shrinks to show the matching packages only.



4. If required, select the following check boxes:
 - Specify version: if this check box is selected, you can select the desired version from the drop-down list of available versions. By default, the latest version is taken.
 - Options: If this check box is selected, you can type the options in the text field.
5. Click Install Package.

You can use the various packaging tools, including [devpi](#) or [PyPi](#).

To specify a custom repository, follow these steps

1. In the [Project Interpreter](#) page of the project settings, click **+**, and then, in the [Available Packages](#) dialog box, click Manage Repositories.
2. In the Manage Repositories dialog box that opens, click **+** to add a URL of a local repository, for example, something like `http://somehost/alice/dev`.
3. In the Manage Repositories dialog box, click OK.
4. Back in the [Available Packages](#) dialog box, click **↻** to reload the list of packages. As a result, the packages that exist on the local server appear.

Tip PyCharm provides a quick fix that automatically installs the package you're trying to import: if, after the keyword `import`, you type a name of a package that is not currently available on your machine, a quick fix suggests you to either ignore the unresolved reference, or download and install the missing package:



Uninstalling packages

To uninstall a package

1. On the [Project Interpreter](#) page, in the list of packages, select the ones to be deleted.
2. Click . The selected packages are removed from disk.

Upgrading packages

To upgrade a package

1. On the [Project Interpreter](#) page, in the list of packages, select the packages to be upgraded.
2. Click . The selected package is upgraded to the latest available version.

Managing Dependencies

Possibility to extract information from requirements.txt or setup.py; Quick fix to add packages to requirements.txt or setup.py; Quick fix to install missing packages; Notification about the unsatisfied requirements.

PyCharm makes it possible to track the unsatisfied dependencies in your projects, and provides integration with the major means of dependencies management.

In this section:

- [Creating and Running setup.py](#)
- [Creating Requirement Files](#)
- [Populating Dependencies Management Files](#)
- [Resolving Unsatisfied Dependencies](#)

Creating and Running setup.py

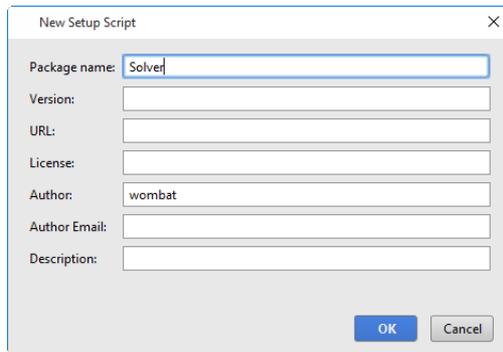
PyCharm provides an action that helps create `setup.py` script. Once `setup.py` is created, the corresponding action becomes disabled, but instead appears the action that allows running tasks of this utility.

In this section:

- [Creating setup.py](#)
- [Running setup.py](#)

To create setup.py for a project

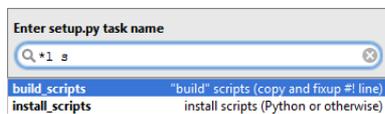
1. In the Project tool window, choose the project you want to package.
2. On the main menu, choose Tools | Create setup.py.
3. In the New Setup Script dialog box, specify package name, version and the other required information:



4. Click OK when ready. PyCharm creates `setup.py` and opens it in the editor.

To run a task of the setup.py utility

1. On the main menu, choose Tools | Run setup.py.
2. In the Enter setup.py task name dialog box, type the letters of the task names. Note that asterisk wildcard and initial letters of the snake_case names are honored.
As you type, the suggestion list shrinks to show the matching names only. Choose the desired task, and press `Enter`.



- [Creating requirements](#)
- [Configuring the default requirements file](#)

Creating requirements

To define requirements, follow these general steps

1. [Create new file](#) in the root directory of your project.
2. In the New File dialog box, specify the file name.
3. Type the names of the required packages as plain text. Note that recursive requirements syntax is supported: you can use the main requirements file, and include the other requirements with `-r` syntax.

Configuring the default requirements file

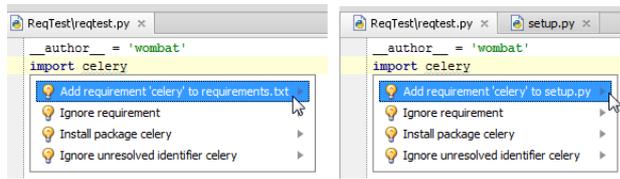
To configure the default requirements file

1. [Open the Settings/Preferences dialog box](#), and then click the page [Python Integrated Tools](#).
2. In the Package requirements file field, type the name of the requirements file. By default, `requirements.txt` is suggested. You can either accept default, or click the browse button and locate the desired file.

Though you can edit the dependencies management files according to their syntax, PyCharm provides quick fixes that enable populating these files.

To populate dependency management files

1. Create `setup.py` or `requirements.txt`, as described in the sections [Creating and Running setup.py](#) and [Creating Requirement Files](#).
2. In an `import` statement of a Python file, click a package which is not yet imported. PyCharm suggests a quick fix:

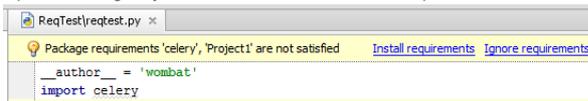


3. Select and apply the suggested quick fix. The package in question is added to the dependency management file.

PyCharm provides quick fixes and notification related to the unsatisfied dependencies.

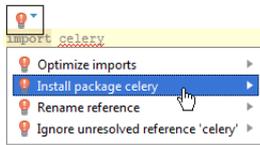
To resolve unsatisfied dependencies, do one of the following

- Open for editing a Python file that contains unsatisfied dependencies. A notification bar is displayed on top:



Click one of the provided links to satisfy or ignore requirements.

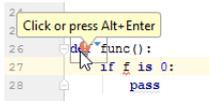
- If no dependencies management files are present, PyCharm suggests a quick fix to just install a missing package:



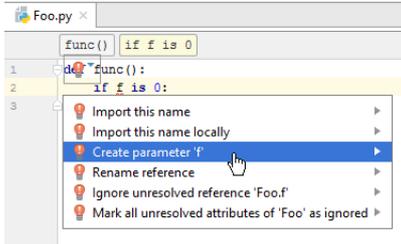
Resolving References

PyCharm's on-the-fly inspection immediately detects unresolved references, and highlights them with the red curly line. PyCharm suggests quick fixes to deal with the unresolved references in the source code.

When you place the caret at an unresolved reference, PyCharm shows the **red light bulb**.



Click the bulb, or press **Alt+Enter** to reveal the list of available quick fixes:



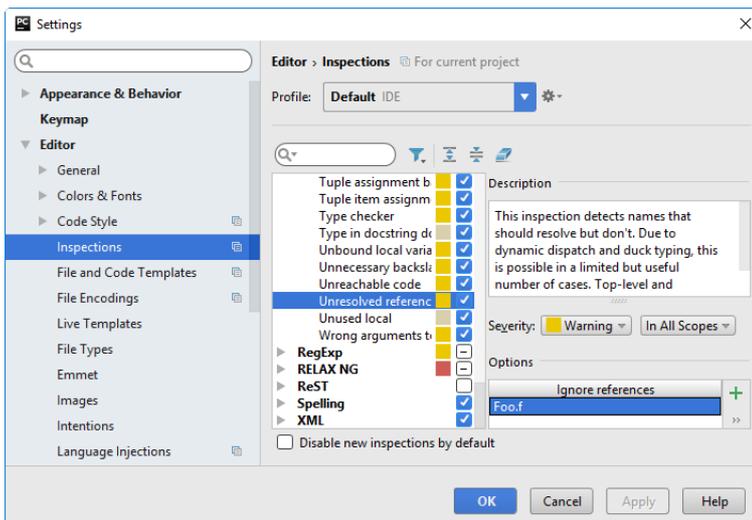
PyCharm suggests a number of solutions. For example, choose one of the following options:

- Import this name to add an import statement.
- Create parameter to add a parameter with some initial value to the list of function parameters.

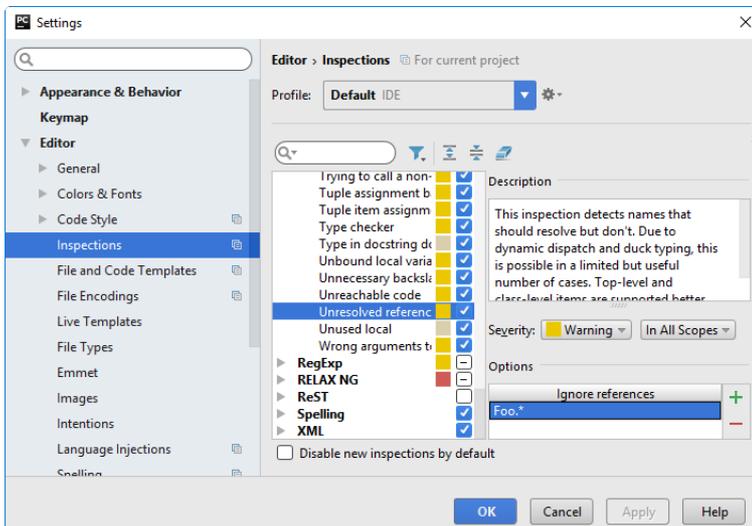
```
def func(f=None):
```

See also [Extract Parameter](#) and [Change Signature](#).

- Ignore unresolved reference <fully qualified symbol name>. So doing, the fully qualified name of the symbol in question will be added to the list of ignored references (File | Settings/Preferences | Editor | Inspections - Python - Unresolved references).



- Mark all attributes of <fully qualified type name> as ignored. In this case, the list of ignored references (File | Settings/Preferences | Editor | Inspections - Python - Unresolved references) will be modified with the fully qualified name of the type in question with the wildcard (*) at the end.



Using Python Stubs and Typeshed

In this section:

– [Overview](#)

Overview

[Typeshed](#) is a set of files with type annotations for the standard Python library and various packages.

Using Typeshed allows making the code completion and error highlighting more "intelligent".

The Python skeletons repository <https://github.com/JetBrains/python-skeletons> is now deprecated.

Cleaning .pyc Files

In this section:

- [Basics](#)
- [Removing Python compiled files](#)

Basics

Depending on the selected Python interpreter, the following Python compiled files are created:

- `.pyc` (for Python interpreter)
- `$py.class` (for Jython interpreter)

By default, the `.pyc` and `$py.class` files are ignored, and thus are not visible in the Project tool window. However, PyCharm makes it possible to delete `.pyc` files from projects or directories.

Please note the following:

- If you [rename](#) or [delete](#) a Python file, the corresponding compiled file is also deleted.
- Python compiled files are deleted recursively.
- If you perform [update from VCS](#) and skip automatic cleanup, then removing Python compiled files is vital. The reason is that after update from version control, some of the Python files could have been deleted, and executing the Clean Python compiled files command will help you get rid of the unnecessary Python compiled files.

Removing Python compiled files

To remove Python compiled files

1. In the [Project tool window](#), right-click a project or directory, where Python compiled files should be deleted from.
2. On the context menu, choose Clean Python compiled files.
The `.pyc` and `$py.class` files residing in the selected directory are silently deleted.

Profiler

This feature is supported in the Professional edition only.

In this section:

- [Overview. yappi and CProfile vs VMprof](#)
- [Starting the profiling session](#)
- [Working with the profiling results](#)
 - [Jumping to the source code](#)
 - [Viewing Call Graph](#)
 - [Reviewing the existing snapshots](#)

Overview. yappi and CProfile vs VMprof

PyCharm allows running the current [run/debug configuration](#) while attaching a Python profiler to it.

If you have a [yappi](#) profiler installed on your interpreter, PyCharm starts the profiling session with it by default, otherwise it uses the standard [cProfile](#) profiler.

Besides these two tracing profilers, PyCharm supports also sampling (statistical) profiler [vmprof](#), which should be installed on the selected Python interpreter.

Warning! If you are Windows 64 bit user, you have to install Python 32 bit, to make [vmprof](#) work.

Install 32-bit Python as described on the page [Python Releases for Windows](#).

Note the following:

- A profiler runs on both **local** and **remote** interpreters.
- A profiler runs in the following order: `vmprof`, `yappi`, `cProfile`.

Starting the profiling session

To start the profiling session, do one of the following:

- Click  on the main toolbar.
- Choose Run | Profile <current run.debug configuration name> on the main menu.

The profiler starts in the dedicated tab of the [Run tool window](#).

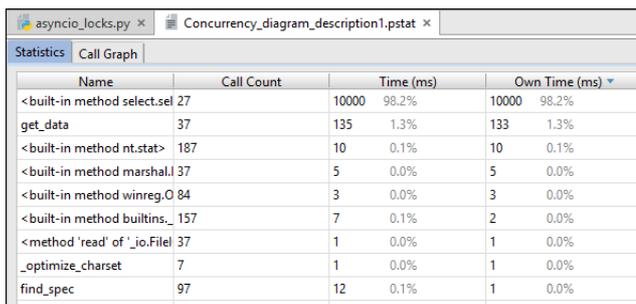
Working with the profiling results

On the toolbar of the profiler tab, click . This action results in the following:

- The snapshot is saved to the default location under `.PyCharmXX/system/snapshots` directory under the user's home. For CProfiler, it is saved as `<project name>.pstat` file:

```
Run > asyncio_locks
C:\Python3\Python3.5\python.exe "C:\Program Files (x86)\JetBrains\PyCharm 163.7258\helpers\profiler\run_profi
Starting cProfile profiler
Task A: Compute factorial(2)...
Task A: factorial(2) = 2
Task B: Compute factorial(2)...
Task B: Compute factorial(3)...
Task B: factorial(3) = 6
Task C: Compute factorial(2)...
Task C: Compute factorial(3)...
Task C: Compute factorial(4)...
Task C: factorial(4) = 24
Task D: Compute factorial(2)...
Task D: Compute factorial(3)...
Task D: Compute factorial(4)...
Task D: Compute factorial(5)...
Task D: factorial(5) = 120
Snapshot saved to C:\Users\wombat\PyCharm2016.3\system\snapshots\Concurrency_diagram_description1.pstat
Process finished with exit code 0
```

- The profiling results open in the `<project name>.pstat` tab in the editor, which consists of two tabs: Statistics and Call Graph:



Name	Call Count	Time (ms)	Own Time (ms)
<built-in method select.select>	27	10000 98.2%	10000 98.2%
get_data	37	135 1.3%	133 1.3%
<built-in method nt.stat>	187	10 0.1%	10 0.1%
<built-in method marshal>	37	5 0.0%	5 0.0%
<built-in method winreg.O	84	3 0.0%	3 0.0%
<built-in method builtins._	157	7 0.1%	2 0.0%
<method 'read' of '_io.File	37	1 0.0%	1 0.0%
_optimize_charset	7	1 0.0%	1 0.0%
find_spec	97	12 0.1%	1 0.0%

For `vmprof`, it is saved as `<project name>.prof` file.

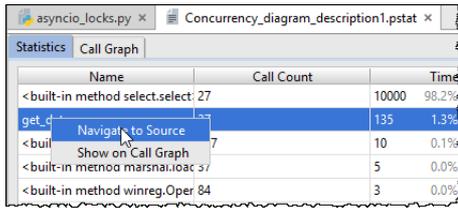
Tip! For the `vmprof` profiler, the following differences to the other profilers are important:

- The snapshot includes the additional tab Call Tree.
- The option Profile lines is available in `vmprof` and enabled by default. Once statistics is gathered, apart from just seeing the standard profiler report, call graph and call tree, you can also see the

line profiling results right in the editor in the left gutter. Lines consuming more processor time are marked red.

Jumping to the source code

To navigate to the source code of a certain function, right-click the corresponding entry on the Statistics tab, and choose Navigate to Source on the context menu:

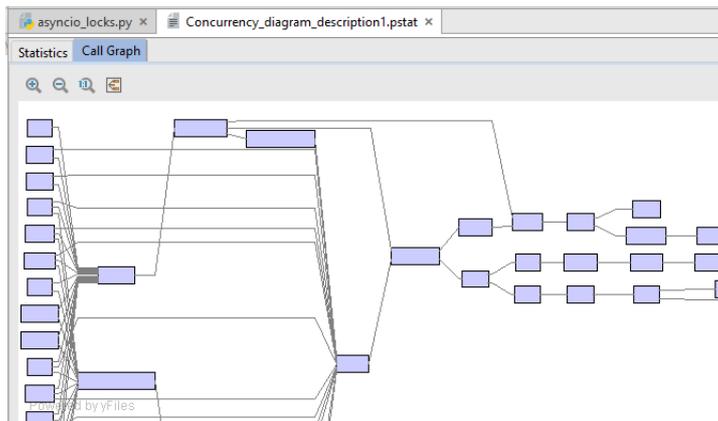


The source code of the function in question opens in the editor.

Viewing Call Graph

To navigate to the call graph of a certain function, right-click the corresponding entry on the Statistics tab, and choose Show on Call Graph on the context menu.

The Call Graph tab opens with the function in question highlighted:

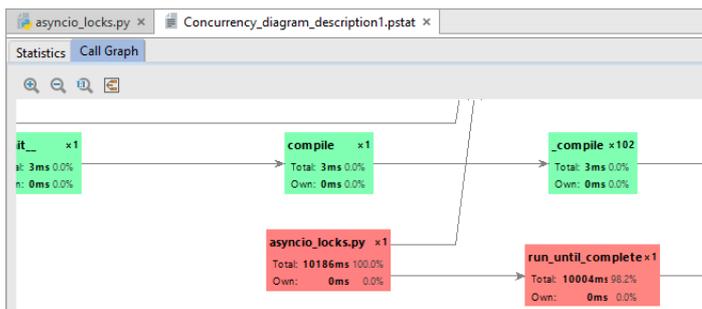


To increase the graph scale, click ; to show actual size of the graph, click .

Use the toolbar button to fit contents into the current diagram size.

To decrease the graph scale, use .

Note the color codes on the Call Graph. The functions marked red consume more time; the fastest functions are green.



Reviewing the existing snapshots

To open an existing snapshot, follow these steps:

1. On the main menu, choose Tools | Open CProfile snapshot.
2. In the [Select PStat file](#) dialog, choose the desired file with the extension `.pstat`.
The profiling results open in the `<project name>.pstat` tab in the editor.

Django

This feature is supported in the Professional edition only.

In this section:

- Django
 - [Prerequisite](#)
 - [Supported versions of Django and Python](#)
 - [Django support](#)
 - [Enabling or disabling Django support](#)
- [Creating Django Project](#)
- [Creating Django Application in a Project](#)
- [Creating Templates](#)
- [Debugging Django Templates](#)
- [Navigating Between Templates and Views](#)
- [Navigating to Implementing Blocks of Templates](#)
- [Running Tasks of manage.py Utility](#)
- [Referring to Static Contents](#)
- [Viewing Model Dependency Diagram](#)
- [Internationalization and Localization Support](#)

Prerequisite

[Django framework](#) and the corresponding [Python](#) interpreter are properly installed on your machine.

Supported versions of Django and Python

PyCharm supports the latest Django versions. The corresponding Python versions depend on Django. See [What Python version can I use with Django?](#)

Django support

Django support in PyCharm includes:

- Dedicated [project type](#).
- Ability to [run the tasks](#) of the `manage.py` utility.
- Django templates support (syntax and error highlighting, code completion, navigation, completion for block names, resolve and completion for custom tags and filters, and quick documentation for tags and filters).
- Ability to [create templates from usage](#).
- Ability to [debug Django templates](#).
- Live templates (snippets) for the quick development of Django templates.
- Run/debug configuration for [Django server](#).
- [Navigation between views and templates](#).
- Code insight support for Django ORM.
- [Code completion](#) and resolve in
 - `views.py` and `urls.py` files:

```
class AboutView|
  template_ @ TemplateView      django.views.generic
  @ abs(number)                 _builtin_
  @ all(iterable)               _builtin_
```

- Models:

```
class Ox(models.M
  @ Model      django.db.models
  @ Manager    django.db.models
  @ ManyToManyField  django.db.models
```

- Meta model options:

```
class Meta:
  ordering = ["horn_length"]
  vnp
  verbose_name_plural
  Press Ctrl+Enter to choose the first suggestion >>
```

- Class-based views. PyCharm provides [Intention action](#) to convert Django function-based generic views to class-based views.
- [Generating model dependency diagrams](#) for Django models.

Enabling or disabling Django support

Django support can be turned on or off by selecting/clearing the check box Enable Django support in the [Django page](#).

To enable Django support, follow these steps:

1. Open Settings/Preferences dialog, and click the page [Django](#).
2. Make sure that the check box Enable Django support is selected.
3. Apply changes (if any) and close the dialog.

Creating Django Project

This feature is supported in the Professional edition only.

[Django project](#) is intended for productive web development with Django. PyCharm takes care of creating specific directory structure and files required for a Django application, and providing the correct settings.

To create a Django project

1. On the main menu, choose File | New | Project, or click the New Project button in the [Welcome screen](#). Create New Project dialog box opens.
2. In the [Create New Project](#) dialog box, specify the following:
 - Project name and location.
 - Project type Django project.
 - In the Python Interpreter drop-down list, select the Python SDK you want to use. If the desired interpreter is not found in the list, click  and choose the interpreter type.
Refer to the section [Configuring Available Python Interpreters](#).
 - If Django is missing in the selected interpreter, PyCharm displays an information message that Django will be downloaded.
3. Click  (More Settings), and specify the following:
 - The Django application name.

Note The name of a Django application should not be the same as the Django project name.
 - The directory where the templates will be stored.
 - If necessary, select the check box Enable Django admin.
4. Click Create.

Creating Django Application in a Project

This feature is supported in the Professional edition only.

To add a new site to your existing Django project, use the `startapp` task of `manage.py` utility.

To add a new Django application to an existing project

1. On the main menu, choose Tools | Run manage.py task
2. In the Enter manage.py task name dialog box, type `startapp`. Note suggestion list that appears under the dialog box after entering the first letter, and shrinks as you type to show the exact match only.
3. In the dialog box that opens, type the name of the new Django application.

Creating Templates

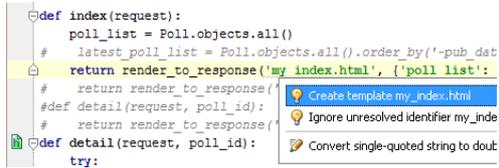
This feature is supported in the Professional edition only.

The storage for templates can be specified on creating a Django project or a Google App Engine project ; at a later time, one can configure the directories for templates, using the [Python Template Languages](#) page of the Settings/Preferences dialog.

To create a template for a view

Suppose you reference a template file that doesn't yet exist. PyCharm marks such a reference as unresolved, and provides an intention action to create a template file "from usage".

1. Place the caret at the unresolved reference to a template.
2. Press `Alt+Enter`, or click the yellow light bulb to show the list of available intention actions.
3. From the suggestion list, choose action Create template <name>:



```
def index(request):
    poll_list = Poll.objects.all()
    # latest_poll_list = Poll.objects.all().order_by('-pub dat
    return render_to_response('my_index.html', {'poll_list':
    # return render_to_response(
    #def detail(request, poll_id):
    # return render_to_response(
def detail(request, poll_id):
    try:
```

The screenshot shows a code editor with a Python function `index`. The line `return render_to_response('my_index.html', {'poll_list':` has a yellow lightbulb icon above it. A tooltip menu is open, showing the following options:

- Create template my_index.html
- Ignore unresolved identifier my_inde
- Convert single-quoted string to doub

Create Template dialog box appears, showing the read-only template name (Template path field), and a drop-down list of possible template locations (Templates root field).

Note that PyCharm automatically discovers the directories specified in the `TEMPLATE_DIRS` or `TEMPLATE_LOADERS` fields of the `settings.py` file. You can specify the other directories in addition.

4. In the Create Template dialog box, select the template directory, where the new template will be created.
The Template root field provides a list of possible locations for the new template. This list includes the template directories specified in the [Python Template Languages](#) page of the Settings dialog, plus the directories, which are automatically detected in the `TEMPLATE_DIRS` or `TEMPLATE_LOADERS` variables of the `settings.py` file.
5. Click OK. The empty `*.html` file with the name in question is created in the specified location.

Debugging Django Templates

This feature is supported in the Professional edition only.

In this section:

- [Introduction](#)
- [Prerequisite](#)
- [Debugging a Django template](#)

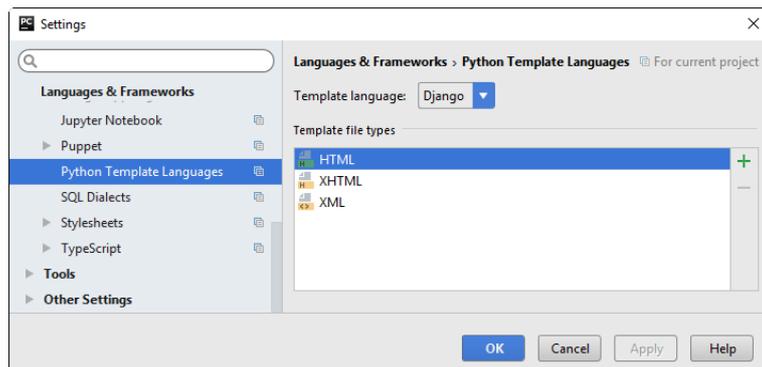
Introduction

PyCharm allows placing breakpoints to the lines of Django template files, at the lines with Django tags or expressions.

Prerequisite

Django is specified as the project [template language](#).

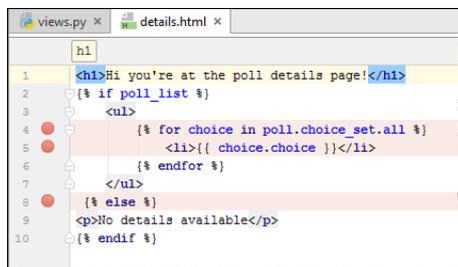
To do that, open the Project Settings dialog, under the Project Settings select page [Python Template Languages](#), select project where the templates reside, and then choose Django from the Template Language drop-down list.



Debugging a Django template

To debug a Django template, follow these general steps

1. Place breakpoints in the desired lines of the Django templates:



2. Launch Django server in the Debug mode.

This is how it's done:

1. On the main menu, choose Run | Edit Configurations.
2. If the desired [Django Server](#) run/debug configuration exists, select it, otherwise create a new one, as described in the section [Creating and Editing Run/Debug Configuration](#).
3. Then click the button  on the main toolbar, or press `Shift+F9`.

PyCharm will open the template in your browser, and suspend at the breakpoints you've set.

The Debugger session starts, and Debug tool window appears.

3. In the [Debug tool window](#), you can:
 - Examine the rendering contexts in the [Variables pane](#).
 - [Step through the breakpoints](#) defined within the Django template.
 - Use the debugging [console](#).

- If you want to trace back exceptions that are raised in course of template debugging, [open](#) Breakpoints dialog, and in the Django Exception Breakpoints tab, select the check box Suspend.

- Debugging Jinja and Mako templates is not supported.

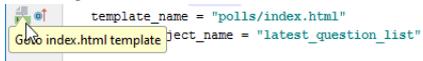
Navigating Between Templates and Views

This feature is supported in the Professional edition only.

PyCharm makes it possible to easily navigate between templates and views, using the gutter icons  and .

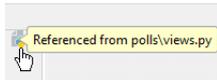
To navigate from a view to a template

1. Open `views.py` file in the editor.
2. Click the gutter icon next to the desired function:

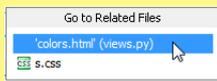


To navigate from a template to the referencing view

1. Open the desired template file in the editor.
2. Click the gutter icon:



Tip If there are more related files (for example, a view and a style sheet), select the desired target from the pop-up that appears:

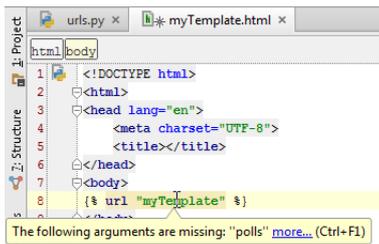


Navigation and Completion for Named URL Tags in Django Templates

This feature is supported in the Professional edition only.

PyCharm provides extensive support for the [named url tags](#). This support includes:

- Django inspection that checks whether the required arguments are passed correctly to a `{% url %}` tag:

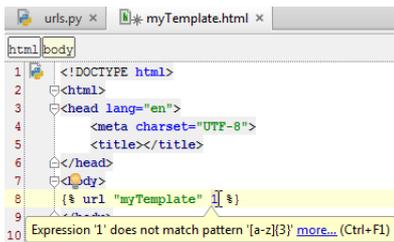


The screenshot shows a Django template with the following code:

```
1 <!DOCTYPE html>
2 <html>
3 <head lang="en">
4 <meta charset="UTF-8">
5 <title></title>
6 </head>
7 <body>
8 {% url "myTemplate" %}
```

A warning message is displayed: "The following arguments are missing: 'polls' more... (Ctrl+F1)".

- Django inspection checks argument values against regex groups and adds warning if a group does not match:

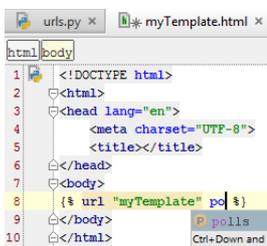


The screenshot shows a Django template with the following code:

```
1 <!DOCTYPE html>
2 <html>
3 <head lang="en">
4 <meta charset="UTF-8">
5 <title></title>
6 </head>
7 <body>
8 {% url "myTemplate" "l" %}
```

A warning message is displayed: "Expression 'l' does not match pattern '[a-z]{3}' more... (Ctrl+F1)".

- Suggestion list on code completion includes names for the named arguments (if any):

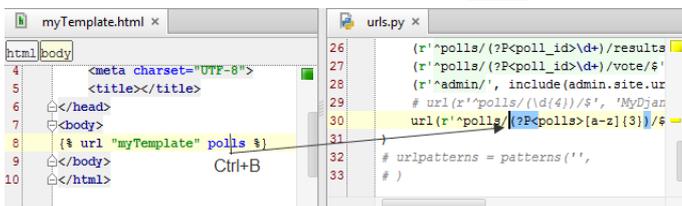


The screenshot shows a Django template with the following code:

```
1 <!DOCTYPE html>
2 <html>
3 <head lang="en">
4 <meta charset="UTF-8">
5 <title></title>
6 </head>
7 <body>
8 {% url "myTemplate" poll %}
```

A suggestion list is shown for the argument 'poll', with 'polls' as a suggestion.

- Ability to [navigate to an appropriate regex group](#) by pressing `Ctrl+B` on a url argument:



The screenshot shows two windows. The left window shows a Django template with the following code:

```
1 <!DOCTYPE html>
2 <html>
3 <head lang="en">
4 <meta charset="UTF-8">
5 <title></title>
6 </head>
7 <body>
8 {% url "myTemplate" polls %}
```

The right window shows the `urls.py` file with the following code:

```
26 ('^polls/(?P<poll_id>d+)/results'
27 ('^polls/(?P<poll_id>d+)/vote/$'
28 ('^admin/', include(admin.site.urls))
29 # url(r'^polls/(?P<poll_id>d+)/$', 'MyDjango')
30 url(r'^polls/(?P<polls>[a-z]{3})/$',
31 )
32 # urlpatterns = patterns('',
33 # )
```

An arrow points from the `polls` argument in the template to the `polls` group in the URL pattern.

Navigating to Implementing Blocks of Templates

This feature is supported in the Professional edition only.

In this section:

- [Introduction](#)
- [Jumping to an implementing block](#)

Introduction

If a template extends another template, PyCharm provides the possibility to navigate from the blocks in a base template to the implementing templates, using the gutter icons .

While you hover the mouse pointer over this icon, a tooltip informs that the block is inherited.

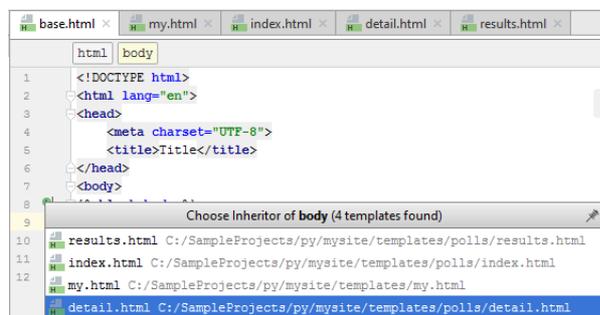


Jumping to an implementing block

To navigate from a block in an extended template to the implementing block

1. Open for editing the desired base template.
2. Select the block in question, and click the gutter icon .
3. If a single implementation block is encountered, PyCharm navigates directly to this block.

If more than one template implements same base template, choose the target from the pop-up menu:



The implementing template opens in the editor, with the caret at the name of the block in question.

Tip Use the regular navigation to declaration (`Ctrl+B`) to jump to the extended template.

Running Tasks of manage.py Utility

This feature is supported in the Professional edition only.

In this section:

- [Overview](#)
- [Configuring manage.py utility](#)
- [Running manage.py utility](#)
- [Working in the manage.py utility console](#)
- [Handling error](#)

Overview

With PyCharm, you can run Django manage.py utility from within the IDE. Each task of this utility is executed in the [manage.py](#) console.

Note that Run manage.py task command is available for both local and remote interpreters.

Configuring manage.py utility

It's important to note that configuration of the `manage.py` utility is done in the [Django page](#) of the Settings/Preferences dialog .

To configure manage.py utility, follow these steps

1. [Open](#) the Settings/Preferences dialog, and then under Languages and Frameworks node, click [Django](#).
2. In this page, choose the desired Django project.
3. In the Manage.py tasks section, specify the following:
 - In the field Manage script, specify the desired `manage.py` script.

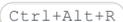
Note that by default PyCharm shows the `manage.py` script that resides under the Django project root. If you are not happy with this suggestion, you can choose any other `manage.py` script by clicking the browse button .

- In the Environment variables field, specify the environment variables to be passed to the script. By default, this field is empty. Click the browse button  to open the Environment Variables dialog box. Use the toolbar buttons to make up the list of variables.

If you want to see the system environment variables, click Show link in this dialog box.

Running manage.py utility

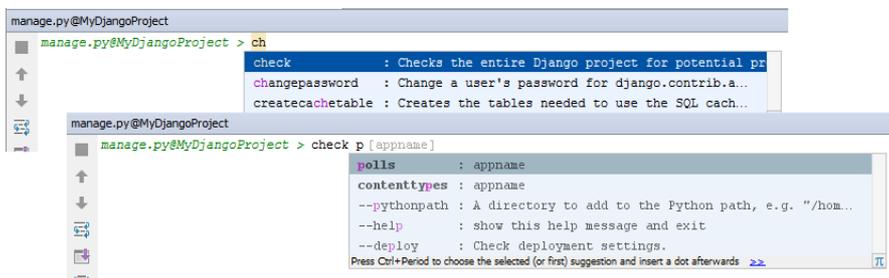
To run a task of the manage.py utility

1. On the main menu, choose Tools | Run manage.py task , or press  .
The `manage.py` utility starts in its own console.
2. Type the name of the desired task.

Working in the manage.py utility console

In the `manage.py` console, one can:

- Scroll through the history of executed commands using the up and down arrow keys.
- Use [code completion](#) ():



- View [quick documentation](#) ():



Handling error

PyCharm smartly handles errors. When your Django project can't run due to an error, this error displays in the `manage.py` console instead of the command line:

Django errors

```
Failed to get real commands on module 'py': python process died with code 1: Traceback (most recent call last):
File "C:\Program Files (x86)\JetBrains\PyCharm
4.5\helpers\pycharm\django_manage_commands_provider\provider.py", line 20, in <module>
django.setup()
File "C:\Python34\lib\site-packages\django\__init__.py", line 18, in setup
apps.populate(settings.INSTALLED_APPS)
File "C:\Python34\lib\site-packages\django\apps\registry.py", line 108, in populate
app_config.import_models(all_models)
File "C:\Python34\lib\site-packages\django\apps\config.py", line 198, in import_models
self.models_module = import_module(models_module_name)
File "C:\Python34\lib\importlib\__init__.py", line 109, in import_module
return _bootstrap._gcd_import(name[level:], package, level)
File "<frozen importlib._bootstrap>", line 2254, in _gcd_import
File "<frozen importlib._bootstrap>", line 2237, in _find_and_load
File "<frozen importlib._bootstrap>", line 2226, in _find_and_load_unlocked
File "<frozen importlib._bootstrap>", line 1200, in _load_unlocked
File "<frozen importlib._bootstrap>", line 1129, in _exec
File "<frozen importlib._bootstrap>", line 1471, in exec_module
File "<frozen importlib._bootstrap>", line 321, in _call_with_frames_removed
File "C:\SampleProjects\py\MyDjangoApp\models.py", line 4, in <module>
raise Exception("Failed to load models")
Exception: Failed to load models
```

Referring to Static Contents

This feature is supported in the Professional edition only.

In this section:

- [Prerequisites](#)
- [Referring to static contents](#)

Prerequisites

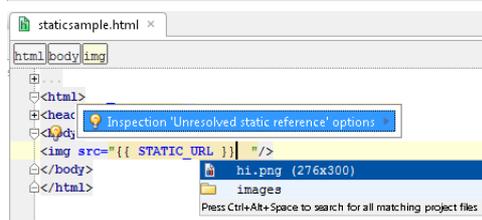
Before referring to static contents, make sure that the following prerequisites are met:

- 'django.contrib.staticfiles', is added to `INSTALLED_APPS` in the `settings.py` file of your application.
- The directory with the static contents (for example, images), named `static`, resides under the application root.

Note that you can configure the directory for the static content in the `settings.py` file, by changing the `STATICFILES_DIRS` field.

To refer to static contents, follow these general steps

1. In the [pre-configured directory for templates](#), create a template file (`Alt+Insert` - HTML/XHTML file).
2. After the template tag `STATIC_URL`, start typing, and then use `Ctrl+Space`. Inspection that detects unresolved references to the static contents is also available.

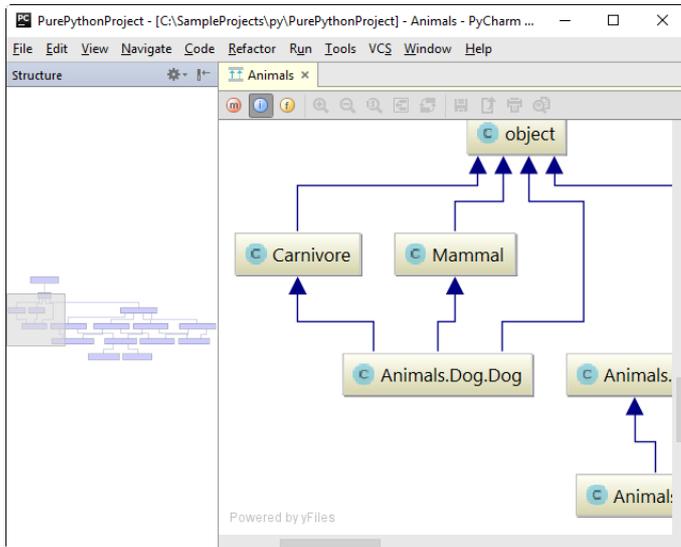


Viewing Model Dependency Diagram

Model dependency diagram enables you to get an overview of the models within your Django application, and analyze their relationships.

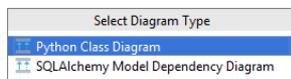
Model dependency diagrams are available for:

- Django models
- Google App Engine models
- SQLAlchemy



To open the Model Dependency diagram of a project

1. Do one of the following:
 - On the main menu, choose View | Show Model Dependency Diagram.
 - On the context menu of the Project tool window, or the editor, point to Diagrams, and choose Show Diagram or Show Diagram Pop-up.
 - Press `Ctrl+Shift+Alt+U`, or `Ctrl+Alt+U`.
2. Select the type of diagram from the pop-up window:



Tip If you invoke Model Dependency diagram for a specific model, the diagram will open with the model in question centered and having the focus, and zoomed to actual size.

In the Model Dependency diagram, you can perform the following operations

- Select elements.
- Add notes, delete elements.
- Change diagram layout.
- Change diagram scale.
- Navigate to source code.
- Navigate through the models using the Structure view (`Ctrl+F12`).
- Find usages of the selected node element.
- Invoke refactoring commands.

Tip Keeping the `Alt` key pressed invokes the magnifier tool, which will help you have a closer look at the most interesting or problematic areas of your Model dependency diagram.



Internationalization and Localization Support

This feature is supported in the Professional edition only.

In this section:

- Internationalization and Localization Support
 - [Overview](#)
 - [i18n-related features](#)
 - [Prerequisites](#)
- [Creating Message Files](#)
- [Extracting Blocks of Text from Django Templates](#)
- [Compiling Message Files](#)
- [Navigating Between Text and Message File](#)

Overview

i18n support with PyCharm falls into the following major aspects:

- Internationalization, which involves extracting strings out of your source code and presenting them as properties that are further referenced in the source code.
- Localization, which means translating these properties into the target languages.

Note i18n support is available for the Django applications only.

i18n-related features

PyCharm provides helpful features that simplify working on software internationalization and localization issues. These features are:

- Possibility to run i18n-related tasks of the `manage.py` utility.
- Syntax highlighting in `*.po` files.
- Intention action to surround blocks of text in Django templates.
- Navigation between blocks of text and locales.

Prerequisites

i18n support is available for the Django applications.

- `gettext` utilities are downloaded and installed on your machine.
- `locale` directory is created in the project root.
- Django is the [project template language](#).

Creating Message Files

This feature is supported in the Professional edition only.

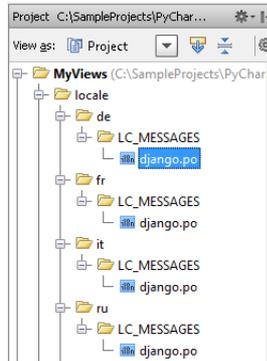
PyCharm stores locales in the language-related subdirectories of the `locale` directory. For creating locales, run the `makemessages` task of the `manage.py` utility.

To create a message file

1. On the main menu, choose Tools | Run manage.py task, or press `Ctrl+Alt+R`.
2. In the `manage.py` task window, enter `makemessages`, type `--locale <locale name >` and press `Enter`.

Repeat this step for each locale you want to create.

If there are strings marked for localization, PyCharm will produce a directory and `django.po` file for each locale:



If there are no such strings, only an empty directory structure is created.

Extracting Blocks of Text from Django Templates

This feature is supported in the Professional edition only.

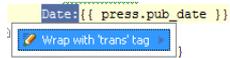
PyCharm provides a dedicated intention action to wrap strings in Django templates in `{% trans %}`, or `{% blocktrans %}` tags.

The lines with `i18n` tags are marked with  icon in the gutter.

To wrap block of text in translation tags

1. Open the desired Django template for editing, and select strings to be marked for translation.

2. Press `Alt+Enter`, or click the light bulb to reveal the list of available intention actions:



3. Select intention action Wrap with 'trans' tag, and press `Enter`. PyCharm wraps selected text in translation tags, and adds `{% load i18n %}`, if extracting text is performed for the first time.

Compiling Message Files

This feature is supported in the Professional edition only.

For compiling locales, run the `compilemessages` task of the `manage.py` utility.

To compile a message file

1. On the main menu, choose Tools | Run manage.py task, or press `N/A`.
2. In the Enter manage.py task name dialog box, select `compilemessages`, and press `Enter`.
`django.mo` files are produced for each locale.

This feature is supported in the Professional edition only.

Use the  gutter icons to navigate from a template to a localization file. To jump from localization file to the corresponding template, use `Ctrl+Click`.

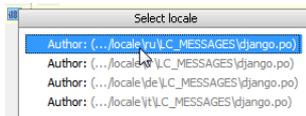
In this section:

- [Navigating from template to locale](#)
- [Navigating from locale to template](#)
- [Viewing references](#)

Navigating from template to locale

To navigate from a template to locale

1. Click  icon in the gutter next to the desired tag.
2. If a tag is referenced from several locales, select one from the pop-up window:



The selected `django.po` file opens in the editor, with the caret resting at the `msgid` that corresponds to the tag in question.

Navigating from locale to template

Note According to the Django documentation, each `django.po` file contains comments with the path a template above each `msgid`.

To navigate from a locale to template

1. In the desired `django.po` file, place the caret at the comment above the locale in question:

```
#: templates\my_view\detail.html:11
msgid "Author:"
msgstr ""
```

2. On the main menu, choose `Navigate | Declaration`, or use any other method, described in [Navigating to Declaration or Type Declaration of a Symbol](#). The corresponding template files opens in the editor.

Viewing references

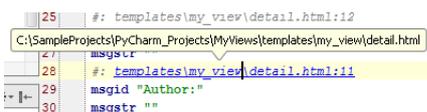
To view references to a localization tag

- Hover your mouse pointer over the gutter icon next to the desired tag. A balloon that pops up, shows a list of locale files that reference the selected tag:



To view which template a locale references

- Keeping `Ctrl` key pressed, hover your mouse pointer over the comment above the locale in question. The comment turns into a hyperlink, and the balloon shows reference:



Buildout

This feature is supported in the Professional edition only.

PyCharm supports `buildout` in the following activities:

- [Django server run/debug configurations](#).
- Django-enabled [console](#).
- [Running the tasks](#) of the `manage.py` utility.

In this section:

- [Prerequisites](#)
- [Buildout Support](#)

Prerequisites

PyCharm recognizes `buildout` support in existing projects. However, you have to perform the following general steps as prerequisites (outside of PyCharm):

1. Add `buildout.cfg` and `setup.py` files to the root directory of your project.
2. Check out `bootstrap.py` and place it to the root directory of your project.
3. Execute `bootstrap.py` to create the `buildout` directory structure, download `buildout.exe`, and the other dependencies.
4. Execute `bin\buildout.exe` script. This will download the dependencies and generate a runner script for each of the parts listed in `buildout.cfg`.

Buildout Support

Buildout files are marked with  icon.

Buildout support includes:

- capability to enable [Buildout support](#) for a project, and specify the script to be used for resolving references in the source code.
- [Code completion](#).
- Error and syntax highlighting.
- Code [formatting](#) and [folding](#).

CoffeeScript Support

This feature is supported in the Professional edition only.

In this section:

- CoffeeScript Support
 - [Overview](#)
 - [Preparing for CoffeeScript development](#)
 - [Coding assistance](#)
- [Compiling CoffeeScript to JavaScript](#)
- [Running CoffeeScript](#)
- [Debugging CoffeeScript](#)

Overview

PyCharm provides [CoffeeScript](#) support. PyCharm recognizes `*.coffee` files, and allows you to edit them providing full range of coding assistance without any additional steps from your side. CoffeeScript files are marked with  icon.

To run, debug, and test your code, you will need it translated into JavaScript which requires a **compiler** and **Node.js**. For more details on CoffeeScript compilation, see [Compiling CoffeeScript to JavaScript](#).

Preparing for CoffeeScript development

1. Make sure the **CoffeeScript** and **Node.js** plugins are **installed** and **enabled**. The plugins are not bundled with PyCharm, but they can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#). Once enabled, the plugins are available at the IDE level, that is, you can use them in all your PyCharm projects.
2. Download and install the [Node.js](#) runtime environment.
3. Configure the Node.js interpreter in PyCharm:
 1. Open the by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Node.js and NPM under Languages & Frameworks.
 2. On the [Node.js and NPM](#) page that opens, specify the location of the desired Node.js interpreter.

See [Configuring Node.js Interpreters](#) for details.

Coding assistance

CoffeeScript support includes:

- [Code completion](#) for keywords, labels, variables, parameters and functions.
- Error and syntax highlighting.
- Code [formatting](#) and [folding](#).
- Refactoring:
 - Common refactoring procedures, such as **extract method**, **inline**, **rename/move**, etc.
 - CoffeeScript-specific refactoring procedures, such as **change signature**, **extract parameter**, **extract variable**.

See [JavaScript-Specific Refactorings](#) for details.

- [Code generation](#)
 - Generating code stubs based on [file templates](#) during file creation.
 - Ability to create [line and block comments](#) (`Ctrl+Slash`) / (`Ctrl+Shift+Slash`).
- Navigation through source code
 - [Navigating with Structure View](#).
 - Navigate | Declaration (`Ctrl+B`).
 - Navigate | Implementation (`Ctrl+Alt+B`) from overridden method / subclassed class.
 - Navigate | Symbol (`Ctrl+Shift+Alt+N`).
- [Compiling to JavaScript](#) for further running, debugging, and testing, see [Running CoffeeScript](#) and [Debugging CoffeeScript](#).
- Executing CoffeeScript files involves:
 - Ability to [preview](#) results of CoffeeScript files compilation to JavaScript.
 - Ability to launch CoffeeScript files from the context menu.
 - [Run/debug configuration](#) for NodeJS includes the ability to use CoffeeScript plugin.

Compiling CoffeeScript to JavaScript

This feature is supported in the Professional edition only.

On this page:

- [Introduction](#)
- [Before you start](#)
- [Installing the CoffeeScript compiler globally](#)
- [Installing the CoffeeScript compiler in a project](#)
- [Creating a File Watcher](#)
- [Examples of customizing the behaviour of a compiler](#)
- [Compiling the CoffeeScript code](#)
- [Previewing the compilation results without running a compiler](#)

Introduction

CoffeeScript code is not processed by browsers that work with JavaScript code. Therefore to be executed, CoffeeScript code has to be translated into JavaScript. This operation is referred to as **compilation** and the tools that perform it are called **compilers**.

PyCharm supports integration with the [coffee-script](#) compilation. The tool translates CoffeeScript code into JavaScript and creates [source maps](#) that set correspondence between lines in your CoffeeScript code and in the generated JavaScript code, otherwise your breakpoints will not be recognised and processed correctly.

In PyCharm, compiler configurations are called **File Watchers**. For each supported compiler, PyCharm provides a predefined **File Watcher** template. Predefined **File Watcher** templates are available at the PyCharm level. To run a compiler against your project files, you need to create a project-specific **File Watcher** based on the relevant template, at least, specify the path to the compiler to use on your machine.

The easiest way to install the CoffeeScript compiler is to use the **Node Package Manager (npm)**, which is a part of [Node.js](#). See [Installing and Removing External Software Using Node Package Manager](#) for details.

Depending on the desired location of the CoffeeScript compiler executable file, choose one of the following methods:

- Install the compiler **globally** at the PyCharm level so it can be used in any PyCharm project.
- Install the compiler in a specific project and thus restrict its use to this project.
- Install the compiler in a project as a [development dependency](#).

In either installation mode, make sure that the parent folder of the CoffeeScript compiler is added to the `PATH` variable. This enables you to launch the compiler from any folder.

PyCharm provides user interface both for **global** and **project** installation as well as supports installation through the command line.

Before you start

1. Download and install [Node.js](#). The runtime environment is required for two reasons:
 - The CoffeeScript compiler is started through **Node.js**.
 - **NPM**, which is a part of the runtime environment, is also the easiest way to download the CoffeeScript compiler.

If you are going to use the command line mode, make sure the path to the parent folder of the **Node.js** executable file and the path to the `npm` folder are added to the `PATH` variable. This enables you to launch the CoffeeScript compiler and **npm** from any folder.

2. Install and enable the **NodeJS** repository plugin as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).
3. Make sure the **File Watchers** plugin is enabled. The plugin is bundled with PyCharm and activated by default. If it is not, enable the plugin. See [Enabling and Disabling Plugins](#) for details.

Installing the CoffeeScript compiler globally

Global installation makes a compiler available at the PyCharm level so it can be used in any PyCharm project. Moreover, during installation the parent folder of the compiler is automatically added to the `PATH` variable, which enables you to launch the compiler from any folder.

– Run the installation from the command line in the **global** mode:

1. Launch the embedded Terminal (View | Tool Windows | Terminal or by hovering your mouse pointer over  in the lower left corner of PyCharm and choosing Terminal from the menu
2. Switch to the directory where **NPM** is stored or define a `PATH` variable for it so it is available from any folder, see [Installing NodeJS](#).
3. Type the following command at the command line prompt:

```
npm install -g coffee-script
```

The `-g` key makes the compiler run in the **global** mode. Because the installation is performed through **NPM**, the CoffeeScript compiler is installed in the `npm` folder. Make sure this parent folder is added to the `PATH` variable. This enables you to launch the compiler from any folder.

For more details on the **NPM** operation modes, see [npm documentation](#). For more information about installing the CoffeeScript compiler, see <https://npmjs.org/package/coffee-script>.

– Run **NPM** from PyCharm using the Node.js and NPM page of the Settings dialog box.

1. Open the [Settings / Preferences Dialog](#) by pressing `(Ctrl+Alt+S)` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for

macOS, and click Node.js and NPM under Languages & Frameworks.

2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the **global** and at the **project** level. Click **+**.
3. In the Available Packages dialog box that opens, select the required package to install.
4. Select the Options check box and type `-g` in the text box next to it.
5. Optionally specify the product version and click Install Package to start installation.

Installing the CoffeeScript compiler in a project

Local installation in a specific project restricts the use of a compiler to this project.

– Run the installation from the command line:

1. Launch the embedded Terminal (View | Tool Windows | Terminal or by hovering your mouse pointer over  in the lower left corner of PyCharm and choosing Terminal from the menu
2. Switch to the project root folder and type the following command at the command line prompt:

```
npm install coffee-script
```

– Run **NPM** from PyCharm using the Node.js and NPM page of the Settings dialog box.

1. Open the **Settings / Preferences Dialog** by pressing `(Ctrl+Alt+S)` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Node.js and NPM under Languages & Frameworks.
2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the **global** and at the **project** level. Click **+**.
3. In the Available Packages dialog box that opens, select the required package.
4. Optionally specify the product version and click Install Package to start installation.

Project level installation is helpful and reliable in **template-based projects** of the type **Node Boilerplate** or **Node.js Express**, which already have the `node_modules` folder. The latter is important because **NPM** installs the CoffeeScript compiler in a `node_modules` folder. If your project already contains such folder, the CoffeeScript compiler is installed there.

Projects of other types or **empty** projects may not have a `node_modules` folder. In this case **npm** goes upwards in the folder tree and installs the CoffeeScript compiler in the first detected `node_modules` folder. Keep in mind that this detected `node_modules` folder may be **outside** your current project root.

Finally, if no `node_modules` folder is detected in the folder tree either, the folder is created right under the current project root and the CoffeeScript compiler is installed there.

In either case, make sure that the parent folder of the CoffeeScript compiler is added to the `PATH` variable. This enables you to launch the compiler from any folder.

Creating a File Watcher

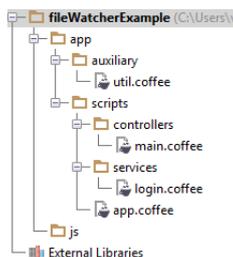
PyCharm provides a common procedure and user interface for creating **File Watchers** of all types. The only difference is in the predefined templates you choose in each case.

1. To start creating a **File Watcher**, open the Settings/Preferences dialog box by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS on the main menu, and then click File Watchers under the Tools node. The **File Watchers page** that opens, shows the list of **File Watchers** that are already configured in the project.
2. Click the Add button **+** or press `(Alt+Insert)` and choose the CoffeeScript predefined template from the pop-up list. Your code will be translated to JavaScript and supplied with generated [source maps](#).
3. In the Program text box, specify the path to the `coffee.cmd` file. Type the path manually or click the Browse button  and choose the file location in the dialog box that opens.
4. Proceed as described on page [Using File Watchers](#).

Examples of customizing the behaviour of a compiler

Any **compiler** is an external, third-party tool. Therefore the only way to influence a **compiler** is pass arguments to it just as if you were working in the command line mode. Below are two examples of customizing the default output location for the **CoffeeScript compiler**.

Suppose, you have a project with the following folder structure:



By default, the generated files will be stored in the folder where the original file is. You can change this default location and have the generated files stored in the `js` folder. Moreover, you can have them stored in a flat list or arranged in the folder structure that repeats the original structure under the `app` node.

To have all the generated files stored in the output `js` folder without retaining the original folder structure under the `app` folder

to have all the generated files stored in the output `js` folder without retaining the original folder structure under the `app` folder.

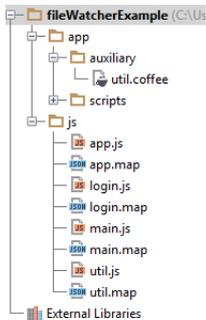
1. In the Arguments text box, type:

```
--output $ProjectFileDir$\js\ --compile --map $FileName$
```

2. In the Output paths to refresh text box, type:

```
$ProjectFileDir$\js\${FileNameWithoutExtension}.js:$ProjectFileDir$\js\${FileNameWithoutExtension}.map
```

As a result, the project tree looks as follows:



– To have the original folder structure under the `app` node retained in the output `js` folder:

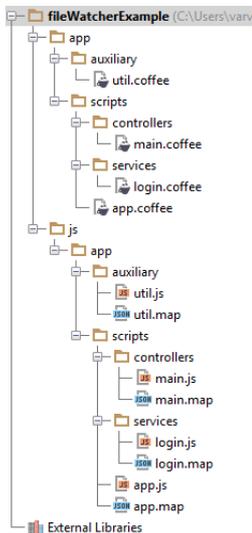
1. In the Arguments text box, type:

```
--output $ProjectFileDir$\js\${FileDirRelativeToProjectRoot}\ --compile --map $FileName$
```

2. In the Output paths to refresh text box, type:

```
$ProjectFileDir$\js\${FileDirRelativeToProjectRoot}\${FileNameWithoutExtension}.js:$ProjectFileDir$\js\${FileDirRelativeToProjectRoot}
```

As a result, the project tree looks as follows:



Compiling the CoffeeScript code

When you open a CoffeeScript file, PyCharm checks whether an applicable file watcher is available in the current project. If such file watcher is configured but disabled, PyCharm displays a pop-up window that informs you about the configured file watcher and suggests to enable it.

If an applicable file watcher is configured and enabled in the current project, PyCharm starts it automatically upon the event specified in the [New Watcher dialog](#).

- If the Immediate file synchronization check box is selected, the **File Watcher** is invoked as soon as any changes are made to the source code.
- If the Immediate file synchronization check box is cleared, the **File Watcher** is started upon save (File | Save All, **Ctrl+S**) or when you move focus from PyCharm (upon frame deactivation).

The **compiler** stores the generated output in a separate file. The file has the name of the source **CoffeeScript** file and the extension `js` or `js.map` depending on the **compiler** type. The location of the generated files is defined in the Output paths to refresh text box of the [New Watcher dialog](#). Based on this setting,

PyCharm detects the **compiler** output. However, in the Project Tree, they are shown under the source `.coffee` file which is now displayed as a node.

Previewing the compilation results without running a compiler

PyCharm can perform static analyses of your CoffeeScript code without actually running a compiler and display the predicted compilation output in the dedicated read-only viewer.

1. Open the desired CoffeeScript file in the editor, and right-click the editor background.
2. On the context menu, choose Preview Compiled CoffeeScript File. The preview is opened in the dedicated read-only viewer; the left-hand pane shows the original CoffeeScript source code and the right-hand pane shows the JavaScript code that will be generated by the compiler when it runs.

Running CoffeeScript

This feature is supported in the Professional edition only.

On this page:

- [Introduction](#)
- [Compiling CoffeeScript manually and running the generated JavaScript code](#)
- [Compile CoffeeScript on the fly during run](#)

Introduction

CoffeeScript code is not processed by browsers that work with JavaScript code. Therefore to be executed, CoffeeScript code has to be translated into JavaScript. This operation is referred to as **compilation** and the tools that perform it are called **compilers**.

For more details about **compilation** in PyCharm, see the section [Using File Watchers](#).

In either case, running CoffeeScript is supported only in the **local** mode. This means that PyCharm itself starts the Node.js engine and the target application according to a [run configuration](#) and gets full control over the session.

For more details about running Node.js applications, see [Running and Debugging Node.js](#).

There are two approaches to running CoffeeScript in PyCharm:

- Compile the CoffeeScript code manually and then run the output JavaScript code as if it were a Node.js application.
- Run the original CoffeeScript code through the NodeJS run configuration and have PyCharm compile it on the fly.

Compiling CoffeeScript manually and running the generated JavaScript code

1. [Compile the CoffeeScript code into Javascript](#).
2. [Start creating a Node.js run configuration](#) with the following mandatory settings:
 1. The Node.js engine to use. By default, the field shows the path to the interpreter specified on the [Node.js](#) page during Node.js configuration.
 2. In the Working directory field, specify the location of the files referenced from the starting CoffeeScript file to run, for example, **includes**. If this file does not reference any other files, just leave the field empty.
 3. In the Path to Node App JS File text box, specify the full path to the JavaScript file that was generated from the original CoffeeScript file during the compilation.
3. Save the configuration and click  on the toolbar.
4. Proceed as while [running a Node.js application](#).

Compile CoffeeScript on the fly during run

1. This mode requires that the `register.js` file, which is a part of the `coffee-script` package, should be located inside the project. Therefore you need to install the `coffee-script` package on the [Node.js](#) page locally, as described in [Installing and Removing External Software Using Node Package Manager](#).
2. Open the starting CoffeeScript file in the editor or select in the Project tool window and choose Create <CoffeeScript_file_name> on the context menu. Alternatively, start creating a Node.js run configuration as described in [Running and Debugging Node.js](#). In the [Run/Debug Configuration: Node JS](#) dialog that opens, specify the following mandatory settings:
 1. The Node interpreter to use. Select the relevant interpreter configuration or create a new one, see By default, the field shows the path to the interpreter specified on the [Node.js](#) page during Node.js configuration.
For Linux and macOS, this setting is overridden by the Node.js from the path to the CoffeeScript compiler executable file.
 2. In the Node parameters text box, type the following:

```
--require coffee-script/register
```
 3. In the Working directory field, specify the [working directory](#) of the application. All references in the **starting CoffeeScript file**, for example, **imports**, will be resolved relative to this folder, unless such references use full paths.
By default, the field shows the **project root folder**. To change this predefined setting, choose the desired folder from the drop-down list, or type the path manually, or click the Browse button  and select the location in the dialog box, that opens.
 4. In the JavaScript file text box, specify the full path to the CoffeeScript file to run.

Note that all the mandatory fields will be filled in automatically if you create a run configuration directly from the required CoffeeScript file.

3. Save the configuration and click  on the toolbar.
4. Proceed as while [running a Node.js application](#).

Debugging CoffeeScript

This feature is supported in the Professional edition only.

CoffeeScript code is not processed by browsers that work with JavaScript code. Therefore to be executed, CoffeeScript code has to be translated into JavaScript. This operation is referred to as **compilation** and the tools that perform it are called **compilers**.

To debug CoffeeScript in PyCharm, you need **source maps** generated in addition to the JavaScript code. [Source maps](#) set correspondence between lines in your CoffeeScript code and in the generated JavaScript code, otherwise your breakpoints will not be recognised and processed correctly. JavaScript and source maps are generated by compiling the CoffeeScript code manually using the File Watcher of the type **CoffeeScript**. After that you can debug the output JavaScript code as if it were a Node.js application.

For more details about **compilation** in PyCharm, see the section [Using File Watchers](#).

Debugging CoffeeScript is supported only in the **local** mode. This means that PyCharm itself starts the Node.js engine and the target application according to a [run configuration](#) and gets full control over the session.

For more details about debugging Node.js applications, see [Running and Debugging Node.js](#).

1. Set the [breakpoints](#) in the CoffeeScript code, where necessary.
2. [Compile the CoffeeScript code into Javascript](#) using the File Watcher of the type **CoffeeScript Source Map**.
3. [Start creating a Node.js run configuration](#) with the following mandatory settings:
 1. The Node.js engine to use. By default, the field shows the path to the interpreter specified on the [Node.js](#) page during Node.js configuration.
 2. In the Working directory field, specify the location of the files referenced from the starting CoffeeScript file to run, for example, **includes**. If this file does not reference any other files, just leave the field empty.
 3. In the Path to Node App JS File text box, specify the full path to the JavaScript file that was generated from the original CoffeeScript file during the compilation.
4. Save the configuration and click  on the toolbar.
5. Proceed as while [debugging a Node.js application locally](#).

Testing CoffeeScript

This feature is supported in the Professional edition only.

To run unit **CoffeeScript** unit test, you need to use dedicated packages that

Cython Support

This feature is supported in the Professional edition only.

In this section:

- [Prerequisites](#)
- [Cython support](#)

Prerequisites

PyCharm provides initial [Cython](#) support out-of-the-box. PyCharm recognizes `*.pyx`, `*.pxd`, and `*.pxi` files, and allows you to edit them.

However, if you want to compile and run `*.pyx` files, make sure that the following prerequisites are met:

- [Cython](#) is downloaded and installed on your computer.
- C compiler is downloaded and installed on your computer.

Cython support

Cython files are marked with  icon.

Cython support includes:

1. Coding assistance:
 - Error and syntax highlighting.
 - [Code completion](#) for keywords, fields of structs, and attributes of extension types.
 - Code [formatting](#) and [folding](#).
 - Ability to create [line comments](#) (`Ctrl+Slash`).
 - Cython syntax for [typed memoryviews](#).
2. [Code inspections](#). Almost all Python code inspections work for Cython.
3. [Refactorings](#).
4. Numerous ways to [navigate](#) through the source code, among them:
 - [Navigating with Structure View](#).
 - [Navigate | Declaration](#) (`Ctrl+B`).
 - [Navigate | Implementation](#) (`Ctrl+Alt+B`) from overridden method / subclassed class.
5. Advanced facilities to [search through the source code](#), in particular, [finding usages](#).
6. Compiling Cython modules:
 - Compilation is done using external tools. The preferred build systems (`Makefile`, `setup.py`, etc.) should be [configured as external tools](#).
 - C compiler should be downloaded and installed on your computer.
7. Cython [debugger](#). Refer to [Cython documentation](#) for details.

Databases and SQL

This feature is supported in the Professional edition only.

In this section:

- Databases and SQL
 - [Databases and SQL support](#)
- [Managing Data Sources](#)
- [Working with the Database tool window](#)
- [Working with Database Consoles](#)
- [Working with the Table Editor](#)
- [Running SQL Script Files](#)
- [Running Injected SQL Statements](#)
- [Using language injections in SQL](#)
- [Extending the functionality of database tools](#)

Databases and SQL support

PyCharm features for working with databases and SQL include:

- Integration with the most popular database management systems such as [Oracle](#), [PostgreSQL](#), [MySQL](#), [SQL Server](#) and others. To be able to work with your databases, you should define them as data sources. See [Managing Data Sources](#).
- Database tool window for managing data structures in your databases (View | Tool Windows | Database). See [Working with the Database tool window](#).
- Database consoles that let you compose and execute SQL statements as well as analyze and modify retrieved data ([Ctrl+Shift+F10](#) in the Database tool window). See [Working with Database Consoles](#).
- Table editor that provides a GUI for working with table data ([F4](#) in the Database tool window). See [Working with the Table Editor](#).
- SQL code generation and editing features in the database consoles and the editor, e.g.
 - Predefined code snippets (a.k.a. live templates) such as for `CREATE TABLE`, `SELECT`, `INSERT`, `UPDATE` and other statements ([Ctrl+J](#)).
 - Auto-completion and highlighting of SQL keywords, and table and column names.
 - Data type prompts for columns ([Ctrl+P](#)).

Standardized and DBMS-specific SQL dialects are supported.

- Structure view for tables in the table editor and Database Console tool window as well as for open database consoles and SQL files ([Ctrl+F12](#)). See e.g. [Using the Structure view to sort data, and hide and show columns](#).
- Quick documentation view for database objects and table cells ([Ctrl+Q](#)). See e.g. [Using the quick documentation view](#).
- Navigation capabilities, e.g.
 - From a table or column reference to its definition: [Ctrl+B](#).
 - To the view of a table or column in the Database tool window: [Alt+F1](#) | Database View.
 - By means of the navigation bar: [Alt+Home](#).
 - By means of the Switcher: [Ctrl+Tab](#).
- Database diagrams ([Ctrl+Alt+U](#) or [Ctrl+Shift+Alt+U](#) in the Database tool window).

Managing Data Sources

This feature is supported in the Professional edition only.

On this page:

- [Data sources](#)
- [Defining a database as a data source](#)
- [Creating a DB data source for H2 or SQLite by means of drag and drop](#)
- [Creating DB data sources by importing connection settings](#)
- [Creating a DDL data source](#)
- [Creating a DDL data source by means of drag and drop](#)
- [Changing data source settings](#)
- [Making a DB data source available in all your projects](#)
- [Removing data sources](#)

Data sources

To be able to work with your databases in PyCharm, you should define them as data sources.

In addition to data sources that correspond to real databases (DB data sources), PyCharm also supports DDL data sources. These are represented by one or more SQL files containing data definition language statements (SQL DDL statements).

Metaphorically, DDL data sources function as databases without data.

Data sources provide the basis for SQL coding assistance and code validation.

Defining a database as a data source

1. [Open the Database tool window](#) and click  on the toolbar.
2. In the Data Sources and Drivers dialog that opens, click **+** and select the database management system (DBMS) that you are using.
3. In the Name field, if necessary, edit the name of the data source.
4. If there is the message Download missing driver files in the lower part of the dialog, specify the driver files.
(To interact with a database, PyCharm needs a database driver. The drivers, generally, are DBMS-specific.)

Do one of the following:

- To download the necessary driver, click the Download link.
- To specify the driver files that you already have available on your computer, click the `<DriverName>` link to the right of Driver.
On the page where the driver settings are shown, under JDBC drivers / Additional, click **+** and select the files in the dialog that opens.

Go back to the page with the data source settings.

5. Specify the database settings.
For most of the DB management systems, these are the database host name (or IP address), port, the database name, and also your database user name and password.

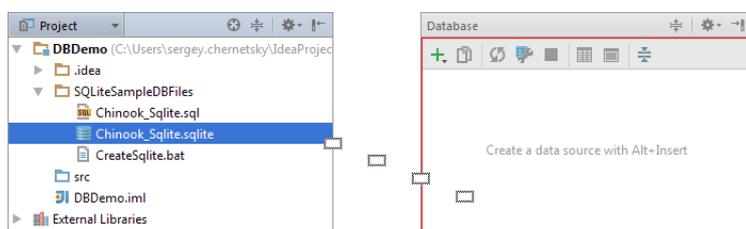
If your database is a local file or directory (which may be the case for SQLite, H2, Derby or HSQLDB), you should specify the location of that file or directory. To do that, click  to the right of the File or Path field and select the database file or directory in the dialog that opens.

6. To access your database using SSH or SSL, select the SSH/SSL tab and specify the corresponding [settings](#) there.
7. To make sure that the settings are correct and PyCharm can properly interact with your database, click Test Connection on the General tab.
8. Click OK to save the settings and close the dialog.
As a result, your new data source is shown in the Database tool window, and the input pane of the associated database console opens.

Creating a DB data source for H2 or SQLite by means of drag and drop

If you have H2 or SQLite database files available locally, you can create DB data sources for them by dragging the files to the Database tool window. The files can be dragged from the Project tool window, or from your file manager (e.g. Explorer or Finder).

1. If the database files are in your project folder, open the Project tool window. Otherwise, open your file manager.
2. [Open the Database tool window](#).
3. Select the file or files of interest in the Project tool window, or in your file manager.
4. Drag the selected file or files into the Database tool window. (For each of the files a separate data source will be created.)



5. If you don't have the necessary database driver files yet, you can download them now. Click  on the toolbar of the Database tool window. (Alternatively, select Properties from the context menu.)
6. In the Data Sources and Drivers dialog that opens, within the line Download missing driver files, click the Download link.

- Click Test Connection to make sure that PyCharm can properly communicate with the database.
- Click OK in the Data Sources and Drivers dialog.

Creating DB data sources by importing connection settings

Files that contain database connection settings (e.g. `settings.py`) can be used for creating DB data sources.

- If the files that you want to import the settings from are not in your project yet, copy them there.
- [Open the Database tool window](#).
- Do one of the following:
 - Click **+** on the toolbar and select Import from sources.
 - Right-click the area under the toolbar or any of the existing data sources, point to New and click Import from sources.

The Data Sources and Drivers dialog opens. The names of candidate data sources are shown in the left-hand pane in green.
- Specify the driver files if they are missing.

Do one of the following:

 - To download the necessary driver, click the Download link.
 - To specify the driver files that you already have available on your computer, click the <DriverName> link to the right of Driver.

On the page where the driver settings are shown, under JDBC drivers / Additional, click **+** and select the files in the dialog that opens.

Go back to the page with the data source settings.
- Click Test Connection to make sure that PyCharm can properly communicate with the database.
- Click OK in the Data Sources and Drivers dialog.

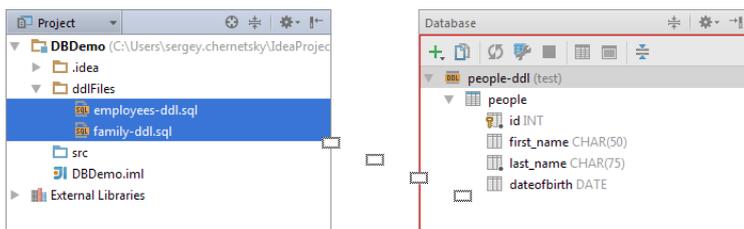
Creating a DDL data source

- [Open the Database tool window](#) and click  on the toolbar.
- In the Data Sources and Drivers dialog that opens, click **+** and select DDL Data Source.
- In the Name field, if necessary, edit the name of the data source.
- Under DDL Files, click **+** and select the necessary SQL file or files in the dialog that opens.
- From the Extend list, if necessary, select another data source as a parent. As a result, the data source whose properties you are editing will "inherit" all the DDL definitions from its parent.
- Click OK to save the settings and close the dialog.

Creating a DDL data source by means of drag and drop

You can create DDL data sources by dragging DDL SQL files to the Database tool window. The files can be dragged from the Project tool window, or from your file manager (e.g. Explorer or Finder).

- If the necessary DDL SQL files are in your project folder, open the Project tool window. Otherwise, open your file manager.
- [Open the Database tool window](#).
- Select the file or files of interest in the Project tool window, or in your file manager.
- Drag the selected file or files into the Database tool window. For a new data source to be created, the red border, when dropping the file or files, should surround most of the window area (rather than one of the existing data sources).



Changing data source settings

- [Open the Database tool window](#) and select the data source of interest.
- Do one of the following:
 - Click  on the toolbar.
 - Select Properties from the context menu.
 - Press `Alt+Enter`.
- In the Data Sources and Drivers dialog that opens, edit the settings as necessary. See:
 - [DB data source settings](#)
 - [DDL data source settings](#)

Making a DB data source available in all your projects

When a DB data source is created, it's assigned to a project. That is, by default, it's available only in the project in which it was defined.

If you want to make a data source available in all your projects, you should make it global:

1. Open the Data Sources and Drivers dialog (e.g. `Alt+Enter`) and select the data source of interest.
2. Click  on the toolbar or select Make Global from the context menu.
3. Click Apply or OK.

In a similar way, you can move a global data source to the project level - to make it available only in the current project: use  or Move to Project from the context menu.

Note that the DDL data sources exist only on the project level.

Removing data sources

To remove unnecessary data sources, you can use the Database tool window or the Data Sources and Drivers dialog.

Using the Database tool window. Select the data sources to be removed and do one of the following:

- Press `Delete`.
- Select Delete from the context menu.
- Select Edit | Delete.

Using the Data Sources and Drivers dialog. Select the data sources to be removed and do one of the following:

- Click  on the toolbar.
- Press `Delete`.
- Select Remove from the context menu.

Working with the Database tool window

This feature is supported in the Professional edition only.

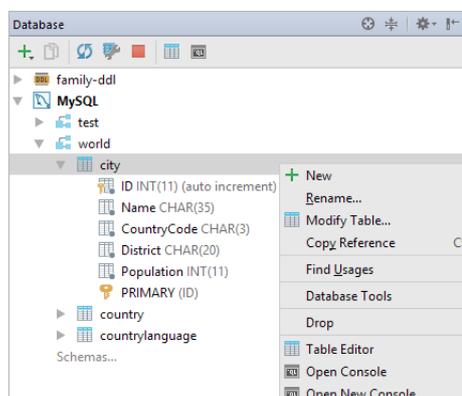
On this page:

- [Overview of the tool window](#)
- [Opening the Database tool window](#)
- [Creating a data source](#)
- [Synchronizing the view of a DB data source](#)
- [Resolving visualization problems](#)
- [Adjusting the view by means of view options](#)
- [Adjusting the view by means of object filters](#)
- [Showing and hiding schemas](#)
- [Finding items](#)
- [Finding usages of database objects](#)
- [Creating a copy of a data source](#)
- [Creating a database or schema](#)
- [Creating a table, a column, an index, or a primary or foreign key](#)
- [Modifying templates for generated index and key names](#)
- [Viewing basic info about an item](#)
- [Renaming items](#)
- [Previewing changes](#)
- [Modifying the definition of a table, column, index, or a primary or foreign key](#)
- [Opening DDL definitions of database objects in the editor](#)
- [Opening DDL definitions in a database console](#)
- [Generating DDL definitions on the clipboard](#)
- [Comparing table structures](#)
- [Viewing diagrams](#)
- [Copying a table to another database or schema](#)
- [Importing delimiter-separated values into a database](#)
- [Opening the table editor](#)
- [Copying data from one table to another one](#)
- [Saving data in files in various forms and formats](#)
- [Configuring data output formats and options](#)
- [Creating database backups with mysqldump or pg_dump](#)
- [Opening a default database console](#)
- [Creating and opening a new database console](#)
- [Generating Java entity classes for tables and views](#)
- [Closing database connections](#)
- [Removing items](#)

See also, [Database Tool Window](#).

Overview of the tool window

The Database tool window provides access to functions for working with databases and DDL data sources. It lets you view and modify data structures in your databases, and perform other associated tasks.



Opening the Database tool window

Do one of the following:

- Select [View | Tool Windows | Database](#).
- Point to  or  in lower-left corner of the workspace, and then click Database.
- Click Database on the right-hand [tool window bar](#) (if the tool window bars are currently shown).

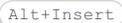
Creating a data source

To start creating a [data source](#), you can use the New command when the Database tool window is active, e.g.:

- File | New
-  on the toolbar
- New in the context menu
- 

The DDL Data Source option is for creating a DDL data source. Other data source options correspond to different scenarios of creating a DB data source:

- Data Source. A "usual way" of creating a data source. In this scenario, you start by selecting your DBMS.
- Data Source from URL. In this scenario, you start by specifying your database URL.
- Data Source from Path. In this scenario, you start by specifying your database location (a local file or folder). This option is appropriate only for Derby, H2, HSQLDB and SQLite.
- Import from sources. If you have files that contain database connection settings, you can create data sources by importing those settings. See [Creating DB data sources by importing connection settings](#).

You can also start creating a data source in the [Data Sources and Drivers dialog](#). Open the dialog (e.g. ) and use the Add command there: Add from the context menu,  on the toolbar, or .

For more information, see [Managing Data Sources](#).

Synchronizing the view of a DB data source

If the [Auto sync option](#) for a DB data source is off, the only way to synchronize its view in the Database tool window with the actual state of the database is by using the Synchronize command.

1. Select the item whose view you want to synchronize. This may be a DB data source, schema or table.
2. Do one of the following:
 - Press .
 - Click  on the toolbar.
 - Select Synchronize from the context menu.

Resolving visualization problems

If what you see in the Database tool window is somewhat problematic (e.g. no data structures are shown, the objects below the schema level are missing, etc.), try the following to resolve the problem:

1. [Synchronize the view of your data source](#) (.
2. Make sure that at least one of the available schemas is selected for viewing: check the Schemas popup. See [Showing and hiding schemas](#).
3. Switch to using the JDBC-based introspector:  | Options, select the [Introspect using JDBC metadata](#) check box. Then synchronize the view.
4. Clear the PyCharm schema cache ([Database Tools | Forget Cached Schema](#) from the context menu for the data source) and then synchronize the view.

Adjusting the view by means of view options

You can adjust the view in the tool window by turning the corresponding view options on and off. To access those options, click  on the title bar.

For more information, see [View options](#).

Adjusting the view by means of object filters

You can limit the set of tables and other database objects shown in the Database tool window by specifying object filters. The object filter is set for each DB data source individually, in the Data Sources and Drivers dialog () on the Options tab. The object filter syntax is described underneath the Object filter field.

Filter examples

 `f.*` Only the objects whose names start with `f` will be shown.

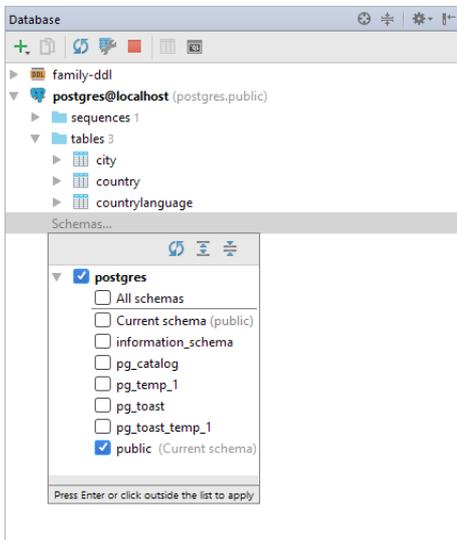
 `table:[gh].*` The tables whose names start with `g` or `h` and all the objects in other categories will be shown.

 `view:new_.*||routine:-[ps].*` The views whose names start with `new_`, the routines whose names start with the letters other than `p` or `s`, and all the objects in the categories other than views and routines will be shown.

Showing and hiding schemas

To show or hide schemas:

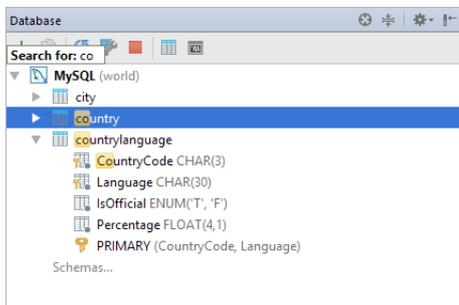
1. Within the DB data source of interest, double-click Schemas, or select Schemas and press . (Alternatively, right-click any element within the corresponding data source, point to Database Tools and select Manage Shown Schemas.)
2. Select the schemas you want to show and press .



To hide schemas, use the Schemas popup or the Database Tools | Hide Schemas context menu command.

Finding items

To find an item of interest, simply start typing its name. The specified text within item names is highlighted, and the first of the items that contains the specified text is selected.



Finding usages of database objects

You can search for usages of database objects in your files and consoles, and also in the source code of other objects (if loaded, see [Load sources for](#)). For example, you can look for references to a table or view in the code of other views, stored procedures and functions.

1. Select the item of interest.
2. Do one of the following:
 - Press **Alt+F7**.
 - Select Find Usages from the context menu.
 - Select Edit | Find | Find Usages in the main menu.

Creating a copy of a data source

1. Select the data source of interest.
2. Do one of the following:
 - Click  on the toolbar.
 - Select Duplicate from the context menu.
 - Press **Ctrl+D**.

Creating a database or schema

1. Select any element within the DB data source of interest.
2. Do one of the following:
 - Select File | New | Schema or File | New | Database.
 - Click  and select Schema or Database.
 - In the context menu, select New | Schema or New | Database.
 - Press **Alt+Insert** and select Schema or Database.
3. In the dialog that opens, specify the name of the schema or database. If necessary, under SQL Script, edit the statement to be executed. Click Execute.
4. If you have created a PostgreSQL database and want to see it in the Database tool window, [create a data source](#) for that database.

See also, [Track creation and deletion of databases/schemas](#).

Creating a table, a column, an index, or a primary or foreign key

1. Depending on what you are going to create:
 - To create a table, select a schema, table or column within the target DB data source.

- To create a column, select the target table or a column within that table.
 - To create an index, or a primary or foreign key, select the column or columns for which you want to create an index, or a primary or foreign key constraint.
2. Carry out the New command and select the item to be created. E.g. for a table, do one of the following:
 - Select File | New | Table.
 - Click **+** and select Table.
 - In the context menu, select New | Table.
 - Press **Alt+Insert** and select Table.
 3. In the [dialog that opens](#), specify the item definition.

Modifying templates for generated index and key names

When you [create indexes, and primary and foreign key constraints](#), their default names are generated according to corresponding templates. For a primary key, for example, the template is `{table}_{columns}_pk`.

You can view and modify these templates in the Settings / Preferences dialog: **Ctrl+Alt+S** | Editor | Code Style | SQL | Code Generation.

The templates can contain variables (e.g. `{table}`) and text. When generating a name, the specified text is reproduced literally.

To get the info about the variables and how you should use them, place the cursor into the field of interest and press **Ctrl+Q**.

`{columns}` and `{ref_columns}`, depending on the situation, are the name of the column, or a list where the column names are separated with the underscore (`_`).

`{unique?u:}` checks if the index is unique (`unique?`), and, if it is, inserts the sequence of characters specified between `?` and `:` (in this example, it's `u`). If the index is not unique, the sequence between `:` and `}` is inserted (in this example, it's nothing).

Example. Using the template `{table}_{columns}_{unique?u:}index`, you are creating an index on the columns `FirstName` and `LastName` in the table `persons`. If the index is unique, its name, by default, will be `persons_FirstName_LastName_u_index`. If the index is not unique, its name will be `persons_FirstName_LastName_index`.

Viewing basic info about an item

You can view basic info about an item in the quick documentation view. For a table, for example, the first ten rows and the table definition (the `CREATE TABLE` statement) are shown.

To open the quick documentation view, select the item of interest and do one of the following:

- Select View | Quick Documentation.
- Press **Ctrl+Q**.

See also, [Show first rows](#).

Renaming items

1. Select the item to be renamed.
2. Do one of the following:
 - Select Refactor | Rename.
 - Select Rename from the context menu.
 - Press **Shift+F6**.
3. Use the [dialog that opens](#) to specify a new name and associated options.

Previewing changes

Changes to database objects sometimes assume associated changes to SQL script files and statements in database consoles. For example, you may be changing the name of a table, and this name may be used in your files and consoles.

In such cases, you can look at potential changes, and decide where those changes are desirable and where aren't.

Potentially affected code fragments are shown in the [Find tool window](#) when you click Preview in the corresponding dialogs. Here is an overview of some of the available controls:

- Exclude (**Delete**) and Remove (**Alt+Delete**). Use these context menu commands for the items that shouldn't be changed.
- Execute SQL Script. If this option is on, and you click Do Refactor, the corresponding SQL statements are run to modify the corresponding database objects.
- Open in Console. Use this button to open the corresponding SQL statements in a [database console](#).
- Do Refactor. Click this button to change the corresponding code fragments and, if the Execute SQL Script option is on, to run the corresponding SQL statements.

Modifying the definition of a table, column, index, or a primary or foreign key

1. Select the item whose definition you want to change. This may be a table, a column, an index, or a primary or foreign key.
2. Do one of the following:
 - Select Modify <item_type> from the context menu (e.g. Modify Table).
 - Press **Ctrl+F6**.
3. Use the [dialog that opens](#) to change the item definition.

Opening DDL definitions of database objects in the editor

1. Select the object whose definition you want to view or edit.
2. Do one of the following:
 - Click  on the toolbar.
 - Select <ObjectType> Editor e.g. Routine Editor or Edit Source from the context menu.
 - Press **F4**.
3. If your object is a table or view, select the DDL tab in the lower part of the editor.

Opening DDL definitions in a database console

You can open DDL definitions of tables and views in [database consoles](#).

1. In a DB data source, select the table or view of interest.
2. Do one of the following:
 - Select Open DDL in Console from the context menu.
 - Press **Shift+F4**.

Generating DDL definitions on the clipboard

1. Select the item or items of interest. These may be data sources, schemas, tables, views, stored procedures or functions, etc.
2. Do one of the following:
 - Select Generate and Copy DDL from the context menu.
 - Press **Ctrl+Shift+C**.

Now you can paste the definitions into a [database console](#) or an SQL file.

Comparing table structures

1. Select two data sources, schemas or tables.
2. Do one of the following:
 - Select Compare from the context menu.
 - Press **Ctrl+D**.

The comparison results are shown in the [differences viewer](#).

Viewing diagrams

To open a diagram for a data source, schema or table, select the item of interest and do one of the following:

- Press **Ctrl+Shift+Alt+U** or **Ctrl+Alt+U**.
- In the context menu, select Diagrams, and then select Show Visualisation or Show Visualisation Popup.

Copying a table to another database or schema

You can copy (export) a table along with all its data to another schema or database. This is possible even when the source and target databases belong to different DBMSs, e.g. PostgreSQL and MySQL.

To copy a table:

1. Drag the table to the destination schema or database.
2. In the [dialog that opens](#), specify the settings for your new table.

Importing delimiter-separated values into a database

To import a text file containing delimiter-separated values (CSV, TSV, etc.) into your database, use drag-and-drop or the Import from File context menu command.

If you drag a file into a schema (or carry out the Import from File command for a schema), PyCharm will create a new table for the data that you are importing. If you drag a file into an existing table (or perform the command for a table), PyCharm will try to add the data to that table.

1. Do one of the following:
 - Drag a file from the Project tool window (the file may be a .zip archive) onto a schema or table in the Database tool window.
 - Right-click the target schema or table in the Database tool window, select Import from File and then select the file to import the data from (this file may be a .zip archive).
2. In the [dialog that opens](#), specify the data conversion settings, and, if a new table is to be created, the table name and structure.

Opening the table editor

1. Select the table of interest.
2. Do one of the following:
 - Click  on the toolbar.
 - Select Table Editor from the context menu.
 - Press **F4**.

For more information, see [Working with the Table Editor](#).

Copying data from one table to another one

1. Drag the source table to the destination table.
2. In the [dialog that opens](#), specify the data mapping info and other settings for the destination table.

Saving data in files in various forms and formats

You can save database data in files as SQL `INSERT` and `UPDATE` statements, [TSV and CSV](#), HTML tables and [JSON](#) data. A separate file is created for each individual table or view.

1. Select the data source or the schemas, tables and views of interest.
2. In the context menu, point to Dump Data to File(s) and select the output format (e.g. Comma Separated Values (CSV)).
3. In the dialog that opens, specify the destination directory and, if a single file is going to be created, the file name.

Configuring data output formats and options

To configure the output formats for the Dump Data to File(s) command (see [Saving data in files in various forms and formats](#)), select one of the following from the menu associated with the command:

- Configure CSV Formats. This command opens the [CSV Formats Dialog](#) that lets you manage your delimiter-separated values formats (e.g. CSV, TSV).
- Go to Scripts Directory. This command lets you switch to the directory where the scripts that convert table data into various output formats are stored.

For SQL INSERTs and UPDATEs, there are additional options: Add Table Definition and Skip Generated Columns. Those can be set in the table editor or the result pane of a database console. See e.g. [Specifying data output format and options](#).

Creating database backups with mysqldump or pg_dump

You can create backups for database objects by running [mysqldump](#) for MySQL or [pg_dump](#) for PostgreSQL.

1. Within a MySQL or PostgreSQL data source, select the items of interest (e.g. schemas, tables and views).
2. Select Dump Data with "mysqldump" or Dump Data with "pg_dump" from the context menu.
3. In the dialog that opens, specify the location of `mysqldump` or `pg_dump` executable, and the settings for performing the dump. If necessary, edit the command-line options in the lower part of the dialog (autocompletion is available).

Opening a default database console

1. Select the DB data source of interest or any node within it.
2. Do one of the following:
 - Click  on the toolbar.
 - Select Open Console from the context menu.
 - Press `Ctrl+Shift+F10`.

For more information, see [Working with Database Consoles](#).

Creating and opening a new database console

1. Select the DB data source of interest or any node within it.
2. Do one of the following:
 - Select Open New Console from the context menu.
 - Click `+` and select Console File.

For more information, see [Working with Database Consoles](#).

Generating Java entity classes for tables and views

1. Select the tables and views of interest.
2. In the context menu, point to Scripted Extensions and click Generate POJOs.cj or Generate POJOs.groovy.
3. In the dialog that opens, specify the directory in which you want to create your `.java` class files.

Closing database connections

PyCharm connects to databases automatically, when needed. (The names of the data sources with open database connections are shown in the Database tool window in bold.)

To close unnecessary database connections, select the corresponding data sources and do one of the following:

- Click  on the toolbar.
- Select Disconnect from the context menu.
- Press `Ctrl+F2`.

Removing items

Depending on what you are going to remove:

- Data source. Use the Remove command (Edit | Remove, Remove from the context menu, or `Delete` on the keyboard).
- Schema, table, column, index, a primary or foreign key constraint, stored procedure or function, etc. Use the Drop command (Edit | Drop, Drop from the context menu, or `Delete` on the keyboard).
- Primary or foreign key constraint. For removing primary and foreign key constraints, in addition to Drop, there are the following context menu commands: Database Tools | Drop Primary Key and Database Tools | Drop Foreign Key. Note that the Drop Foreign Key command is available only when a column with

the corresponding foreign key constraint is selected .

- All rows in a table. Use the Database Tools | Truncate context menu command for the corresponding table.

See also, [Confirm Drop Dialog](#).

Working with Database Consoles

This feature is supported in the Professional edition only.

Database consoles let you compose and execute SQL statements for databases defined in PyCharm as data sources. They also let you analyze and modify the retrieved data.

The following standardized and DBMS vendor-specific SQL dialects are supported: DB2, Derby, H2, HSQLDB, MySQL, Oracle, Oracle SQL*Plus, PostgreSQL, SQL Server, SQL92, SQLite, and Sybase .

One database console is created for a data source automatically when a data source is created. If necessary, you can create additional consoles.

Database consoles are persistent: they are stored as SQL files.

A database console created in one of your projects can then be accessed from any other project.

General console usage instructions are on this page:

- [Creating a database console](#)
- [Opening a database console](#)
- [Viewing and modifying console settings](#)
- [Changing the SQL dialect](#)
- [Closing a console](#)
- [Managing database consoles](#)

More specific subjects are discussed in:

- [Writing and Executing SQL Statements](#)
- [Working with Query Results](#)

See also, [Database Console](#).

Creating a database console

When you create a DB data source, one database console for that data source is created automatically. If necessary, you can create additional consoles.

To create a database console, you can use the [Database tool window](#) or the [Scratches view](#) of the [Project tool window](#). The procedure is the same in both cases:

Select the data source of interest or any node within it, and do one of the following:

- Select File | New | Console File from the main menu.
- Select New | Console File from the context menu
- Press `Alt+Insert` and select Console File.

In the Database tool window, you can also use `+ | Console File` and the Open New Console context menu command.

Opening a database console

The [Database tool window](#) lets you open only default consoles, i.e. the ones that were created by PyCharm automatically.

To open the consoles that you created yourself, you should use the [Scratches view](#) of the [Project tool window](#).

Using the Database tool window. Select the data source of interest or any node within and do one of the following:

- Click  on the title bar if the toolbar is hidden.
- Click  on the toolbar if the toolbar is shown.
- Select Open Console from the context menu.
- Press `Ctrl+Shift+F10`.

Using the Scratches view. For the console of interest, do one of the following:

- Double-click the console.
- Select View | Jump to Source from the main menu.
- Select Jump to Source from the context menu.
- Press `F4`.

Viewing and modifying console settings

Before actually starting to use a console, you may want to take a look at the console settings and adjust them to your needs.

- To access these settings, click  on the toolbar of the input pane or on the toolbar of the Database Console tool window. (Alternatively, `Ctrl+Alt+S` | Tools | Database.)

As a result, the [Database page](#) of the Settings / Preferences dialog will open.

Changing the SQL dialect

By default, the SQL dialect used in a console is defined by the DBMS of an associated data source. If for some reason you want to use a different dialect:

- Right-click the editing area of the input pane, select Change Dialect (<CurrentDialect>), and select the necessary dialect.

In addition to particular dialects, also the following option is available:

- <Generic SQL>. Basic SQL92-based support is provided including completion and highlighting for SQL keywords, and table and column names. Syntax error highlighting is not available. So all the statements in the input pane are always shown as syntactically correct.

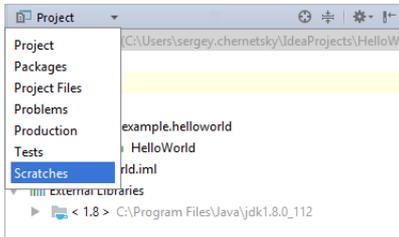
Closing a console

1. Click **X** (**Ctrl+Shift+F4**) to close the Database Console tool window.
2. Click **x** on the editor tab (**Ctrl+F4**) to close the input pane.

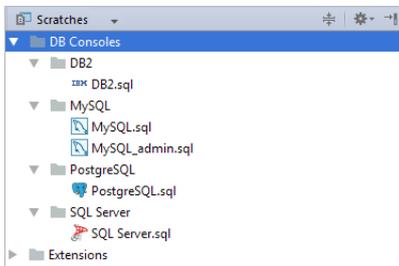
Managing database consoles

To manage your database consoles, use the Scratches view of the [Project tool window](#).

To open this view, select Scratches from the list in the left-hand part of the title bar.



The view shows the existing database consoles (represented by SQL files) grouped by your data sources (shown as folders). The default consoles (the ones that were created by PyCharm automatically) have the same names as the corresponding data sources.



You can:

- Create new consoles. Select the target data source or a node within it, and do one of the following:
 - Select File | New | Console File from the main menu.
 - Select New | Console File from the context menu
 - Press **Alt+Insert** and select Console File.
 - Rename your consoles. When a new console is created, it has the name of the data source with a number at the end, e.g. `MySQL_1`. If you want to give a console a more descriptive name, select the console and do one of the following:
 - Select Refactor | Rename from the main or the context menu.
 - Press **Shift+F6**.
- Then, specify a new name in the dialog that opens.
- Save your console files in arbitrary directories. Select the console and then select Refactor | Copy (**F5**) or File | Save As (**N/A**). Specify the file name and location in the dialog that opens.
 - Group your consoles. This is done by creating folders and then dragging your consoles into those folders.
 - Open your consoles. Select the consoles of interest and do one of the following:
 - Select View | Jump to Source from the main menu.
 - Select Jump to Source from the context menu.
 - Press **F4**.
 - View the history of changes for your consoles. Select File | Local History | Show History from the main menu or Local History | Show History from the context menu.
 - Delete individual consoles and groups of consoles. Use Edit | Delete, Delete from the context menu or **Delete** on the keyboard.

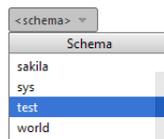
This feature is supported in the Professional edition only.

Use the input pane of the database console to compose and execute your SQL statements, and also to perform other, associated tasks. (The input pane is shown as a tab in the editor.)

- [Selecting the default schema or database](#)
- [Controlling the schema search path for PostgreSQL](#)
- [Composing SQL statements](#)
- [Editing data for INSERT statements in table format](#)
- [Navigating to a table or column view in the Database tool window](#)
- [Configuring the Execute command](#)
- [Executing an SQL statement](#)
- [Executing parameterized statements](#)
- [Executing a group of statements](#)
- [Executing all statements](#)
- [Executing a part of a statement \(e.g. a subquery\)](#)
- [Executing auto-memorized statements](#)
- [Outputting the result of a SELECT statement into a file](#)
- [Using the error notification bar](#)
- [Canceling running statements](#)
- [Managing database transactions](#)
- [Showing execution plans](#)
- [Showing DBMS_OUTPUT for Oracle](#)

Selecting the default schema or database

You can select the default schema or database by using the list in the right-hand part of the toolbar. If you do so, you'll be able to omit the name of that schema or database in your statements.



Note that you cannot switch schemas like this for read-only MySQL data sources, see [Read-only](#).

See also, [Controlling the schema search path for PostgreSQL](#).

Controlling the schema search path for PostgreSQL

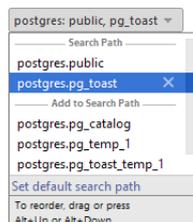
When working with a PostgreSQL data source, the default search path (one set in the database) is used unless you specify a different search path.

To control the search path, use the popup in the right-hand part of the toolbar.

If the search path should include only one schema, open the popup and click the necessary schema. In the same way you can replace a schema with another one in a single-schema search path, or restore the default search path (use the Set default search path link).

To form a search path that includes two or more schemas, open the popup and use:

- [Space](#) to add a highlighted schema to the search path and also to remove a schema from the search path.
- [Alt+Up](#) and [Alt+Down](#) to reorder the schemas within the search path.
- [Enter](#) to apply the changes, i.e. to set the search path.

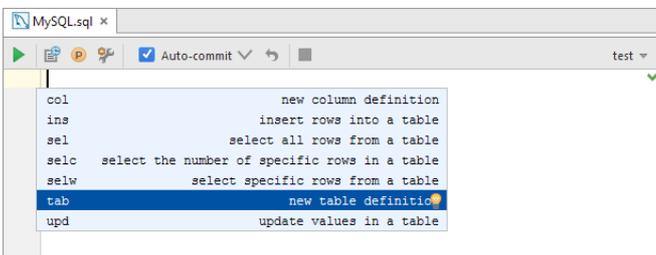


More instructions and usage hints are available right in the popup.

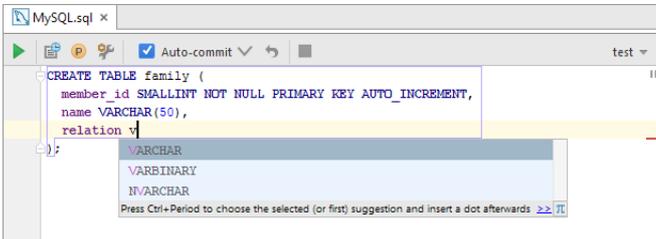
Composing SQL statements

When composing your SQL statements, use:

- Predefined patterns ([Ctrl+J](#) or Code | Insert Live Template).



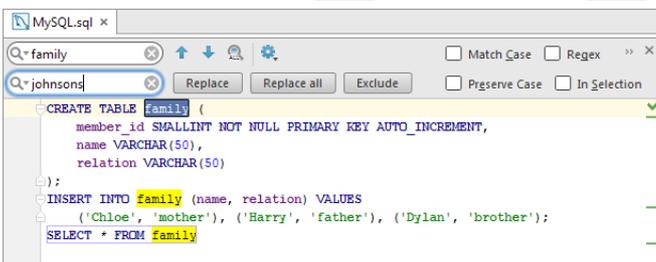
– Auto-completion and highlighting of SQL keywords, and table and column names.



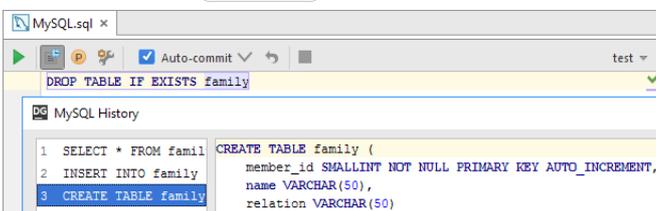
– Data type prompts for columns (`Ctrl+P` or View | Parameter Info).



– Advanced find and replace capabilities (`Ctrl+F` or Edit | Find | Find, and `Ctrl+R` or Edit | Find | Replace).



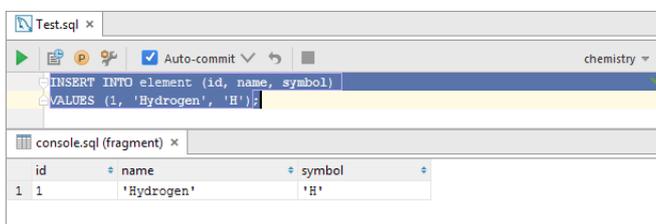
– The console history (`Ctrl+Alt+E`). See [Executing auto-memorized statements](#).



See also, [Navigating to a table or column view in the Database tool window](#).

Editing data for INSERT statements in table format

1. Select the `INSERT` statement of interest.
2. Select Edit as Table from the context menu. As a result, the table editor opens.



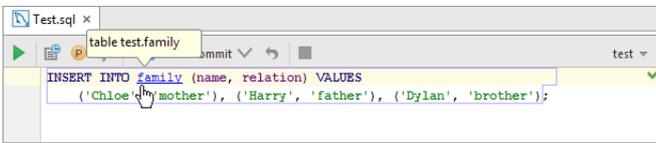
3. Use context menu commands and associated shortcuts for working with the data in the table editor.

Navigating to a table or column view in the Database tool window

When composing a statement, it's sometimes useful to take a look at the structure of a table, or to see the info about a column (field) in the context of the table to which it belongs. For such purposes, PyCharm provides the ability to switch from the name of a table or column in the input pane to its view in the Database tool window.

The following ways are available for using this feature:

- Place the cursor within the name of the table or column of interest. Then use `Ctrl+B`. (Alternatively, you can use Navigate | Declaration from the main menu or Go To | Declaration from the context menu.)
- Press and hold the `Ctrl` key, and point to the name of interest. When the text turns into a hyperlink, click the hyperlink.



Configuring the Execute command

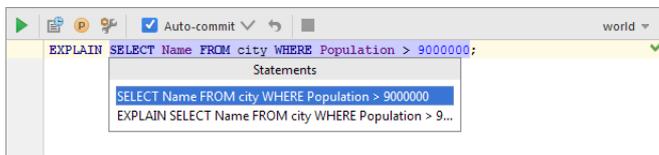
The Execute command (▶ on the toolbar, `Ctrl+Enter` or Execute from the context menu) is used to run your statements.

PyCharm provides many options for the Execute command depending on the cursor position and on whether there is a selection.

The options are specified on the Tools | Database page in the Settings / Preferences dialog (File | Settings | Tools | Database on Windows and Linux; PyCharm | Preferences | Tools | Database on macOS). For more information, see [Execute in Console](#).

Executing an SQL statement

1. Place the cursor within the statement.
2. Do one of the following:
 - Click ▶ on the toolbar.
 - Press `Ctrl+Enter`.
 - Select Execute from the context menu.
3. Select the statement or statements to be run. (The suggestion list always contains an item for running all the statements.)

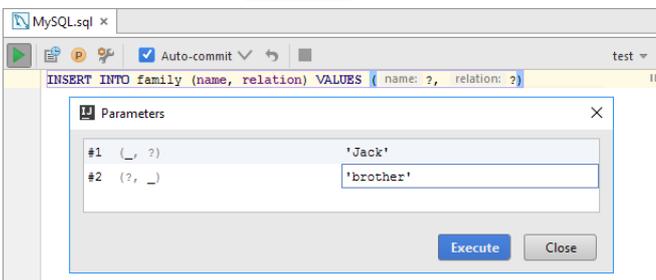


See also, [Execute in Console](#).

Executing parameterized statements

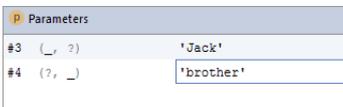
Your statements can contain parameters, however, by the time you execute such statements the values of the parameters must be specified. There are the following ways of specifying the parameter values:

- Click ▶ on the toolbar or press `Ctrl+Enter` to execute the statement. In the dialog that opens, specify the parameter values and click OK.



(To start editing a value, switch to the corresponding table cell and start typing. To indicate that you have finished editing a value, press `Enter` or switch to a different cell. To quit the editing mode and restore an initial value, press `Escape`.)

- Alternatively, you can open the Parameters pane in the Database Console tool window (Ⓟ on the toolbar) and specify the corresponding values there. (The values are edited in the same way as in the corresponding dialog.) Then execute the statement (▶ on the toolbar or `Ctrl+Enter`).



For more information, see [Parameters pane](#).

See also, [User Parameters](#) and [Always review parameters before execution](#).

Executing a group of statements

To execute a group of statements that follow one another in the console, select (highlight) the statements (to select all the statements, use `Ctrl+A`) and do one of the following:

- Click ▶ on the toolbar.
- Press `Ctrl+Enter`.
- Select Execute from the context menu.

See also, [Using the error notification bar](#) and [Execute in Console](#).

Executing all statements

To execute all the statements contained in a console, as an alternative to the Execute command, you can use the Run console.sql command.

This command is available in the context menu, and its keyboard equivalent is `Ctrl+Shift+F10`.

The Run console.sql command, generally, runs faster but:

- The statements with parameters don't run.
- Retrieved data for the `SELECT` statements are not shown.

Executing a part of a statement (e.g. a subquery)

To execute a part of a statement (e.g. a subquery), select (highlight) the fragment that you want to execute and do one of the following:

- Click  on the toolbar.
- Press `Ctrl+Enter`.
- Select Execute from the context menu.

See also, [Execute in Console](#).

Executing auto-memorized statements

As you run SQL statements in the consoles, PyCharm memorizes them. So, at a later time, you can view the statements you have already run and, if necessary, run them again.

To open the dialog where the auto-memorized statements are shown (the History dialog), do one of the following:

- Click  on the toolbar.
- Press `Ctrl+Alt+E`.

There are two panes in the History dialog. The left-hand pane shows the list of the statements that you have run. For "long" statements, only their beginnings are shown. When you select a statement in this pane, the overall statement is shown in the pane to the right.

You can filter the information: just start typing. As a result, only the statements that contain the typed text will be shown.

You can copy the statements from the History dialog into the input pane of the console. To copy a statement, do one of the following:

- Double-click the statement to be copied.
- Select the statement of interest and press `Enter`.
- Select the statement and click OK.

(Once the statement is in the input pane, you can run it straight away.)

You can delete unnecessary memorized statements. To delete a statement, select the statement in the History dialog and press `Delete`.

Outputting the result of a SELECT statement into a file

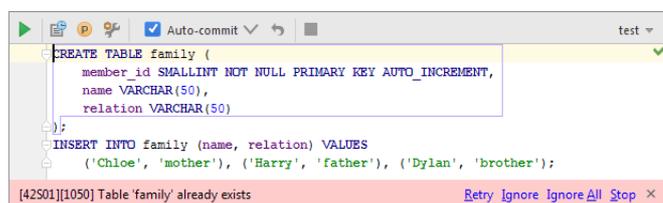
Instead of the Result pane of the Database Console tool window, you can output the result of a `SELECT` statement into a file.

1. Right-click the `SELECT` statement of interest.
2. Point to Execute to File and select the output format.
3. Specify the output file location and name.

Using the error notification bar

If when running a statement an error occurs, an error notification bar appears in the lower part of the input pane.

This bar may be particularly useful when executing a sequence of statements (see [Executing a group of statements](#)) because in such a case it lets you select how to react.



The options are:

- Retry. Execute the sequence of statements starting from the one that caused the error.
- Ignore. Skip the erroneous statement and execute the sequence starting from the next statement. If another error occurs, the error notification bar will appear again.
- Ignore All. Skip the erroneous statement and execute the sequence starting from the next statement. If other errors occur, all the erroneous statements will be skipped and the error notification bar won't appear for these statements.
- Stop. Stop the execution of the sequence.

Showing the error notification bar in the input pane is enabled or disabled in the Settings dialog (the [Show error notifications in editor](#) check box on the [Database page](#)).

Canceling running statements

To terminate execution of the current statement or statements, do one of the following:

- Click  on the toolbar of the input pane, or on the toolbar of the Database Console tool window.
- Press `Ctrl+F2`.

Managing database transactions

The Auto-commit check box, and the Commit  and Rollback  icons on the toolbar let you manage database transactions.

The Auto-commit check box is used to turn the autocommit mode for the database connection on or off.

In the autocommit mode, each SQL statement is executed in its own transaction that is implicitly committed. Consequently, the SQL statements executed in this mode cannot be rolled back.

If the autocommit mode is off, transactions are committed or rolled back explicitly by means of the Commit () or Rollback () command. Each commit or rollback starts a new transaction which provides grouping for a series of subsequent SQL statements.

In this case, the data manipulations in the transaction scope are committed or rolled back all at once when the transaction is committed or rolled back.

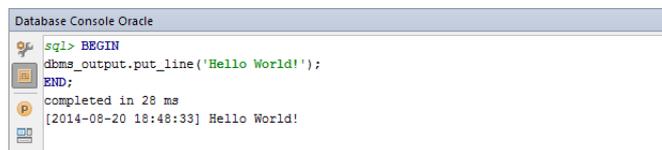
Showing execution plans

The following context menu commands let you show an [execution plan](#) (a.k.a. explain plan) for a statement:

- Explain Plan. The result is shown in a mixed tree/table format on a dedicated Plan tab.
- Explain Plan (Raw). The result is shown in table format. (Technically, `EXPLAIN <CURRENT_STATEMENT>` or similar statement is executed.)

Showing DBMS_OUTPUT for Oracle

For Oracle, you can enable or disable showing the contents of the DBMS_OUTPUT buffer in the output pane. To do that, use  on the toolbar of the Database Console tool window (`Ctrl+F8`).



```
Database Console Oracle
SQL> BEGIN
dbms_output.put_line('Hello World!');
END;
completed in 28 ms
[2014-08-20 18:48:33] Hello World!
```

Working with Query Results

This feature is supported in the Professional edition only.

When you run a query (a `SELECT` statement) in the console, the data retrieved from the database are shown in table format in the Result pane of the Database Console tool window. Depending on the settings, a new Result tab opens for each query, or one and the same tab is used. In the latter case, the results on the tab are updated for each next query.

Use the Result pane to sort, add, edit and remove the data as well as to perform other, associated tasks.

- [Hiding or showing the toolbar](#)
- [Pinning the Result tab](#)
- [Switching between subsets of rows](#)
- [Making all rows visible simultaneously](#)
- [Navigating to a specified row](#)
- [Navigating to related records](#)
- [Sorting data](#)
- [Reordering columns](#)
- [Hiding and showing columns](#)
- [Restoring the initial table view](#)
- [Using the Structure view to sort data, and hide and show columns](#)
- [Using the quick documentation view](#)
- [Transposing the table](#)
- [Enabling coding assistance for a column](#)
- [Selecting cells and ranges: using unobvious techniques](#)
- [Modifying cell contents](#)
- [Modifying values in a number of cells at once](#)
- [Adding a row](#)
- [Deleting rows](#)
- [Submitting and reverting changes](#)
- [Managing database transactions](#)
- [Comparing tables](#)
- [Copying table data to the clipboard or saving them in a file](#)
- [Copying and pasting data: data types are converted if necessary](#)
- [Specifying data output format and options](#)
- [Exporting the data to another table, schema or database](#)
- [Saving a LOB in a file](#)
- [Updating the table view](#)
- [Viewing the query](#)

Hiding or showing the toolbar

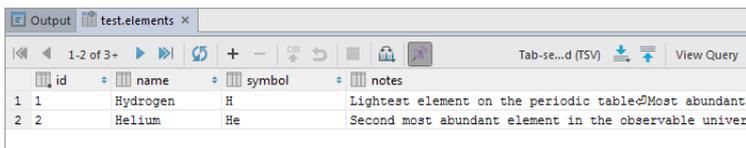
To hide or show the toolbar of the Result pane and also that of the Database Console tool window:

- Click  on the title bar of the Database Console tool window and click Show Toolbar.

Pinning the Result tab

If one and the same tab is used to show your query results, and you get the result that you want to keep, you can pin the tab to the tool window. Do one of the following:

- Right-click the tab and select Pin Tab.
- Click  on the toolbar.



	id	name	symbol	notes
1	1	Hydrogen	H	Lightest element on the periodic table; Most abundant
2	2	Helium	He	Second most abundant element in the observable universe

See also, [Show query results in new tab](#).

Switching between subsets of rows

If only a subset of the rows that satisfy the query is currently shown, to switch between the subsets, use:

-  First Page
-  Previous Page (`Ctrl+Alt+Up`)
-  Next Page (`Ctrl+Alt+Down`)
-  Last Page

See also, [Making all rows visible simultaneously](#).

Making all rows visible simultaneously

If you want all the rows that satisfy the query to be shown simultaneously:

1. Click  on the toolbar of the Database Console tool window.
2. Switch to the Database | Data Views page, specify `0` in the Result set page size field, and click OK.
3. Click  or press `Ctrl+F5` to refresh the table view.

See also, [Updating the table view](#) and [Result set page size](#).

Navigating to a specified row

To switch to a row with a specified number:

1. Do one of the following:
 - Press `Ctrl+G`.
 - Right-click the table and select Go To | Row from the context menu.
 - Select Navigate | Row from the main menu.
2. In the dialog that opens, specify the row number and click OK.

Navigating to related records

If a row references a record in a different table or is referenced in a different table, you can switch to the corresponding table to see the related record or records.

To switch to a referenced row:

1. Do one of the following:
 - Press `Ctrl+B`.
 - Select Go To | Referenced Data from the context menu.
2. If more than one record is referenced, select the target record in the pop-up that appears.

To switch to a row that references the current one, or to see all the rows that reference the current one:

1. Do one of the following:
 - Press `Alt+F7`.
 - Select Go To | Referencing Data from the context menu.
2. Select the target in one of the following categories:
 - First Referencing Row. All the rows in the corresponding table will be shown and the first of the rows that references the current row will be selected.
 - All Referencing Rows. Only the rows that reference the current row will be shown.

The options described above can also be accessed by using one of the following:

- `F4`.
- Go To | Related Data in the context menu.
- Navigate | Related Data in the main menu.

Sorting data

You can sort table data by any of the columns by clicking the cells in the header row.

Each cell in this row has a sorting marker in the right-hand part and, initially, a cell may look something like this: . The sorting marker in this case indicates that the data is not sorted by this column.

If you click the cell once, the data is sorted by the corresponding column in the ascending order. This is indicated by the sorting marker appearance:

. The number to the right of the marker (1 on the picture) is the sorting level. (You can sort by more than one column. In such cases, different columns will have different sorting levels.)

When you click the cell for the second time, the data is sorted in the descending order. Here is how the sorting marker indicates this order: .

Finally, when you click the cell for the third time, the initial state is resorted. That is, sorting by the corresponding column is canceled: .

See also, [Restoring the initial table view](#) and [Using the Structure view to sort data, and hide and show columns](#).

Reordering columns

To reorder columns, use drag-and-drop for the corresponding cells in the header row.

	 member_id	 relation	 name	 1
1	5	sister	lice	
2	1	mother	hloe	
3	3	brother	ylan	
4	4	sister	mily	
5	2	father	arry	

See also, [Restoring the initial table view](#).

Hiding and showing columns

To hide a column, right-click the corresponding header cell and select Hide column.

To show a hidden column:

1. Do one of the following:

– Right-click any of the cells in the header row and select Column List.

– Press `Ctrl+F12`.

In the list that appears, the names of hidden columns are shown struck through.

	member_id	name
1 5	test.family	Alice
2 1		Chloe
3 3	member_id int(11)	Dylan
4 4	name varchar(50) r1	Emily
5 2	relation varchar(50)	Harry
6 6		Jack

2. Select (highlight) the column name of interest and press `Space`.

3. Press `Enter` or `Escape` to close the list.

See also, [Restoring the initial table view](#) and [Using the Structure view to sort data, and hide and show columns](#).

Restoring the initial table view

Click on the toolbar and select Reset View to restore the initial table view after reordering or hiding the columns, or sorting the data. As a result, the data, generally, becomes unsorted, the columns appear in the order they are defined in the corresponding query, and all the columns are shown.

Using the Structure view to sort data, and hide and show columns

When working with the Result pane, the table structure view is available as the corresponding popup.

The structure view shows the list of all the columns and lets you sort the data as well as hide and show the columns.

To open the structure popup, do one of the following:

– Right-click a cell in the table header row and select Column List.

– Press `Ctrl+F12`.

In the popup, select the column of interest and do one of the following:

– To sort the data by this column in the ascending order, press `Shift+Alt+Up`.

– To sort the data in the descending order, press `Shift+Alt+Down`.

– To cancel sorting by this column, press `Ctrl+Shift+Alt+Backspace`.

– To hide the column (or show a hidden column), press `Space`. (The names of hidden columns are shown struck through.)

	member_id	name
1 5	test.family	Alice
2 1		Chloe
3 3	member_id int(11)	Dylan
4 4	name varchar(50) r1	Emily
5 2	relation varchar(50)	Harry
6 6		Jack

The shortcuts for sorting table data (`Shift+Alt+Up`, `Shift+Alt+Down` and `Ctrl+Shift+Alt+Backspace`) can be used in the Result pane without opening the structure view.

See also, [Sorting data, Hiding and showing columns](#) and [Restoring the initial table view](#).

Using the quick documentation view

The quick documentation view provides details about the values in the selected cell or cells. For example, if a cell contains long text, normally, you can see only its beginning. The whole text is shown in the quick documentation view.

Most abundant pure element on Earth	Most abundant pure element on Earth
Highly reactive nonmetallic element@T...	Highly reactive nonmetallic element
Lightest halogen@Extremely reactive@...	Third most abundant element in the unive
Colorless, odorless, inert monatomic g...	Colorless gas or pale blue liquid
Soft, silver-white, highly reactive me...	Lightest halogen
Alkaline earth metal@Shiny grey solid...	Extremely reactive
Silvery white soft metal	Highly toxic pale yellow diatomic gas
Crystalline, reflective with bluish-ti...	

If a cell contains an image, you can see that image in the quick documentation view.

relation	picture
brother	120x120 PNG image 24.95K
brother	120x120 PNG image 22.32K
cat	120x120 PNG image 25.95K
father	120x120 PNG image 17.65K
mother	120x120 PNG image 27.38K
sister	120x120 PNG image 27.78K



You can also see the records referenced in the current record as well as the records that reference the current one.

	first_name	last_name	address_id	email
1	MARY	SMITH	5	MARY.SMITH@sakilacustomer.org
2	PATRICIA	JOHNSON	6	PATRICIA.JOHNSON@sakilacustomer.org

Documentation for [1x2]

first_name	last_name
MARY	SMITH

Referenced address:
fk_customer_address(address_id)

address_id	address	address2	district	city_id	postal_code	phone
5	1913 Hanoi Way		Nagasaki	463	35200	2830338429

If necessary, you can switch to the transposed view. This is when the rows and columns are interchanged. Thus, for a row, the cells are shown one beneath the other.

	first_name	last_name	address_id	email
1	MARY	SMITH	5	MARY.SMITH@sakilacustomer.org
2	PATRICIA	JOHNSON	6	PATRICIA.JOHNSON@sakilacustomer.org

Documentation for [1x2]

Regular View

Column	1
first_name	MARY
last_name	SMITH
address_id	5
email	MARY.SMITH@sakilacustomer.org

To open the quick documentation view, press `Ctrl+Q` or select Quick Documentation from the View or the context menu.

To switch to the transposed view, click Transposed View. See also, [Transposing the table](#).

To close the quick documentation view, press `Escape`.

Transposing the table

The transposed table view is available. In this view, the rows and columns are interchanged.

To turn this view on or off, click  on the toolbar and select Transpose. Alternatively, use the Transpose context menu command.

Enabling coding assistance for a column

You can assign a column one of the supported languages (e.g. SQL, HTML or XML): right-click the corresponding header cell, select Edit As and select the language. As a result, you get coding assistance for the selected language in all the cells of the corresponding column.

You can also assign a language to an [individual cell](#).

Selecting cells and ranges: using unobvious techniques

Adding cells with the same contents. Select a cell. Now, to add the nearest cell with the same contents to the selection, press `Alt+J`. (When looking for the corresponding cell, PyCharm moves down.) Each next press of `Alt+J` will add another cell to the selection.

To remove the cells from the selection one by one - starting from the last selected cell - use `Shift+Alt+J`.

If a number of cells in the same row are initially selected, `Alt+J` and `Shift+Alt+J` work the same way.

Expanding a selection: cell - column - row - table. Select a cell. Now, to select all the cells in the current column, press `Ctrl+W`. The second press of `Ctrl+W` cancels the selection of the column and selects all the cells in the current row. Finally, the third press of `Ctrl+W` selects the whole table.

`Ctrl+W` works similarly if a number of cells or a range is initially selected.

Modifying cell contents

You can modify values in the table cells and, if appropriate, upload files.

1. To start editing a value or uploading a file, do one of the following:

- Double-click the corresponding table cell.
- Right-click the cell and select Edit or Edit Maximized from the context menu.
- Select the cell and press `F2` or `Shift+Enter`. In the latter case, the cell will be maximized.
- Select the cell and start typing. Note that in this case the initial cell contents are deleted right away and is replaced with the typed value.

2. When in the editing mode, you can:

- Modify the value right in the cell. To start a new line, use `Ctrl+Enter`. To enter the value, press `Enter`. To restore an initial value and quit the editing mode, press `Escape`.

Oxygen	O	Highly reactive nonmetallic element
Fluorine	F	Lightest halogen Extremely reactive Highly toxic pale yellow diatom
Neon	Ne	Colorless, odorless, inert monatomic g...

- Use value completion. Press `Ctrl+Space` to open the suggestion list. The list contains the values from the current column that match your input.

- Maximize the cell if you need more room for editing. To do that, press `Ctrl+Shift+Alt+M`, or right-click the cell and select Maximize. When working in a maximized cell, use `Enter` to start a new line and `Ctrl+Enter` to enter the value. To restore an initial value and quit the editing mode, press `Escape`.

```
Highly reactive nonmetallic elementThird most abundant elem...
Lightest halogen
Extremely reactive
Highly toxic pale yellow diatomic gas
```

- Upload a file into the field (e.g. to replace an existing file with a new one). To do that, right-click the cell and select Load File. Then select the necessary file in the dialog that opens. If a field can contain text, this function can be used to insert the contents of a text file into the field.
- Replace the current value with the default one or `null` (if appropriate). To do that, right-click the cell and select Set DEFAULT or Set NULL.
- Edit a value in the cell as a fragment in one of the supported languages (e.g. SQL, HTML or XML). To do that, right-click the cell, select Edit As and select the language. As a result, you get coding assistance for the language you have selected.

```
Highly reactive nonmetallic elementThird most abundant element
<ul>
  <li>Lightest halogen</li>
  <li>Extremely reactive</li>
  <li>Highly toxic pale yellow diatomic gas</li>
</ul>
  <li> http://www.w3.org/1999/html
  locator http://www.w3.org/1999/xlink
  simple http://www.w3.org/1999/xlink
```

3. To complete the task, you may want to submit the changes. See [Submitting and reverting changes](#).

Modifying values in a number of cells at once

You can modify a value in a number of cells at once:

1. Select the range or ranges of interest.
2. Start editing the value: select Edit from the context menu, press `F2` or simply start typing. The changes are applied to all the selected cells only if those cells can contain the same value.
3. To enter the value, press `Enter`. To cancel editing, press `Escape`.
4. To complete the task, you may want to submit the changes. See [Submitting and reverting changes](#).

Adding a row

If `+` on the toolbar is enabled, you can add rows to the table.

1. To start adding a row, do one of the following:
 - Click `+` on the toolbar.
 - Right-click the table and select Add New Row from the context menu.
 - Press `Alt+Insert`.

Note that the context menu Clone Row command (`Ctrl+D`) can be used as an alternative.
2. Enter the values into the cells. For instructions, see [Modifying cell contents](#).
3. To save the new row, select Submit from the context menu or press `Ctrl+Enter`. See also, [Submitting and reverting changes](#).

Deleting rows

If `-` on the toolbar is enabled, you can delete rows. To do that:

1. Select the row or rows that you want to delete. Rows are selected by clicking the cells in the column where the row numbers are shown. To select more than one row, use mouse clicks in combination with the `Ctrl` key.
2. Do one of the following:
 - Click `-` on the toolbar.
 - Press `Ctrl+Y` or `Delete`.
3. Submit the changes to the server or confirm you intention to delete the selected row or rows. See also, [Submitting and reverting changes](#).

Submitting and reverting changes

PyCharm lets you specify how the changes that you make to data in a table are submitted to the database server. There is the [Submit changes immediately option](#) for that.

By default, this option is off. So the changes are accumulated in PyCharm unless you carry out the Submit command `Submit` on the toolbar, Submit in the context menu or `Ctrl+Enter`. Before you submit the changes, you can revert them `Revert` on the toolbar, Revert in the context menu or `Ctrl+Z`.

The changes for a table are submitted all at once.

The scope of the Revert command is defined by the current selection in a table: the command is applied only to the changes within the selection. So you can revert an individual change, a group of changes or all the changes.

If nothing is currently selected, the Revert command is applied to the whole table.

Unsubmitted changes are highlighted. New rows are green, cells with changed values are blue, and the rows that are going to be deleted are gray.

If the Submit changes immediately option is on, the changes are submitted right-away, and, generally, you don't need to use the Submit command.

Managing database transactions

By default, the autocommit mode for a table is on. So each change of a value, or adding or deleting a row - [when submitted to the database server](#) - is implicitly committed and cannot be rolled back.

To turn the autocommit mode off, click  on the toolbar and then click Auto-commit.

When the autocommit mode is off, all the changes you have submitted to the server can be explicitly committed or rolled back by means of the Commit or the Rollback context menu command.

Comparing tables

You can compare the current table with any other table which is open in the table editor or shown in the Database Console tool window. To do that, click  on the toolbar and select the table of interest.

The comparison results are shown in the [differences viewer](#).

Copying table data to the clipboard or saving them in a file

When copying table data to the clipboard or saving them in a file, the data are converted into one of the available output formats. This can be SQL `INSERT` or `UPDATE` statements, [TSV or CSV](#), an HTML table or [JSON](#) data. See [Specifying data output format and options](#).

To copy or save the data, use:

- Copy (available in the Edit and the context menu, the keyboard equivalent is `Ctrl+C`). This command copies the data for the selected cells onto the clipboard.
- Dump Data | To Clipboard (available in the context menu and can also be accessed by means of  on the toolbar). This command copies the data for the whole table onto the clipboard.
- Dump Data | To File (available in the context menu and can also be accessed by means of  on the toolbar). This command saves the data for the whole table in a file. Before actually saving the data, the dialog is shown which lets you select the output format and see how your data will look in a file.

Copying and pasting data: data types are converted if necessary

You can copy (`Ctrl+C`) and paste (`Ctrl+V`) selected cells and ranges of cells - within the same table or from one table to another one. When pasting, PyCharm converts data types automatically if and as necessary.

Specifying data output format and options

To specify the output format and options for the Copy and Dump Data commands (see [Copying table data to the clipboard or saving them in a file](#)), do one of the following:

- Click  on the toolbar.
- Right-click the table and point to Data Extractor: <current_format>.

In the menu that opens, the output formats are in the upper part: SQL Inserts, SQL Updates, etc. (The options that look like file names are also the output formats or, to be more exact, the scripts that implement corresponding data converters.)

The output option are:

- Allow Transposition. This option affects only delimiter-separated values formats (TSV, CSV). If the table is shown transposed and you are copying selected cells or rows to the clipboard (e.g. `Ctrl+C`), the selection is copied transposed (as shown) if the option is on and non-transposed (as in the original table) otherwise.
- Skip Generated Columns (SQL). This is the option for SQL INSERTs and UPDATEs. When on, auto-increment fields are not included.
- Add Table Definition (SQL). This is also the option for SQL INSERTs and UPDATEs. When on, the table definition (CREATE TABLE) is added.

Additionally:

- Configure CSV Formats. This command opens the [CSV Formats Dialog](#) that lets you manage your delimiter-separated values formats (e.g. CSV, TSV).
- Go to Scripts Directory. This command lets you switch to the directory where the scripts that convert table data into various output formats are stored.

Exporting the data to another table, schema or database

You can export the data to another table, schema or database:

1. Do one of the following:
 - Click  on the toolbar.
 - Select Export to Database from the context menu.
2. Select the target schema (a new table will be created) or table (the data will be added to the selected table).
3. In the [dialog that opens](#), specify the data mapping info and the settings for the target table.

Saving a LOB in a file

If a cell contains a [binary large object](#) (a.k.a. BLOB or LOB), you can save such a LOB in a file.

1. Right-click the cell that contains the LOB of interest and select Save LOB To File.
2. In the dialog that opens, specify the name and location of the destination file and click OK.

Updating the table view

To refresh the table view, do one of the following:

- Click  on the toolbar.
- Right-click the table and select Reload Page from the context menu.
- Press `Ctrl+F5`.

Use this function to:

- Synchronize the data shown with the actual contents of the database.
- Apply the [Result set page size](#) setting after its change.

Viewing the query

To see the query that was used to generate the table:

- Click View Query on the toolbar.
If necessary, you can select the query text and copy it to the clipboard (`Ctrl+C`).

To close the pane where the query is shown, press `Escape`.

Working with the Table Editor

This feature is supported in the Professional edition only.

The Table Editor provides a GUI for working with table data. You can sort, filter, add, edit and remove the data as well as perform other, associated tasks.

- [Opening a table in the Table Editor](#)
- [Protecting a table from accidental modifications](#)
- [Switching between subsets of rows](#)
- [Making all rows visible simultaneously](#)
- [Navigating to a specified row](#)
- [Navigating to related records](#)
- [Sorting data](#)
- [Filtering data](#)
- [Using quick filtering options](#)
- [Reordering columns](#)
- [Hiding and showing columns](#)
- [Restoring the initial table view](#)
- [Using the Structure view to sort data, and hide and show columns](#)
- [Using the quick documentation view](#)
- [Transposing the table](#)
- [Enabling coding assistance for a column](#)
- [Selecting cells and ranges: using unobvious techniques](#)
- [Modifying cell contents](#)
- [Modifying values in a number of cells at once](#)
- [Adding a row](#)
- [Deleting rows](#)
- [Submitting and reverting changes](#)
- [Managing database transactions](#)
- [Comparing tables](#)
- [Copying table data to the clipboard or saving them in a file](#)
- [Copying and pasting data: data types are converted if necessary](#)
- [Specifying data output format and options](#)
- [Exporting the data to another table, schema or database](#)
- [Saving a LOB in a file](#)
- [Updating the table view](#)
- [Viewing the query](#)
- [Working with the CREATE TABLE statement](#)

Opening a table in the Table Editor

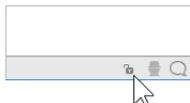
In the [Database tool window](#), do one of the following:

- Double-click the table of interest.
- Click the table and click  on the toolbar (if the toolbar is not currently hidden).
- Select the table and press .
- Right-click the table and select Table Editor from the context menu.

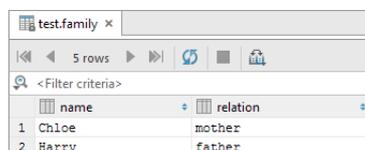
As a result, the table opens in the [Table Editor](#) on a separate editor tab.

Protecting a table from accidental modifications

To protect a table from accidental modifications in the Table Editor, you can make it read-only. To do that, click the padlock icon in the lower-right corner of PyCharm workspace.



As a result, the icon appearance will change to , a padlock will appear on the corresponding editor tab, and you won't be able to make changes to the table.

A screenshot of the Table Editor interface. At the top, there's a tab labeled 'testfamily x'. Below it is a toolbar with navigation and action icons. A search bar contains '<Filter criteria>'. The table has two columns: 'name' and 'relation'. The first row is 'Chloe' with 'mother' as the relation. The second row is 'Harry' with 'father' as the relation. The table is currently in read-only mode, indicated by a padlock icon in the top right corner of the table area.

	name	relation
1	Chloe	mother
2	Harry	father

To turn off the table's read-only status, click the padlock icon again.

Note that the tables with the read-only status in the Table Editor can still be modified when using the database console or in the Database tool window.

Switching between subsets of rows

If only a subset of all the rows is currently shown, to switch between the subsets, use:

-  First Page
-  Previous Page ()
-  Next Page ()
-  Last Page

See also, [Making all rows visible simultaneously](#).

Making all rows visible simultaneously

If you want all the rows to be shown simultaneously:

1. Click  on the toolbar and select Settings.
2. Switch to the Database | Data Views page, specify  in the Result set page size field, and click OK.
3. Click  or press  to refresh the table view.

See also, [Updating the table view](#) and [Result set page size](#).

Navigating to a specified row

To switch to a row with a specified number:

1. Do one of the following:
 - Press .
 - Right-click the table and select Go To | Row from the context menu.
 - Select Navigate | Row from the main menu.
2. In the dialog that opens, specify the row number and click OK.

Navigating to related records

If a row references a record in a different table or is referenced in a different table, you can switch to the corresponding table to see the related record or records.

To switch to a referenced row:

1. Do one of the following:
 - Press .
 - Select Go To | Referenced Data from the context menu.
2. If more than one record is referenced, select the target record in the pop-up that appears.

To switch to a row that references the current one, or to see all the rows that reference the current one:

1. Do one of the following:
 - Press .
 - Select Go To | Referencing Data from the context menu.
2. Select the target in one of the following categories:
 - First Referencing Row. All the rows in the corresponding table will be shown and the first of the rows that references the current row will be selected.
 - All Referencing Rows. Only the rows that reference the current row will be shown.

The options described above can also be accessed by using one of the following:

- .
- Go To | Related Data in the context menu.
- Navigate | Related Data in the main menu.

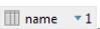
Sorting data

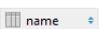
You can sort table data by any of the columns by clicking the cells in the header row.

Each cell in this row has a sorting marker in the right-hand part and, initially, a cell may look something like this: . The sorting marker in this case indicates that the data is not sorted by this column.

If you click the cell once, the data is sorted by the corresponding column in the ascending order. This is indicated by the sorting marker appearance:

. The number to the right of the marker (1 on the picture) is the sorting level. (You can sort by more than one column. In such cases, different columns will have different sorting levels.)

When you click the cell for the second time, the data is sorted in the descending order. Here is how the sorting marker indicates this order: .

Finally, when you click the cell for the third time, the initial state is resorted. That is, sorting by the corresponding column is canceled: .

You can turn on the [Sort via ORDER BY option](#), to enable sorting the data by the corresponding DBMS.

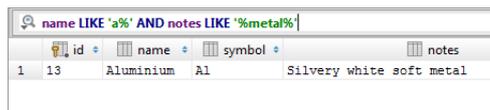
See also, [Restoring the initial table view](#) and [Using the Structure view to sort data, and hide and show columns](#).

Filtering data

1. If the filter box is not currently shown, click  on the toolbar and select Row Filter.

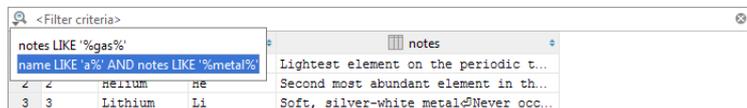
2. In the filter box, specify filtering conditions.

The filtering conditions are specified as in a `WHERE` clause but without the word `WHERE`, e. g. `name LIKE 'a%' AND notes LIKE '%metal%'`. Within the `LIKE` expressions, the SQL wildcards can be used: the percent sign (`%`) for zero or more characters and underscore (`_`) for a single character.



To apply the conditions currently specified in the box, press `Enter`. To cancel filtering, click , or delete the contents of the filter box and press `Enter`.

To reapply a memorized filter, click  and select the filter in the list. See also, [Filter history size](#).



Using quick filtering options

In addition to specifying filtering conditions manually (see [Filtering data](#)), you can use quick filtering options.

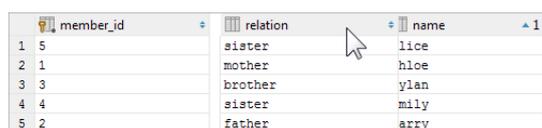
Available as context menu commands, these options are a set of filtering conditions for the current column name. The conditions themselves depend on the value in the current cell.

To use a quick filtering option:

1. Right-click a cell of interest and point to Filter by.
2. Select the necessary condition from the list.

Reordering columns

To reorder columns, use drag-and-drop for the corresponding cells in the header row.



See also, [Restoring the initial table view](#).

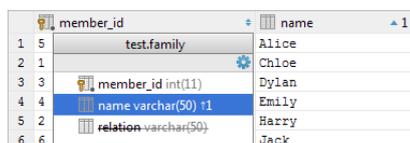
Hiding and showing columns

To hide a column, right-click the corresponding header cell and select Hide column.

To show a hidden column:

1. Do one of the following:
 - Right-click any of the cells in the header row and select Column List.
 - Press `Ctrl+F12`.

In the list that appears, the names of hidden columns are shown struck through.



2. Select (highlight) the column name of interest and press `Space`.
3. Press `Enter` or `Escape` to close the list.

See also, [Restoring the initial table view](#) and [Using the Structure view to sort data, and hide and show columns](#).

Restoring the initial table view

Click  on the toolbar and select Reset View to restore the initial table view after reordering or hiding the columns, or sorting the data. As a result, the data, generally, becomes unsorted, the columns appear in the order they are defined in the database, and all the columns are shown.

Using the Structure view to sort data, and hide and show columns

When working with the Table Editor, the table structure view is available in the Structure tool window or as the corresponding popup.

The structure view shows the list of all the columns and lets you sort the data as well as hide and show the columns.

To open the Structure tool window, do one of the following:

- Select View | Tool Windows | Structure in the main menu.
- Click Structure on the left-hand [tool window bar](#).
- Press `Alt+7`.

To open the structure popup, do one of the following:

- Right-click a cell in the table header row and select Column List.
- Press `Ctrl+F12`.

In the tool window or the popup, select the column of interest and do one of the following:

- To sort the data by this column in the ascending order, press `Shift+Alt+Up`. (In the tool window, you can, alternatively, select Sort | Ascending from the context menu.)
- To sort the data in the descending order, press `Shift+Alt+Down`. (In the tool window, alternatively, Sort | Descending.)
- To cancel sorting by this column, press `Ctrl+Shift+Alt+Backspace`. (In the tool window, alternatively, Sort | Unsorted.)
- To hide the column (or show a hidden column), press `Space`. (The names of hidden columns are shown struck through. In the tool window, alternatively, the Hide Column or Show Column context menu command can be used.)

	member_id	name
1	5	Alice
2	1	Chloe
3	3	Dylan
4	4	Emily
5	2	Harry
6	6	Jack

The shortcuts for sorting table data (`Shift+Alt+Up`), (`Shift+Alt+Down`) and (`Ctrl+Shift+Alt+Backspace`) can be used in the Table Editor without opening the structure view.

See also, [Sorting data](#), [Hiding and showing columns](#) and [Restoring the initial table view](#).

Using the quick documentation view

The quick documentation view provides details about the values in the selected cell or cells. For example, if a cell contains long text, normally, you can see only its beginning. The whole text is shown in the quick documentation view.

Most abundant pure element on Earth	Most abundant pure element on Earth
Highly reactive nonmetallic element@T...	Highly reactive nonmetallic element
Lightest halogen@Extremely reactive@...	Third most abundant element in the unive
Colorless, odorless, inert monatomic g...	Colorless gas or pale blue liquid
Soft, silver-white, highly reactive me...	
Alkaline earth metal@Shiny grey solid...	Lightest halogen
Silvery white soft metal	Extremely reactive
Crystalline, reflective with bluish-ti...	Highly toxic pale yellow diatomic gas

If a cell contains an image, you can see that image in the quick documentation view.

relation	picture
brother	120x120 PNG image 24.95K
brother	120x120 PNG image 22.32K
cat	120x120 PNG image 25.95K
father	120x120 PNG image 17.65K
mother	120x120 PNG image 27.38K
sister	120x120 PNG image 27.78K



You can also see the records referenced in the current record as well as the records that reference the current one.

first_name	last_name	address_id	email
MARY	SMITH	5	MARY.SMITH@sakilacustomer.org
PATRICIA	JOHNSON	6	PATRICIA.JOHNSON@sakilacustomer.org

Documentation for [ix2]

first_name	last_name
MARY	SMITH

Referenced address:

address_id	address	address2	district	city_id	postal_code	phone
5	1913 Hanoi Way		Nagasaki	463	35200	2830338429

If necessary, you can switch to the transposed view. This is when the rows and columns are interchanged. Thus, for a row, the cells are shown one beneath the other.

first_name	last_name	address_id	email
MARY	SMITH	5	MARY.SMITH@sakilacustomer.org
PATRICIA	JOHNSON	6	PATRICIA.JOHNSON@sakilacustomer.org

Documentation for [ix2]

Column	1
first_name	MARY
last_name	SMITH
address_id	5
email	MARY.SMITH@sakilacustomer.org

To open the quick documentation view, press `Ctrl+Q` or select Quick Documentation from the View or the context menu.

To switch to the transposed view, click Transposed View. See also, [Transposing the table](#).

To close the quick documentation view, press `Escape`.

Transposing the table

The transposed table view is available. In this view, the rows and columns are interchanged.

To turn this view on or off, click  on the toolbar and select Transpose. Alternatively, use the Transpose context menu command.

Enabling coding assistance for a column

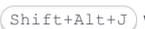
You can assign a column one of the supported languages (e.g. SQL, HTML or XML): right-click the corresponding header cell, select Edit As and select the language. As a result, you get coding assistance for the selected language in all the cells of the corresponding column.

You can also assign a language to an [individual cell](#).

Selecting cells and ranges: using unobvious techniques

Adding cells with the same contents. Select a cell. Now, to add the nearest cell with the same contents to the selection, press . (When looking for the corresponding cell, PyCharm moves down.) Each next press of  will add another cell to the selection.

To remove the cells from the selection one by one - starting from the last selected cell - use .

If a number of cells in the same row are initially selected,  and  work the same way.

Expanding a selection: cell - column - row - table. Select a cell. Now, to select all the cells in the current column, press . The second press of  cancels the selection of the column and selects all the cells in the current row. Finally, the third press of  selects the whole table.

 works similarly if a number of cells or a range is initially selected.

Modifying cell contents

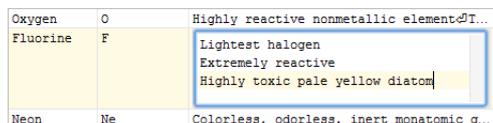
You can modify values in the table cells and, if appropriate, upload files.

1. To start editing a value or uploading a file, do one of the following:

- Double-click the corresponding table cell.
- Right-click the cell and select Edit or Edit Maximized from the context menu.
- Select the cell and press  or . In the latter case, the cell will be maximized.
- Select the cell and start typing. Note that in this case the initial cell contents are deleted right away and is replaced with the typed value.

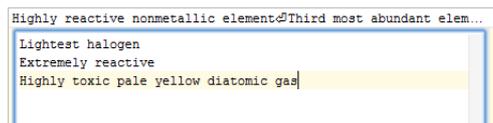
2. When in the editing mode, you can:

- Modify the value right in the cell. To start a new line, use . To enter the value, press . To restore an initial value and quit the editing mode, press .



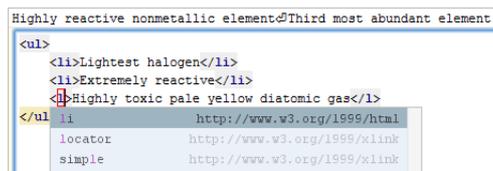
Oxygen	O	Highly reactive nonmetallic element@T...
Fluorine	F	Lightest halogen Extremely reactive Highly toxic pale yellow diatomic gas
Neon	Ne	Colorless, odorless, inert monatomic g...

- Use value completion. Press  to open the suggestion list. The list contains the values from the current column that match your input.
- Maximize the cell if you need more room for editing. To do that, press , or right-click the cell and select Maximize. When working in a maximized cell, use  to start a new line and  to enter the value. To restore an initial value and quit the editing mode, press .



Highly reactive nonmetallic element@Third most abundant elem...
Lightest halogen Extremely reactive Highly toxic pale yellow diatomic gas

- Upload a file into the field (e.g. to replace an existing file with a new one). To do that, right-click the cell and select Load File. Then select the necessary file in the dialog that opens. If a field can contain text, this function can be used to insert the contents of a text file into the field.
- Replace the current value with the default one or `null` (if appropriate). To do that, right-click the cell and select Set DEFAULT or Set NULL.
- Edit a value in the cell as a fragment in one of the supported languages (e.g. SQL, HTML or XML). To do that, right-click the cell, select Edit As and select the language. As a result, you get coding assistance for the language you have selected.



Highly reactive nonmetallic element@Third most abundant element
 Lightest halogen Extremely reactive Highly toxic pale yellow diatomic gas <ul li http://www.w3.org/1999/html locator http://www.w3.org/1999/xlink simple http://www.w3.org/1999/xlink

3. To complete the task, you may want to submit the changes. See [Submitting and reverting changes](#).

Modifying values in a number of cells at once

You can modify a value in a number of cells at once:

1. Select the range or ranges of interest.
2. Start editing the value: select Edit from the context menu, press  or simply start typing. The changes are applied to all the selected cells only if those cells can contain the same value.
3. To enter the value, press . To cancel editing, press .
4. To complete the task, you may want to submit the changes. See [Submitting and reverting changes](#).

Adding a row

If  on the toolbar is enabled, you can add rows to the table.

1. To start adding a row, do one of the following:

- Click  on the toolbar.
- Right-click the table and select Add New Row from the context menu.
- Press `Alt+Insert`.

Note that the context menu Clone Row command (`Ctrl+D`) can be used as an alternative.

2. Enter the values into the cells. For instructions, see [Modifying cell contents](#).

3. To save the new row, select Submit from the context menu or press `Ctrl+Enter`.

See also, [Submitting and reverting changes](#).

Deleting rows

If  on the toolbar is enabled, you can delete rows. To do that:

1. Select the row or rows that you want to delete.

Rows are selected by clicking the cells in the column where the row numbers are shown. To select more than one row, use mouse clicks in combination with the `Ctrl` key.

2. Do one of the following:

- Click  on the toolbar.
- Press `Ctrl+Y` or `Delete`.

3. Submit the changes to the server or confirm you intention to delete the selected row or rows.

See also, [Submitting and reverting changes](#).

Submitting and reverting changes

PyCharm lets you specify how the changes that you make to data in a table are submitted to the database server. There is the [Submit changes immediately option](#) for that.

By default, this option is off. So the changes are accumulated in PyCharm unless you carry out the Submit command  on the toolbar, Submit in the context menu or `Ctrl+Enter`. Before you submit the changes, you can revert them  on the toolbar, Revert in the context menu or `Ctrl+Z`.

The changes for a table are submitted all at once.

The scope of the Revert command is defined by the current selection in a table: the command is applied only to the changes within the selection. So you can revert an individual change, a group of changes or all the changes.

If nothing is currently selected, the Revert command is applied to the whole table.

Unsubmitted changes are highlighted. New rows are green, cells with changed values are blue, and the rows that are going to be deleted are gray.

If the Submit changes immediately option is on, the changes are submitted right-away, and, generally, you don't need to use the Submit command.

Managing database transactions

By default, the autocommit mode for a table is on. So each change of a value, or adding or deleting a row - [when submitted to the database server](#) - is implicitly committed and cannot be rolled back.

To turn the autocommit mode off, click  on the toolbar and then click Auto-commit.

When the autocommit mode is off, all the changes you have submitted to the server can be explicitly committed or rolled back by means of the Commit or the Rollback context menu command.

Comparing tables

You can compare the current table with any other table which is open in the table editor or shown in the Database Console tool window. To do that, click  on the toolbar and select the table of interest.

The comparison results are shown in the [differences viewer](#).

Copying table data to the clipboard or saving them in a file

When copying table data to the clipboard or saving them in a file, the data are converted into one of the available output formats. This can be SQL `INSERT` or `UPDATE` statements, [TSV or CSV](#), an HTML table or [JSON](#) data. See [Specifying data output format and options](#).

To copy or save the data, use:

- Copy (available in the Edit and the context menu, the keyboard equivalent is `Ctrl+C`). This command copies the data for the selected cells onto the clipboard.
- Dump Data | To Clipboard (available in the context menu and can also be accessed by means of  on the toolbar). This command copies the data for the whole table onto the clipboard.
- Dump Data | To File (available in the context menu and can also be accessed by means of  on the toolbar). This command saves the data for the whole table in a file. Before actually saving the data, the dialog is shown which lets you select the output format and see how your data will look in a file.

Copying and pasting data: data types are converted if necessary

You can copy (`Ctrl+C`) and paste (`Ctrl+V`) selected cells and ranges of cells - within the same table or from one table to another one. When pasting, PyCharm converts data types automatically if and as necessary.

Specifying data output format and options

To specify the output format and options for the Copy and Dump Data commands (see [Copying table data to the clipboard or saving them in a file](#)), do one of the following:

- Click `Tab S... (TSV)` on the toolbar.
- Right-click the table and point to Data Extractor: `<current_format>`.

In the menu that opens, the output formats are in the upper part: SQL Inserts, SQL Updates, etc. (The options that look like file names are also the output formats or, to be more exact, the scripts that implement corresponding data converters.)

The output options are:

- Allow Transposition. This option affects only delimiter-separated values formats (TSV, CSV). If the table is shown transposed and you are copying selected cells or rows to the clipboard (e.g. `Ctrl+C`), the selection is copied transposed (as shown) if the option is on and non-transposed (as in the original table) otherwise.
- Skip Generated Columns (SQL). This is the option for SQL INSERTs and UPDATEs. When on, auto-increment fields are not included.
- Add Table Definition (SQL). This is also the option for SQL INSERTs and UPDATEs. When on, the table definition (CREATE TABLE) is added.

Additionally:

- Configure CSV Formats. This command opens the [CSV Formats Dialog](#) that lets you manage your delimiter-separated values formats (e.g. CSV, TSV).
- Go to Scripts Directory. This command lets you switch to the directory where the scripts that convert table data into various output formats are stored.

Exporting the data to another table, schema or database

You can export the data to another table, schema or database:

1. Do one of the following:
 - Click  on the toolbar.
 - Select Export to Database from the context menu.
2. Select the target schema (a new table will be created) or table (the data will be added to the selected table).
3. In the [dialog that opens](#), specify the data mapping info and the settings for the target table.

Saving a LOB in a file

If a cell contains a [binary large object](#) (a.k.a. BLOB or LOB), you can save such a LOB in a file.

1. Right-click the cell that contains the LOB of interest and select Save LOB To File.
2. In the dialog that opens, specify the name and location of the destination file and click OK.

Updating the table view

To refresh the table view, do one of the following:

- Click  on the toolbar.
- Right-click the table and select Reload Page from the context menu.
- Press `Ctrl+F5`.

Use this function to:

- Synchronize the data shown with the actual contents of the database.
- Apply the [Result set page size](#) setting after its change.

Viewing the query

To see the query that was used to generate the table:

- Click View Query on the toolbar.
 - If necessary, you can select the query text and copy it to the clipboard (`Ctrl+C`).

To close the pane where the query is shown, press `Escape`.

Working with the CREATE TABLE statement

Click the DDL tab (in the lower-left part of the Table Editor) to see the `CREATE TABLE` statement used to create the table.

If necessary, you can edit this statement and then run it  on the toolbar or `Ctrl+Shift+F10`.

To regenerate the `CREATE TABLE` statement for the current state of the table in the database, use  or `Ctrl+F5`.

Running SQL Script Files

This feature is supported in the Professional edition only.

You can run an SQL file as a whole. You can also execute individual statements contained in an SQL file.

- [Running an SQL file](#)
- [Executing individual statements](#)

Running an SQL file

When running an SQL script file as a whole:

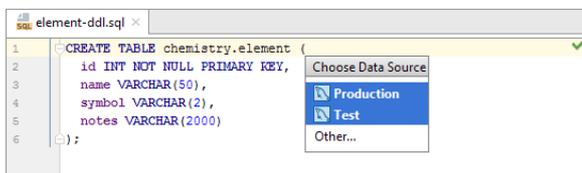
- You don't need to open the file in the editor. You can select the necessary file in the Project tool window.
- You can run the file for more than one data source at once.

On the other hand:

- The statements with parameters won't run.
- Retrieved data for the `SELECT` statements won't be shown.

To run an SQL file:

1. Select the necessary SQL file in the Project tool window, or open the file in the editor.
2. Do one of the following:
 - Select Run "<file_name>" from the context menu.
 - Press `Ctrl+Shift+F10`.
3. In the Choose Data Source pop-up, click the data source to which the script should be applied.
If you want to run the script for more than one data source, select the data sources of interest in the pop-up and press `Enter`.



Executing individual statements

When running individual statements contained in an SQL file:

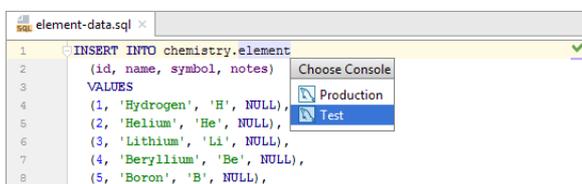
- The statements can contain parameters. Prior to running such statements PyCharm will ask you to specify the parameter values.

On the other hand:

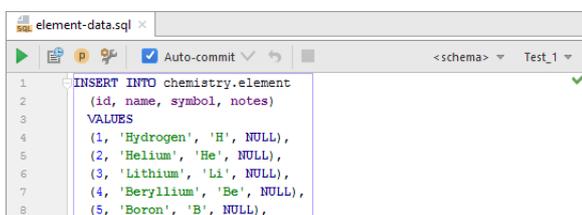
- The statements are run for only one data source at a time.

To run a statement or statements:

1. Open the SQL file of interest in the editor.
2. Place the cursor within the statement you want to execute.
If you want to run more than one statement, select (highlight) the necessary statements.
3. Do one of the following:
 - Press `Ctrl+Enter` or select Execute from the context menu.
 - Press `Alt+Enter` or click , and select Run query in console.
4. Select the [database console](#) to be used.



The statement or statements are executed using the selected console. The corresponding console is associated with the file. The name of the associated console is shown in the right-hand part of the toolbar.



The menu for the console name, lets you associate a different console with the file, or remove the association between the file and the console (<Detach Console>).

```
element-data.sql x
Auto-commit ✓
<schema> Test_1
1 INSERT INTO chemistry.element
2 (id, name, symbol, notes)
3 VALUES
4 (1, 'Hydrogen', 'H', NULL),
5 (2, 'Helium', 'He', NULL),
6 (3, 'Lithium', 'Li', NULL),
7 (4, 'Beryllium', 'Be', NULL),
8 (5, 'Boron', 'B', NULL),
```

- <Detach Console>
- Create New Console -
- Production
- Test

Running Injected SQL Statements

This feature is supported in the Professional edition only.

You can inject an SQL statement into a string literal (see [Using Language Injections](#)) and then run that statement:

1. In the editor, place the cursor within the corresponding string literal.
2. Do one of the following:
 - Press `Ctrl+Enter`.
 - Press `Alt+Enter` and select Run query in console.
 - Click  and select Run query in console.
3. If asked, select the [database console](#) to be used.
4. If the statement contains parameters, specify the parameter values.

Using language injections in SQL

This feature is supported in the Professional edition only.

For language injections in SQL, PyCharm provides the following additional features (for general info, see [Using Language Injections](#)):

- Auto-injection for XML and JSON data types, see [Using auto-injection for XML and JSON](#).
- Data type patterns, see [Using pattern-based injections for user-defined data types](#).

Using auto-injection for XML and JSON

For values defined as XML and JSON types, the corresponding languages are injected automatically.

Example

1. Create an SQL file and open it in the editor.
2. Specify PostgreSQL as an SQL dialect for that file.
3. Copy the following into your SQL file:

```
CREATE TABLE test (  
  my_xml XML DEFAULT ''  
);
```

4. Place the cursor between the quotation marks.
5. Check the light bulb menu (`Alt+Enter`): there is the Edit XML Fragment command there which means that XML has been auto-injected.

Using pattern-based injections for user-defined data types

You can create patterns - e.g. for user-defined data types - and associate those patterns with languages. As a result, PyCharm, when it comes across a data type that matches the pattern, will inject the language specified for that pattern.

In the following example, we'll create a pattern for a data type ending in `DATA` and associate that pattern with XML.

Example

1. In your SQL file, replace `XML` with `MYDATA`.
2. Place the cursor between the quotation marks.
3. Press `Alt+Enter`. Note that there is no more Edit XML Fragment command in the menu. Select Inject by Type, and then select XML (XML files).
4. In the dialog that opens, in the Type pattern field, specify `(?i).*DATA`. (The type patterns are specified using regular expressions. In this example, `(?i)` turns the case-insensitive mode on; `.*` stands for any number of any characters.)
5. Check the light bulb menu again (`Alt+Enter`). The Edit XML Fragment command has become available which means that XML has been injected for the value of the `MYDATA` type.
6. To remove the pattern you have just created (if you don't need it), open the Settings / Preferences dialog (e.g. `Ctrl+Alt+S`), go to the Editor | Language Injections page, find and delete the pattern.

Extending the functionality of database tools

You can extend the functionality of your database tools by writing scripts in Groovy, Clojure and JavaScript.

Example scripts

The PyCharm distribution includes example extension scripts which you can access using the Scratches view of the Project tool window.

The `Extensions/Database Tools and SQL/data/extractors` folder contains the scripts that convert table data into CSV, HTML, JSON, SQL INSERTs and XML formats (see e.g. [Specifying data output format and options](#)). The `Extensions/Database Tools and SQL/schema` folder contains the scripts that generate a Java entity class for a table (see [Generating Java entity classes for tables and views](#)).

Docker

This feature is supported in the Professional edition only.

Warning! The following is only valid when Docker Integration and Python Docker Plugins are installed and enabled!

- [Prerequisites](#)
- [Overview of Docker support](#)
- [Configuring PyCharm to work with Docker](#)
- [Creating a Docker Deployment run/debug configuration](#)
- [Working with Docker in PyCharm](#)

Prerequisites

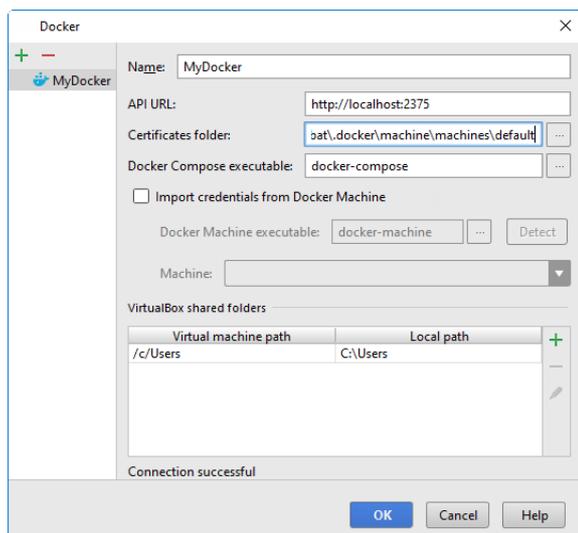
Make sure that the following prerequisites are met:

- Docker is installed, as described on the page [Docker Docs](#). You can install Docker on the various platforms:
 - [Windows](#)
 - [macOS](#)
 - [Linux](#) (Ubuntu, other distributions-related instructions are available as well)
- Before you start working with Docker, make sure that the Docker integration plugin is enabled. The plugin is bundled with PyCharm and is activated by default. If the plugin is not activated, enable it on the [Plugins](#) page of the [Settings / Preferences Dialog](#) as described in [Enabling and Disabling Plugins](#).
- Before you start working with Docker, make sure that the Python Docker plugin is enabled. The plugin is bundled with PyCharm and is activated by default. If the plugin is not activated, enable it on the [Plugins](#) page of the [Settings / Preferences Dialog](#) as described in [Enabling and Disabling Plugins](#).

Overview of Docker support

The Docker integration plugin adds the following to PyCharm:

- Docker configurations. These are named sets of settings for accessing the [Docker Engine](#) API and [Docker Compose](#).

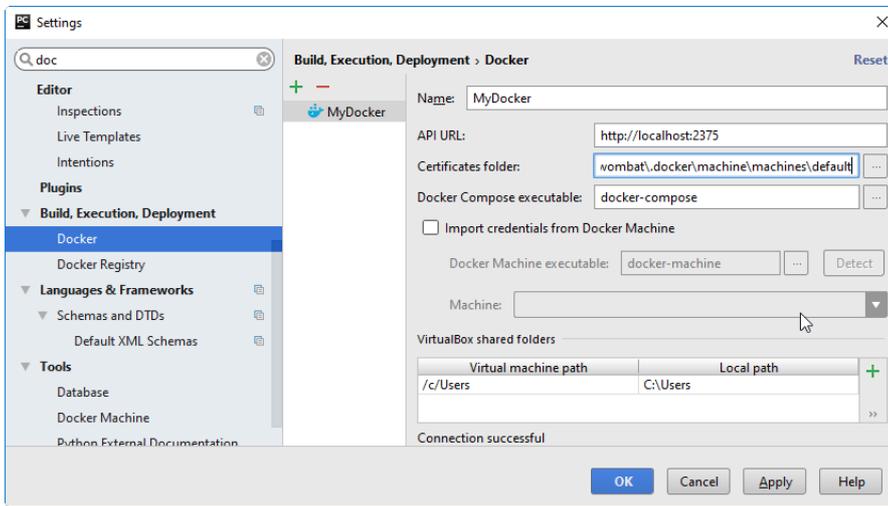


You can create a Docker configuration in two ways:

- Separately, in the Settings / Preferences dialog:
 - [Ctrl+Alt+S](#) | Build, Execution, Deployment | Docker |
 - [+ | Docker](#)
 - See [Docker configuration settings](#).
- When working with a [Docker Deployment](#) run/debug configuration.
- Create [Docker Deployment](#) run/debug configurations. They let you download and build Docker images, and create and start Docker containers. To create a Docker Deployment run/debug configuration, do the following: Run | Edit Configurations | [+ | Docker Deployment](#). See the [run configuration settings](#).
- [Docker tool window](#) (View | Tool Windows | Docker) that lets you manage your Docker images and containers.
- [Docker Registry configurations](#) that represent your Docker image repository user accounts.
- Additional setting in the [Python run/debug configuration](#).

Configuring PyCharm to work with Docker

1. In the Settings / Preferences dialog, open the [Docker](#) page under Build, Execution, Deployment, and click [+](#) to create a Docker configuration:



2. In the **Docker** page, provide the following information:

- Name. Here it is MyDocker.
- API URL. Here it is `http://localhost:2375`.
- Certificates folder. Here it is `/Users/<user name>/.docker/machine/machines/default`

3. Apply the changes and close the Settings / Preferences dialog.

After that, you can [configure a remote interpreter using Docker](#):

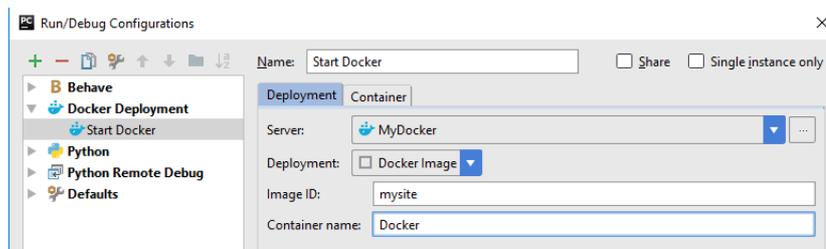


Creating a Docker Deployment run/debug configuration

Having set up Docker, [create the run/debug configuration](#). The Docker Deployment run/debug configuration can be used to download and build the Docker images, to create and start the Docker containers.

First, in the Deployment tab of the **Docker Deployment** run/debug configuration dialog, specify the following:

- Name. Here it is Start Docker.
- Server. Here Docker server is selected from the drop-down list.
- Deployment. Here Docker Image is selected.
- Image ID . Here it is mysite.
- Container. Here it is Docker .



Next, in the Container tab, specify the necessary parameters.

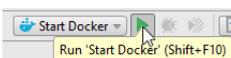
At this point we are interested in exposing 80 port of the container to be available from our local machine, so we should configure a port binding for that:

Container port: 80, Protocol: tcp, Host IP: empty, Host port: 8080

Apply changes and close the **Docker Deployment** run/debug configuration dialog.

Working with Docker in PyCharm

As all the tools are installed, and the integration is configured, the recently created Start Docker Run/Debug Configuration can be launched:



The **Docker** tool window opens, updating you on the provisioning status and the current state of all your Docker containers.



Docker Compose

This feature is supported in the Professional edition only.

- [Prerequisites](#)
- [Important note](#)
- [Working with Docker Compose](#)

Prerequisites

Make sure that the following prerequisites are met:

- You are working on Linux or macOS platform.
- Docker Compose is installed, as described on the page [Docker Compose](#).
- Before you start working with Docker Compose, make sure that the Docker integration plugin is enabled. The plugin is bundled with PyCharm and is activated by default. If the plugin is not activated, enable it on the [Plugins](#) page of the [Settings / Preferences Dialog](#) as described in [Enabling and Disabling Plugins](#).

Important note

A chosen service declared within a Docker Compose configuration file is supported as a Python interpreter. Python run configurations based on this interpreter operate like `docker-compose up` command with the addition that it maps project sources into the chosen service container.

The feature works for Linux with the local installations of Docker and macOS, it won't be shown in PyCharm running on Windows platform. On macOS, Docker Compose configuration file must be situated within the default shared folder `/Users/`.

Working with Docker Compose

In the Settings/Preferences dialog box, expand the node Build, Execution, Deployment, and in the [Docker](#) page, add the required Docker and then specify the Docker Compose executable.

As a result, the Docker Compose option appears in the Configure Remote Python Interpreter dialog box.

Tip PyCharm lets you specify multiple `docker-compose` configuration files and treats them same way as `docker-compose` does.

Documenting Source Code in PyCharm

In this part:

- Documenting Source Code in PyCharm
 - [Basics](#)
 - [Python documentation](#)
- [Enabling Creation of Documentation Comments](#)
- [Creating Documentation Comments](#)
- [Generating Reference Documentation](#)
- [Type Hinting in PyCharm](#)
- [Using Docstrings to Specify Types](#)

Basics

PyCharm provides convenient features for creating documentation comments.

Documentation comments in your source code are available for the [Quick Documentation Lookup](#) and open for review on pressing `Ctrl+Q`.

When you [create additional tags](#), PyCharm provides code completion that suggests the possible tag names.

Python documentation

Documentation comments can be created in accordance with the syntax, selected in the [Python Integrated Tools](#) page of the Settings/Preferences dialog, for example, [reStructuredText](#) or [epytext](#).

If this feature applies to a function, PyCharm generates tags, depending on the selected docstring format, for example:

- For reStructuredText: `:param` tags for each parameter declared in the function signature, and `:return` tag.
- For epytext: `@param` tags for each parameter declared in the function signature, and `@return` tag.

So doing, the tags in [reStructuredText](#) and [epytext](#) markup are highlighted accordingly.

If [configured](#), the documentation comment stubs can be generated with `type` and `rtype` tags.

In the Python files, PyCharm recognizes the documentation comments represented as [Python docstrings](#).

Before you start, make sure that the required docstring format, for example, `epytext` or `reStructuredText`, is selected in the [Python Integrated Tools](#) page of the Settings/Preferences dialog.

Also note that PyCharm captures custom roles from `conf.py`. When configuring the directory that contains `*.rst` files, point to the directory with `conf.py` ([Python Integrated Tools | Path to the directory with *.rst files](#)).

Enabling Creation of Documentation Comments

Warning! Note that this section refers to Python, JavaScript, and the other languages that have special beginning of documentation comments.

Enabling documentation comments

1. [Open Settings/Preferences dialog box](#), expand the Editor node, then expand General node, and click the [Smart Keys](#) page.
2. In the Enter section, select or clear Insert documentation comment stub check box.
3. Then, scroll to the Insert type placeholders in the documentation comment stub option and select or clear the check box as required. Refer to [the option description](#) for details.

Creating Documentation Comments

In this section:

- [Creating documentation comments for a method or function](#)
- [Creating tags](#)
- [Creating and fixing doc comments](#)
- [Creating documentation comments for Python functions](#)
 - [Example of Python comment](#)

Creating documentation comments for a method or function

Please note the following:

- PyCharm checks syntax in the documentation comments and treats it according to the Error settings.
- If the entered text contains HTML tags, the closing tag will be automatically added after typing `>`, provided that this behavior is [enabled](#) in the editor settings.
- When typing in a documentation comment, the caret automatically moves to an expected position. For example:

```
/**
 * <ul>
 *   <li>[caret]
 *   [caret after Enter]
 * </li>
 * </ul>
 */
```

Creating tags

To create tags in a documentation comment block

1. In a comment block, select the desired empty line and type `@` or `:` character.
2. Press `Ctrl+Space`, or just wait for Code Completion to display the suggestion list:

```
"""
:param param1:
:param param2:
:return:
:r
:raise
:raises
:return
:returns
:rtype
```

3. Select a tag from the suggestion list. For example, you can specify the parameters type, or return type.
4. If a certain tag has several values, press `Ctrl+Space` after the tag, and select the desired value from the suggestion list. For example, PyCharm suggests to select the desired parameter name.

```
class SimpleEquation:
) def demo(self, a, b, c):
) """
) :param
) a
) b
) c
) d = n
) root1
) root2
) math
) print
) print
) root1
) root2
SimpleEquation self
SimpleEquation
```

Creating and fixing doc comments

Warning! Note that this section refers to Python, JavaScript, and the other languages that have special beginning of documentation comments.

Documentation comment can be created with the dedicated action Fix Doc Comment. It can be invoked by means of [Find Action](#) command.

Press `Ctrl+Shift+A`, with the caret somewhere within a class, method, function, or field, which should be documented, and enter the action name Fix Doc String. The missing documentation stub with the corresponding tags is added. For example:

```
function loadDocs(myParam1, myParam2){}
```

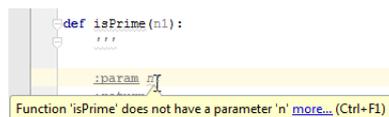
Type the opening documentation comment and press `Enter` to generate the documentation comment stub:

```
/**
 * @param myParam1
 * @param myParam2
 */
```

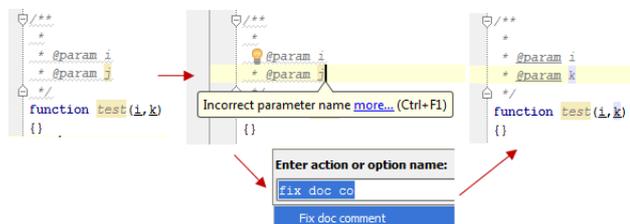
The next case lays with fixing problems in the existing documentation comments.

For example, if a method signature has been changed, PyCharm highlights a tag that doesn't match the method signature, and suggests a quick fix.

For Python, PyCharm suggests an inspection Ignore unresolved reference:



For JavaScript, PyCharm suggests an intention action UpdateJSDoc comment. You can also press `Ctrl+Shift+A`, and type the action name:



Tip The action Fix doc comment has no keyboard shortcut bound with it. You can [configure keyboard shortcut](#) of your own.

Creating documentation comments for Python functions

To create documentation comment for a Python function

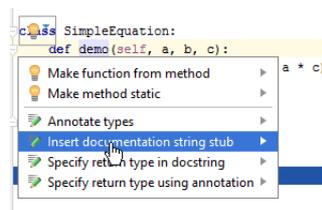
1. Place the caret **after** the declaration of a function you want to document.
2. Type opening triple quotes, and press `Enter`, or `Space`.
3. Add meaningful description of parameters and return values.

Tip Mind the following:

- Generation of docstrings on pressing `Space` after typing opening triple quotes only works when the check box Insert pair quote is cleared in the page [Smart Keys](#) of the editor settings.
- If you rename a parameter of a function, PyCharm will correspondingly update the tag in documentation comment.

To create documentation comment for a Python function using intention action

1. Place the caret somewhere within the function you want to document.
2. Press `Alt+Enter` to show the available intention actions.
3. Choose Insert documentation string stub:



PyCharm generates documentation comment stub according to docstring format, selected in the [Python Integrated Tools](#) page.

Example of Python comment

Consider the following function:

```
def handle(self, myParam1, myParam2):
```

In the [Python Integrated Tools](#) page, select Epytext. Then type the opening triple quotes and press `Enter` or `Space`. PyCharm generates documentation comment stub:

```
...
@param self:
@param myParam1:
@param myParam2:
@return:
...
```

Then select reStructuredText, type the opening triple quotes and press `Enter` or `Space`. PyCharm generates documentation comment stub:

```
...  
:param self:  
:param myParam1:  
:param myParam2:  
:return:  
...
```

Generating Reference Documentation

PyCharm helps produce the formatted API documentation, using the following documentation generators:

- [DocUtils](#)
- [Sphinx](#)

Note The documentation generators should be properly installed on your machine. Refer to their respective download and installation pages for details.
– PyCharm recognizes the docstring format and uses the documentation source directory defined in the [Python Integrated Tools](#) page of the Settings dialog.

Generating Reference Documentation Using DocUtils

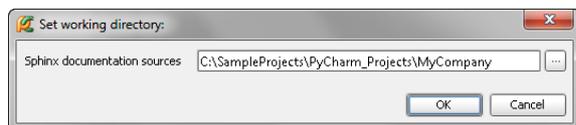
To generate docutils documentation

1. Select [DocUtil task](#) run/debug configuration, and change it as required: specify the configuration name, input and output directories, and optional keys.
2. Launch this run/debug configuration, as described in the section [Running Applications](#).

Generating Reference Documentation Using Sphinx

To create initial infrastructure for Sphinx documentation

1. On the main menu, choose Tools | Sphinx quickstart.
2. If the Sphinx working directory is not specified in the [Python Integrated Tools](#) page, the Set working directory dialog box opens, suggesting you to specify the path to the documentation.



Note If the Sphinx working directory is specified in your project, this dialog will not appear.

3. In the console that opens in PyCharm, answer the questions provided by the [sphinx-quickstart](#) utility. In particular, specify the source directory, where the generated `conf.py` file will be stored. If, answering this question, you just press `(Enter)`, PyCharm will use either the path you've specified in the previous step, or the path specified in the Sphinx working directory field of the [Python Integrated Tools](#) page.

Tip The `sphinx-quickstart` utility is performed only once for a particular directory. If you want to generate the Sphinx infrastructure in a different directory, specify it in the Sphinx working directory field of the [Python Integrated Tools](#) page.

To generate Sphinx documentation

1. Select [Sphinx task](#) run/debug configuration, and change it as required: specify the configuration name, input and output directories.
2. Launch this run/debug configuration, as described in the section [Running Applications](#).

Type Hinting in PyCharm

In this section:

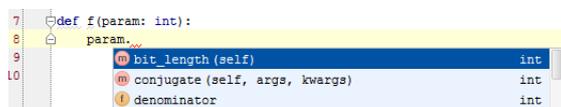
- [Following PEP484](#)
 - [Specifying types of parameters](#)
 - [Specifying return types](#)
 - [Specifying types of local variables and attributes](#)
 - [Python stubs](#)
- [Legacy type syntax for docstrings](#)
 - [Specifying types of local variables](#)
 - [Specifying types of fields](#)
 - [Specifying return types](#)
 - [Specifying parameter types](#)
- [Converting comments](#)

Following PEP484

PyCharm supports type hinting in function annotations and type comments using the `typing` module defined by [PEP 484](#).

Specifying types of parameters

When Python 3 is specified as the project interpreter, you can use annotations to specify the expected parameter type:



```
7 def f(param: int):
8     param
9     bit_length(self) int
10    conjugate(self, args, kwargs) int
11    denominator int
```

For Python 2, you can specify the types of parameters in the [Python stubs](#).

Specifying return types

When Python 3 is specified as the project interpreter, you can use annotations to specify the expected return type:



```
def f() -> int:
    pass
a = f()
a
bit_length(self) int
conjugate(self, args, kwargs) int
denominator int
```

For Python 2, you can specify return types in the [Python stubs](#).

Specifying types of local variables and attributes

Use annotations to specify the types of local variables and attributes:

```
class C:
    foo = None # type: List[str]

    def __init__(self, bar):
        self.bar = bar # type: Optional[str]

    def f2():
        return 'foo'

    def f1():
        x = f2() # type: str
        return x.upper()
```

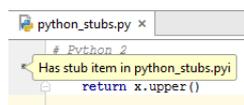
Also, you can specify the types of local variables and attributes in the [Python stubs](#).

Python stubs

PyCharm supports [Python stub files](#) with the `.pyi` extension. These files allow you to specify the type hints using Python 3 syntax for both Python 2 and 3.

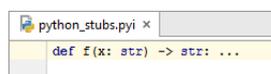
The stub files are [created as usual](#), but you must specify the extension `.pyi` explicitly.

PyCharm shows an asterisk in the left gutter for those Python files that have stubs:



```
python_stubs.py *
# Python 2
Has stub item in python_stubs.pyi
return x.upper()
```

Clicking the asterisk results in jumping to the corresponding file with the `.pyi` extension:



```
python_stubs.pyi *
def f(x: str) -> str: ...
```

Legacy type syntax for docstrings

PyCharm supports legacy approach to specifying types in Python using docstrings. So doing, the supported formats are:

`reStructuredText`

- [Documentation](#)
- [epytext](#)
- [NumPy](#)
- [Google](#)

To choose the desired docstring format, use the [Python Integrated Tools](#) page of the Settings/Preferences dialog.

Type syntax in Python docstrings is not defined by any standard. Thus, PyCharm suggests the following notation:

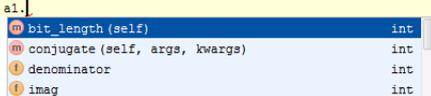
SyntaxDescription

Foo	Class Foo visible in the current scope
x.y.Bar	Class Bar from x.y module
Foo Bar	Foo or Bar
(Foo, Bar)	Tuple of Foo and Bar
list[Foo]	List of Foo elements
dict[Foo, Bar]	Dict from Foo to Bar
T	Generic type (T-Z are reserved for generics)
T <= Foo	Generic type with upper bound Foo
Foo[T]	Foo parameterized with T
(Foo, Bar) -> Baz	Function of Foo and Bar that returns Baz
list[dict[str, datetime]]	List of dicts from str to datetime (nested arguments)

Specifying types of local variables

Consider adding information about the expected type of a local variable using `:type` or `@type` docstrings:

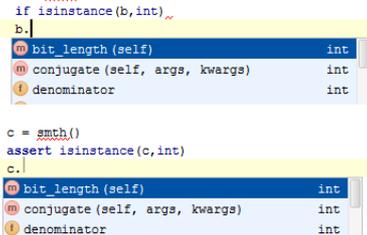
```
def f1():
    a1 = smth()
    """type : int"""
    a1.
```



It is also possible to use `isinstance` to define the expected local variable type:

```
b = smth()
if isinstance(b, int):
    b.
```

```
c = smth()
assert isinstance(c, int)
c.
```



Specifying types of fields

You can use type hinting to specify the expected type of fields:

```
class C():
    def f(self):
        self.foo = deserialize()
        """type : int"""
        self.foo.
```



Alternatively, you can specify types of fields in the docstring of a class:

```
class Foo(object):
    """
    :type a: list[str]
    :type b: int | None
    """
    def __init__(self):
        self.a = []
        self.b = None
```

Specifying return types

Use docstrings `:rtype` or `@rtype` to specify the expected return type:

```

7 def f():
8     """
9     :rtype: int
10    """
11    a=f()
12    a.
13

```

- :rtype: collections.Iterable[int] # return type: 'items' is of type generator or collections.Iterable, 'a' is of type int, see the following code:

```

def my_iter():
    for i in range(10):
        yield i
    items = my_iter()
    for a in items:
        print a

```

- :rtype: list[int] for my_iter # return type: 'a' is of type int, see the following code:

```

def my_iter():
    for i in range(10):
        yield i
    for a in my_iter():
        print a

```

Specifying parameter types

Consider adding information about the expected parameter type. This information is specified using :type or @type docstrings, for example, :param "type_name" "param_name": "param_description".

```

1 def f(param):
2     """
3     @type param: int
4     """
5     param.
6

```

```

1 def f(param):
2     """
3     :type param: int
4     """
5     param.
6

```

Converting comments

For comment-based type hints, PyCharm suggests an intention action that allows you to convert comment-based type hint to a variable annotation. This intention has the name Convert to variable annotation, and works as follows:

Warning! This is available for Python 3.6!

BeforeAfter

```

from typing import List, Optional

xs = [] # type: List[Optional[str]]

```

```

from typing import List, Optional

xs: List[Optional[str]] = []

```

Using Docstrings to Specify Types

In this section:

- [Introduction](#)
- [Prerequisite](#)
- [Parameter type specification](#)
- [Examples](#)
 - [Manually](#)
 - [Important note about reStructuredText/Sphinx](#)
 - [With the aid of the debugger](#)

Introduction

You debug your code permanently, and now in course of debugging you can also collect type information and specify these types in docstrings.

PyCharm provides an intention action that makes it possible to collect type information at runtime, and define type specifications.

However, it is quite possible to specify the types of parameters manually, without the debugger.

Both cases are explored in the section [Examples](#).

Prerequisite

Make sure that the check box Insert type placeholders in the [Smart Keys](#) page of the editor settings is selected.

Note also, that reStructuredText is used for all the subsequent examples, but it is possible to use any of the supported formats of the documentation strings, whether it is plain text, Epytext, Google or NumPy. Refer to the description of the page [Python Integrated Tools](#) for details.

Parameter type specification

To specify the parameter types, follow these general steps

1. Place the caret at the function name, and press `Alt+Enter`.
2. In the list of intention actions that opens, choose Insert documentation string stub. PyCharm creates a documentation stub, according to the selected docstring format, with the type specification, collected during the debugger session.

Examples

Consider the following code:

```
import math

class SimpleEquation:
    def demo(self, a, b, c):
        d = math.sqrt(b ** 2 - 4 * a * c)
        root1 = (-b + d) / (2 * a)
        root2 = (-b - d) / (2 * a)
        print(root1, root2)

SimpleEquation().demo(3, 2, 1)
```

Let us suppose that reStructuredText has been selected as the docstring format on the page [Python Integrated Tools](#).

Manually

Place the caret at the name of the function (here it is `demo`). The suggested intention action is Insert documentation string stub (refer to the section [Creating Documentation Comments](#) for details). Click this intention to produce the documentation comment stub in `reStructuredText` format:

```
import math

class SimpleEquation:
    def demo(self, a, b, c):
        """
        :param a:
        :type a:
        :param b:
        :type b:
        :param c:
        :type c:
        """
        d = math.sqrt(b ** 2 - 4 * a * c)
        root1 = (-b + d) / (2 * a)
        root2 = (-b - d) / (2 * a)
        print(root1, root2)

SimpleEquation().demo(3, 2, 1)
```

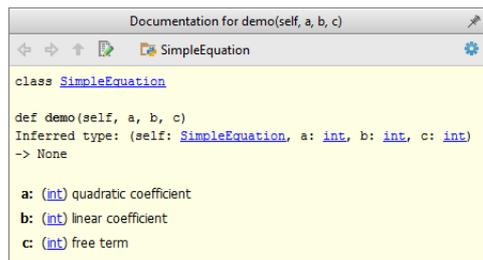
Then, manually specify the desired types of the parameters:

```
import math

class SimpleEquation:
    def demo(self, a, b, c):
        """
        :param a: quadratic coefficient
        :type a: int
        :param b: linear coefficient
        :type b: int
        :param c: free term
        :type c: int
        """
        d = math.sqrt(b ** 2 - 4 * a * c)
        root1 = (-b + d) / (2 * a)
        root2 = (-b - d) / (2 * a)
        print(root1, root2)

SimpleEquation().demo(3, 2, 1)
```

By the way, you can use [quick documentation](#) for the function in question. If you position the caret at the function name and press `Ctrl+Q`, you will see:



Important note about reStructuredText/Sphinx

Note that for the reStructuredText it's possible to specify types in [two formats](#):

- `:param param_type param_name: parameter description` (type description is on the same line as the parameter description).
- `:type param_name: param_type` (type description is on a separate line)

Both variants are shown below:

```
import math

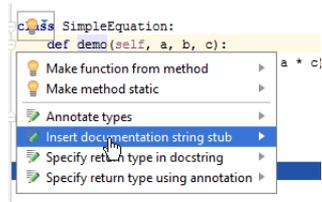
class SimpleEquation:
    def demo(self, a, b, c):
        """
        :param int a: the quadratic coefficient
        :param b: the linear coefficient
        :type b: int
        :param c: the free term
        """
        d = math.sqrt(b ** 2 - 4 * a * c)
        root1 = (-b + d) / (2 * a)
        root2 = (-b - d) / (2 * a)
        print(root1, root2)

SimpleEquation().demo(3, 2, 1)
```

With the aid of the debugger

Now, in the [Python Debugger](#) page of the Settings/Preferences dialog, select the check box `Collect run-time information for code insight`.

Debug the function call, and use intention action `Insert documentation string stub` again. Information about the arguments and return values obtained during the debugging session will be used to pre-populate type annotations in a docstring.



The obtained result is:

```
import math
```

```
class SimpleEquation:
```

```
    def demo(self, a, b, c):
```

```
        """
```

```
        :param a:
```

```
        :type a: int
```

```
        :param b:
```

```
        :type b: int
```

```
        :param c:
```

```
        :type c: int
```

```
        """
```

```
        d = math.sqrt(b ** 2 - 4 * a * c)
```

```
        root1 = (-b + d) / (2 * a)
```

```
        root2 = (-b - d) / (2 * a)
```

```
        print(root1, root2)
```

Flask

This feature is supported in the Professional edition only.

PyCharm supports [Flask](#) development.

In this section:

- Flask
 - [Flask Support](#)
- [Creating Flask Project](#)

Flask Support

Flask support in PyCharm includes:

- Dedicated [project type](#) .
- [Jinja2 templates](#) support.
- Live templates in the Flask group to create stubs of the Flask routes.
- [Navigation between views and templates](#).
- [Code completion](#) and resolve.

Creating Flask Project

This feature is supported in the Professional edition only.

Flask project is intended for productive development of the Flask applications. PyCharm takes care of creating the specific directory structure, and settings.

To create a Flask project, follow these steps

1. On the main menu, choose File | New | Project, or click the New Project button in the [Welcome screen](#). Create New Project dialog box opens.
2. In the [Create New Project](#) dialog box, specify the following:
 - Project name and location.
 - Project type Flask project.
 - In the Python Interpreter drop-down list, select the Python SDK you want to use. If the desired interpreter is not found in the list, click  and choose the interpreter type.
Refer to the section [Configuring Available Python Interpreters](#).
 - If Flask is missing in the selected interpreter, PyCharm displays an information message that Flask will be downloaded.
3. Click  (More Settings), and specify the following:
 - From the drop-down list, select the template language to be used.
 - The directory where the templates will be stored.
4. Click Create.

PyCharm creates an application and produces specific directory structure, which you can explore in the Project tool window. Besides that, PyCharm creates a stub Python script with the name `<project name>.py`, which provides a simple "Hello, World!" example.

Google App Engine

This feature is supported in the Professional edition only.

PyCharm supports all major Google App Engine development practices.

In this section:

- Google App Engine
 - [Prerequisites](#)
 - [Google App Engine support in PyCharm](#)
 - [Creating, deploying, and launching applications on Google App Engine](#)
- [Creating Google App Engine Project](#)
- [Uploading Application to Google App Engine](#)
- [Viewing Logs of a Google App Engine Application](#)
- [Running Tasks of 'appcfg.py' Utility](#)

Prerequisites

Prior to start working, consider the following prerequisites:

- Google App Engine SDK is downloaded and installed on your computer.
- Google App Engine works with Python versions 2.5 and higher.

Google App Engine support in PyCharm

Google App Engine support in PyCharm includes:

- A dedicated [project type](#) with specific directory structure and configuration file.
- Ability to [enable and configure](#) Google App Engine support per project.
- Ability to generate and view [model dependency diagram](#) for the Google App Engine models.
- Ability to [upload applications](#), using the command of the Tools menu.
- Run/debug configuration for [App Engine server](#).

Creating, deploying, and launching applications on Google App Engine

To create, deploy, and launch an application on Google App Engine, follow these general steps

1. In your browser, sign in to your Google account, register an application and get the application ID at <https://appengine.google.com/>.
2. In PyCharm, create a **Google App Engine** project with application id you obtained at <https://appengine.google.com/>. See [Creating Google App Engine Project](#).
3. Develop the desired contents.
4. **Upload** your application.
5. Visit `http://<your-application-name>.appspot.com/` to view your application in action.
6. If necessary, [check the logs](#).

Creating Google App Engine Project

This feature is supported in the Professional edition only.

A [Google App Engine](#) project is intended for productive development of web applications in the Google infrastructure. PyCharm takes care of creating the specific directory structure and settings.

To create a Google App Engine project, follow these steps

1. On the main menu, choose File | New | Project , or click New Project on the [Welcome screen](#).
2. In the [Create New Project](#) dialog box, specify the following:
 1. Project name and location.
 2. The Python interpreter to be used for the project. If the desired interpreter is not found in the list, click the browse button to review the available interpreters and virtual environments, and [configure the new ones](#).
 3. Project type Google App Engine project.

Click OK.

3. In the [App Engine Project](#) dialog box that opens, specify the following:
 - In the Application ID, enter the identifier that you have already defined on the [Create application](#) page of the Google App Engine. The application identifier should meet certain requirements: only lower-case letters, digits, and "-" characters are allowed; the application id should not begin with "-".
 - The location of the App Engine SDK on your machine. Type the path manually in the App Engine SDK directory field. If the SDK directory is added to the `path` variable during installation, it is detected automatically.
 - In the Templates Directory field, specify the directory where the templates will be stored, and where they will be loaded from. You can specify the name of the directory that doesn't yet exist; in this case, the directory will be created.
 - If you want to make Django available for your application, select the check box Enable Django support. With this option enabled, enter the Django project name and application name. Refer to the section [Creating Django Project](#) for details.
4. Click OK. PyCharm creates an application and produces specific file structure, which you can explore in the Project tool window.

Note that PyCharm always uses the Python 2.7 runtime when creating a new project.

Uploading Application to Google App Engine

This feature is supported in the Professional edition only.

To upload an application

- On the Tools menu, point to Google App Engine, and choose Upload App Engine Application.



Having uploaded the application, you can visit it at <http://<your-application-id>.appspot.com/>.

Viewing Logs of a Google App Engine Application

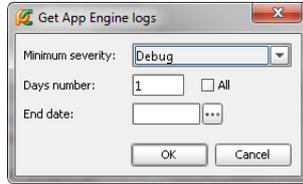
This feature is supported in the Professional edition only.

To download logs of a Google App Engine

1. On the Tools menu, choose Google App Engine, and choose Get App Engine logs.



2. In the Get App Engine logs dialog box, specify the severity of messages, and the desired period.



Running Tasks of 'appcfg.py' Utility

This feature is supported in the Professional edition only.

In this section:

- [Overview](#)
- [Running appcfg.py utility](#)
- [Working in the appcfg.py utility console](#)

Overview

With PyCharm, you can run appcfg.py utility from within the IDE. Each task of this utility is executed in the App Config Tool console.

Note that Run appcfg.py task... command is available for Google App Engine projects only.

Running appcfg.py utility

To run a task of the appcfg.py utility

1. On the main menu, choose Tools | Google App Engine | Run appcfg.py task...:



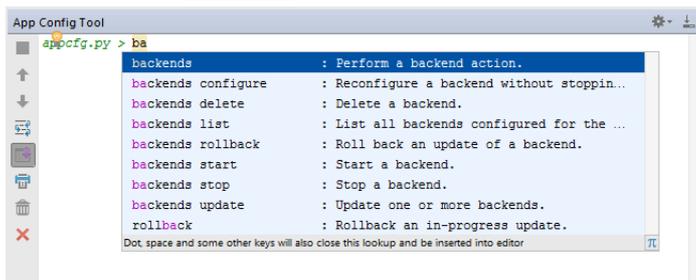
The `appcfg.py` utility starts in its own console.

2. Type the name of the desired task.

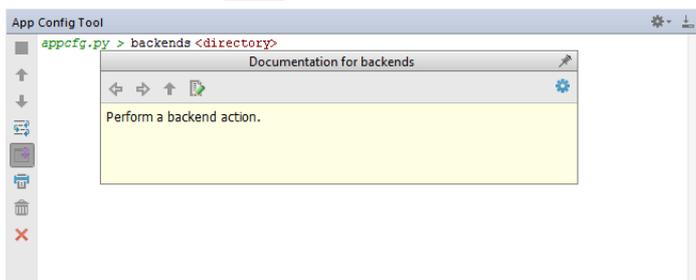
Working in the appcfg.py utility console

In the App Config Tool, one can:

- Scroll through the history of executed commands using the up and down arrow keys.
- Use [code completion](#) (`Ctrl+Space`):



- View [quick documentation](#) (`Ctrl+Q`):



IntelliLang

This is a third-party documentation.

The IntelliLang plugin offers a number of features related to the use of custom languages in PyCharm.

Prerequisites

Before you start working with language injections, make sure that the IntelliLang plugin is enabled. The plugin is bundled with PyCharm and is activated by default. If the plugin is not activated, enable it on the [Plugins](#) page of the [Settings / Preferences Dialog](#) as described in [Enabling and Disabling Plugins](#).

In this part:

- [Usage Examples](#)
- [IntelliLang Configuration](#)
- [Overview](#)
- [General Usage](#)
- [Quick Edit Language](#)
- [Code Inspections](#)

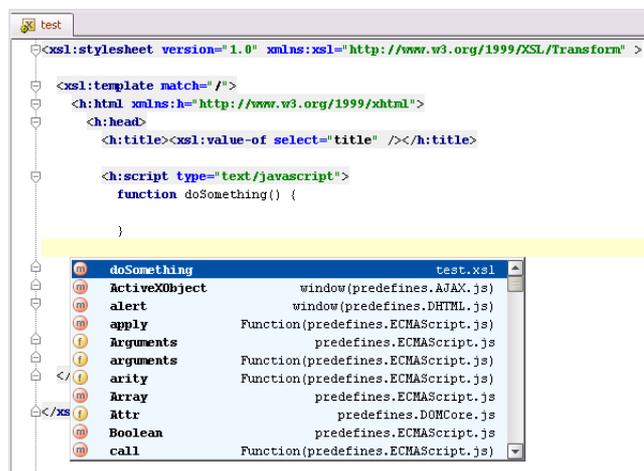
Usage Examples

Among the rather simple use-cases, like having detailed syntax checks for all kinds of "micro-languages" that are used e.g. in `Pattern.compile()`, `XPath.compile()` and so on, here are some less obvious, yet very useful examples how IntelliJLang can leverage PyCharm's support for a much better coding assistance.

The new scripting support in the upcoming Java 1.6 is another case where it will be important to get as much as possible edit-time assistance when script-code is constructed from Java code.

Extended JavaScript support

When dealing with JavaScript that's not directly embedded inside an HTML page, PyCharm usually just treats it as plain text. Consider the following example that creates an HTML page from an XSLT script. Without the JavaScript language being injected into the `script` tag with the XHTML namespace as shown in the screenshot below, this would be treated as plain text, with no further code assistance.



Support for JSP custom tags

With IntelliJLang it's also possible to make the content and attributes of custom JSP tags being treated as another language. This can be useful e.g. for server-side scripting using JavaScript or any other Language implementation available for PyCharm.

One thing that's important to know is that the taglib's URI which supplies a custom tag should be used as the namespace URI of the XML tag to inject a language into. The namespace-textfields contain a list of all known taglib URIs in the project.

Warning! Unfortunately, at the moment the support for refactoring and navigation inside JSP custom tags seems to be broken and attempting to use code completion may result in exceptions thrown in PyCharm core. See also the known issues below.

Pattern validation

Here's an obvious example right from PyCharm's OpenAPI:

```
/** com.intellij.codeInspection.LocalInspectionTool

 * @return descriptive name to be used in suppress comments and annotations,
 *         must consist of [a-zA-Z_0-9]+
 */
@NonNls @NotNull public String getID() {
    return getShortName();
}
```

The contract of the method `getID()` is that it should only return strings that match the pattern "[a-zA-Z_0-9]+". The short note in the JavaDoc can be easily overlooked though because the contract isn't specified in an automatically verifiable way.

However, if this method were annotated as `@Pattern("[a-zA-Z_0-9]+")`, any attempt to return a string that doesn't match that pattern would be flagged in the editor:

```
public abstract class CheckedLocalInspectionTool extends LocalInspectionTool {
    @NonNls @NotNull
    @Pattern("[a-zA-Z_0-9]+")
    public abstract String getID();
}

class MyLocalInspectionTool extends CheckedLocalInspectionTool {
    @NonNls @NotNull
    @Pattern("[a-zA-Z_0-9]+")
    public String getID() {
        return "an incorrect id";
    }
}
```

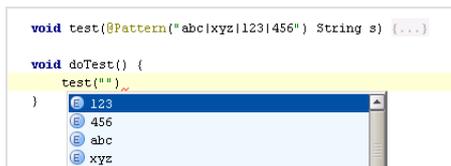
Expression 'an incorrect id' doesn't match pattern: [a-zA-Z_0-9]+

Pattern completion

If a regular expression pattern represents an enumeration of different literal values, the plugin offers completion for those values:

```
void test(@Pattern("abc|xyz|123|456") String s) {...}

void doTest() {
    test("123")
}
```

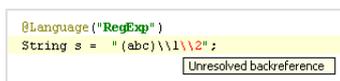


Editing regular expressions

Here are some examples of the enhanced coding support for regular expression patterns:

Backref validation

```
@Language("RegExp")
String s = "(abc)\\1\\2";
```



Surround with

```
@Language("RegExp")
String s = "ab";
```



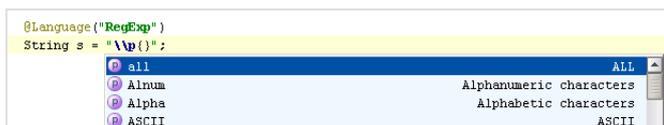
Character category validation

```
@Language("RegExp")
String s = "\\p{xxx}";
```



Character category completion

```
@Language("RegExp")
String s = "\\p{xxx}";
```



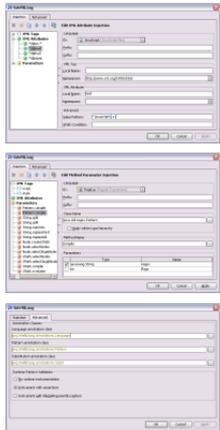
IntelliLang Configuration

The configuration dialog provides the following options:

- Language injection for XML text
- Language injection for XML attributes
- Language injection for method parameters
- Advanced settings that let you change the names of the recognized base-annotations and the way how the pattern-validation is performed during runtime.

Screenshots

The screen shots below show some of the configuration options that can be tweaked and will be described in the following sections.



Language Injection

This tab allows you to configure the language injection feature for XML text, attributes and method parameters. Use the buttons in the toolbar or the context menu actions to add, remove, copy or import new entries. To add a new entry, the appropriate group has to be selected first.

Reordering the entries is possible with the Move Up and Move Down buttons. This can be important to define the precedence of different XML-injection entries. If an entry matches, no more injections are applied unless the injection specifies a [value pattern](#).

XML Text

After adding a new XML text injection, select the ID of the Language to inject and optionally specify the prefix/suffix that make up the injection's context.

In the *XML Tag* pane, specify the local name (i.e. the name *without* any namespace prefix) and the namespace URI of the XML tag surrounding the text that should be treated as the selected language. The *name* field should not be empty, however the *namespace* field is optional.

The *Local Name* field takes a regular expression which makes it possible to specify e.g. multiple tag names (name1|name2), case-insensitive names (e.g. (?i)tagname matches *tagname* as well as *TagName*), etc. Be sure not to enter any whitespace characters as they would be significant for the match.

XML Attributes

This is similar to the XML text configuration. However, it is possible to leave the *name* of the *XML Tag* empty which means that the configuration will apply to any attribute that matches the configured name, regardless of its containing XML tag.

The attribute's name takes the local name of the attribute and is also specified as a regular expression. This can be e.g. used to match e.g. HTML event handler attributes by specifying the name "on.*". The attribute name may also be empty (unless the tag name is empty as well) which means that the configuration applies to all attributes of the containing tag.

Advanced XML Options

The *Advanced* pane for XML Text and Attributes allows an even more fine-grained control over the injection process.

Value Pattern

This field takes a regular expression that determines what part of the XML text's or attribute's value the language should be injected into. This can be used to inject the language only into values that match a certain pattern or to inject it into multiple parts that match the pattern. This is done by using the first capturing group of the pattern as the target for the injection.

Examples:

```
[$#]\{(.*)\}
```

This matches the pattern used by the JSP/JSF Expression Language.

```
^javascript:(.*)
```

This matches the *javascript*-protocol that can be used in hyperlink-hrefs to execute JS code.

XPath Condition

This field takes an XPath expression that can be used to address the injection-target more precisely than just by supplying its name and namespace URI. The context the expression is evaluated in is the surrounding XML tag for XML Text injection and the attribute itself for XML Attribute injection.

Example:

```
lower-case(@type)='text/javascript'
```

This limits the injection to tags whose type attribute contains the value "text/javascript".

It's possible to use the XPath [extension functions](#) that are provided by the [Jaxen](#) XPath engine, e.g. lower-case(). Also, there are three additional functions that can be used to determine the current file's name, extension and file type: file-name(), file-ext() and file-type(). You can also use normal code-completion to get a list of available functions.

Warning! For performance reasons, it's recommended to keep these expressions as simple as possible. Especially expressions that cause the whole document to be scanned, such as //foo/bar might cause performance problems with large files.

Method Parameters

This is a possibility to make use of IntelliLang's features, if, for any reason, the injection annotations cannot be used. This mainly applies to configuring 3rd party/library methods as well as projects that still have to use Java 1.4.

The language selection is identical to the one described above. To select one or more parameters of a certain method, first choose its containing class either by typing in the name (the textfield supports completion) or by using the class chooser available through the [...] button. Next, select a method using the [...] button inside the "Method-Name" pane (it's not possible to edit the method name manually). Once a method has been chosen, the table in the *Parameters* pane is populated with the parameters of the selected method. Select the check box in the first column to have the selected language injected into arguments passed for this parameter. Note that only parameters of type String can be selected.

There's already an IntelliJ IDEA-core (Inspection Gadgets rather) inspection that checks the arguments of some well-known methods to be a valid regular expression. However, IntelliLang can provide more detailed error messages and all the features of the Regular Expression Support plugin.

The XPath language that is configured by default for some of the standard XPath-APIs is provided by the XPathView + XSLT-Support plugin, which is available [here](#).

Importing Configuration Entries

To import the injection configuration from another PyCharm installation use the *Import* button from the toolbar and select the file "IntelliLang.xml" from /options or from a JAR file that contains some exported (via File | Export Settings) PyCharm settings. After selecting the file, a new dialog displays the entries contained in that file. Use the Delete-button to remove all entries you don't want to import. Note that this will *not* change the selected configuration file.

This selective import-feature makes it easy to share certain configurations in a team without losing any local entries like when importing settings via the core File | Import Settings action.

To prevent inconsistent data, the import is only possible if the existing configuration is unchanged or has been saved with the *Apply* button.

Advanced Settings

The advanced configuration tab allows you to specify different names for the base-annotations that should be used. This helps to avoid any dependencies on foreign code where this is not desired or possible. The custom annotations should just provide the same properties as the original ones, i.e. value for all of them and an optional (default = "") prefix and suffix for the @Language replacement.

Here it's also possible to configure the kind of runtime checks the plugin should generate for the @Pattern validation:

- No instrumentation: No checks will be inserted and doesn't touch any compiled class files
- Instrumentation using assert: Pattern-validation is controlled with the -ea JVM switch and throws AssertionError. This is the recommended way due to the potentially negative impact on the performance for methods that are invoked very often.
- Instrumentation using IllegalArgumentException- (for method parameters) and IllegalStateExceptions (for return values). This is the same the @NotNull instrumentation of PyCharm does.

Contributed Configurations

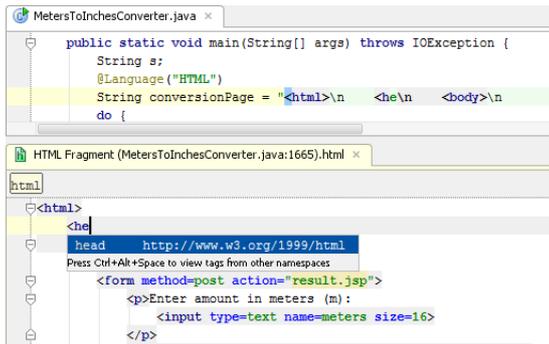
Additional configurations can be downloaded and imported from here. This community effect can help to minimize the configuration efforts and make IntelliLang even easier to use.

Overview

IntelliLang is a combination of three basic kinds of functionality that are meant to support the developer in dealing with certain tasks that relate to (custom) languages in PyCharm:

- Language Injection: Editing the code that is embedded into String literals and XML fragments natively.

You can open and modify an injected language code fragment in the [editor](#). In this way you get all the necessary coding assistance, as if you were working with the source code in the corresponding language.



To open an injected language code fragment in the editor, use the Edit <Language> Fragment [intention action](#).

- Pattern Validation: Provides assistance in making sure that Strings being passed to and from methods match a particular regular expression
- Regular Expression Support: A custom language implementation for regular expressions

Language injection

This makes use of the new possibilities of PyCharm to treat String literals, XML text and attributes as fragments of an arbitrary language (called *Language Injection*). The plugin makes this newly introduced API readily available to everybody for their daily use through two very simple means: Either by using some Java annotations to mark String fields, local variables, method parameters and methods returning Strings as containing a certain language, or by just using a simple UI configuration. There is a set of annotations provided by the plugin, but the actual annotations are freely configurable to avoid any unwanted dependencies.

This enables the developer to get the benefit of a wide range of edit-time features, such as syntax error highlighting, completion, inspections and a lot more while editing fragments of e.g. JavaScript inside regular Java code or in XML files of custom schemas that PyCharm usually doesn't know about.

Pattern Validation

Additionally, the plugin allows you to annotate Java elements of type String to have them checked for compliance with certain Regular Expressions. This can be useful for very simple *languages* where the developer needs to make sure that an expression conforms to a certain syntax, e.g. that a String is a legal Java identifier or a valid printf-like pattern used by `java.util.Formatter`.

This can both be validated on-the-fly while editing the code as well as during runtime (method parameters and return value only, like the `@NotNull` instrumentation of PyCharm core) by instrumenting the compiled classes with assertions that match the value against the supplied pattern.

Regular Expression Support

This part of the plugin implements language-support for `java.util.regex.Pattern` and has been mainly created to support the IntelliLang plugin by adding support for the micro-language that is probably one of the most often used one inside Strings. It features complete support for the syntax of the SDK's regular expression implementation and adds some further features, such as

- Completion and validation for character property names (e.g. `\p{javaJavaIdentifierStart}`) which nobody can usually remember anyway
- Validation and navigation for the use of back-references (e.g. `\1`), e.g. ctrl-b navigates to the capturing group the backref refers to.
- Intention Actions to simplify usages of repeated character occurrences, e.g. `a{0,1}` is offered to be converted to `a?`
- "Surround With" capturing/non-capturing group
- and more

General Usage

On this page:

- [Introduction](#)
- [Using the annotations](#)
 - [@Language](#)
 - [@Pattern](#)
 - [@Subst](#)
- [Supplying context: prefix and suffix](#)
- [Important notes](#)

Introduction

The plugin's usage is simple and straightforward, yet very flexible. Either add the provided set of annotations to your project and start using them, configure the plugin to use a custom set of annotations or simply use the UI configuration to make PyCharm learn that e.g. the String argument of `Pattern.compile()` should be treated as a regular expression.

Using the annotations

IntelliLang makes use of three base-annotations: `@Language`, `@Pattern` and `@Subst`.

- `@Language` is responsible for the Language injection feature.
- `@Pattern` is used to validate Strings against a certain regular expression pattern.
- `@Subst` is used to substitute non-compile time constant expressions with a fixed value. This allows you to validate patterns and build the prefix/suffix of an injected language (see below) even for non-constant expressions that are known to contain certain kinds of values during runtime.

The annotations supplied with IntelliLang are located in the file `annotations.jar` which can be found in `/plugins/IntelliLang/lib`.

@Language

The `@Language` annotation can be used to annotate String fields, local variables, method parameters and methods returning Strings. This will cause String-literals that are assigned to a field/variable, passed as a parameter or are used as a method's return value to be interpreted as the specified language.

Additionally, there's the Language Mismatch inspection which checks for clashes between the *expected language* and the *actual language* when a field/variable is assigned or a value returned from a method.

The plugin supports *direct* and *indirect* annotations, i.e. you can either directly use the annotation like this:

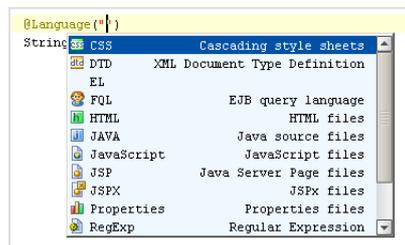
```
@Language("JavaScript")
String code = "var x = 1 + 2";
```

or annotate another annotation class like this:

```
@Language("XPath")
public @interface XPath { }
```

which can then simply be used to annotate elements as `@XPath`.

It's very easy to obtain the `language-id` that must be supplied as the annotation's value attribute: IntelliLang provides the list of available language via the regular code-completion action. Just select the appropriate language from the ctrl-space pop-up window:



@Pattern

The `@Pattern` annotation is responsible for marking Strings that have to comply with a certain regular expression and can be used in just the same way as the `@Language` annotation. That means, it's possible to create derived annotations, such as an `@Number` annotation that requires a String to consist of one or more digits:

```
@Pattern("\\d+")
public @interface Number { }
```

In fact, the predefined annotations already contain two of such derived annotations: The first one, `@PrintFormat`, matches the printf-like pattern used by `java.util.Formatter` and another one, `@Identifier`, describes a valid Java identifier. These are ready to use without having to specify any additional pattern.

@Subst

The `@Subst` annotation is used to substitute references that are not compile-time constant which enables the plugin to do Pattern Validation based on the

assumption that the substituted value is compatible with the values that are expected during runtime. The plugin will complain if the value does not match the expected pattern.

It also helps to build a valid context of prefix/suffix (see next section) for the Language Injection feature. Consider this example:

```
@Subst("Tahoma")
final String font = new JLabel().getFont().getName();

@Language("HTML")
String message = "<html><span style='font: " + font + "; font-size:smaller'> + ... + "</span></html>";
```

Without substituting the value of the variable `font` with the value `Tahoma` (actually it could just be a single character here), the injected fragment would be syntactically incorrect, causing the error *a term expected* to be displayed after the `font:` instruction.

Supplying context: prefix and suffix

When annotating an element, it's possible to supply a prefix and a suffix that should be prepended/appended when the language fragment is being parsed. This can be used to supply context information, i.e. if the prefix for a JavaScript injection is `"var someContextVariable;"`, PyCharm will know that the variable `someContextVariable` is declared and will not warn about it being undeclared when it's used.

Apart from the manual ability to supply a prefix and suffix, IntelliJLang dynamically determines those values from the context a String literal is being used in:

```
@Language(value =JavaScript, prefix = "function doSomethingElse(a){ }")
String code = "function doSomething() {\n" +
"  var x = 1;\n" +
"  doSomethingElse(x);\n" +
"}";
```

In this example, the JavaScript language will be injected into each of the three String literals, and each one's prefix and suffix will be calculated from the preceding and following expressions so that the resulting text that's being parsed is valid JavaScript syntax:

- no "missing }" error will be displayed: The closing brace is part of the first literal's suffix.
- the variable `x` that is used in `doSomethingElse(a);` will be declared: Its declaration is part of the second literal's prefix.
- the function `doSomethingElse()` will be known as well: It's defined in the statically supplied prefix.

Important notes

- There are some issues with the PyCharm Language Injection API that impose certain restrictions on the prefix/suffix of an injected language fragment. For instance, it's not allowed for a token of the language to span across the prefix/suffix of an element. This could e.g. happen if the prefix ends with a whitespace character and the fragment starts with whitespace.

The plugin deals with this special situation in the way that it trims the prefix/suffix and inserts exactly one space character as a separator. However, this doesn't work if a space character is no token separator, which e.g. applies to JavaScript string literals.

Such cases cannot be automatically dealt with and PyCharm core will produce an assertion.

- Even though the dynamic prefix/suffix calculation provides a proper context for the language fragment, some things may not work as expected. Most notably this are the refactoring (rename) and navigation functions.

Quick Edit Language

This is an experimental feature that can be extremely helpful, especially for editing Regular Expression patterns inside String literals due to the *double escaping* requirement. For example, in a regular expression, a literal backslash character has to be written as a double backslash and each of them has to be escaped with another backslash when written inside a String literal.

The Quick Edit function that appears as an Intention Action for any injected language fragment displays a pop-up dialog that allows you to edit the string's value without the double escaping requirement. The dialog also shows the particular prefix/suffix of the fragment in a non-editable area.



The pop-up window can be dismissed by pressing Escape or clicking somewhere outside the pop-up window. Any changes made in the pop-up window are committed by pressing Ctrl-Enter.

Please note that the formatting of the non-editable prefix/suffix may differ from the actual value due to some problems. However, even though the formatting/alignment of the editable text may differ from the expected text, making changes to the text still works as expected.

Warning! Accepting Inspection Quick-Fixes inside the QuickEdit pop-up window may cause strange things to happen when the Quick-Fix attempts to open an Editor for the element being edited. The *Create Method* Quick-Fix of the JavaScript language is an example for that.

Code Inspections

IntelliLang contains a set of inspections that validate the correct use of the supplied annotations (or custom configured ones). These inspections can be configured through the regular Settings | Errors configuration dialog.

Language Injection

The inspections in this category apply to the language injection feature related to the `@Language` annotation.

Unknown Language ID

This inspection provides validation for the use of non-existing Language-IDs. It flags usages of incorrect values of the `@Language` value attribute, such as `@Language("NonExistingID")`.

Language Mismatch

Validation for using references to elements that are annotated as containing different languages or are not annotated at all. The inspection offers a Quick-Fix to annotate such elements with the right annotation for the expected language.

Injection not applicable

This inspection checks whether an `@Language` or any derived annotation is used for anything other than elements of type `String` or `String[]`.

Pattern Validation

This category contains inspections about validating the use of the `@Pattern` or its derived annotations.

Validate Annotated Patterns

This inspection validates that expressions (String literals, as well as other compile-time constant or substituted expressions) match the pattern required by the `@Pattern` annotation. The inspection has an option to ignore non-constant expressions that contain non-substituted references and offers a Quick-Fix to add a substitution where applicable.

Pattern Annotation not applicable

Checks whether a pattern-validation annotation (`@Pattern` or derived ones) is valid to be applied to the annotated element. Only elements of type `String` may be annotated.

Non-annotated Method overrides @Pattern Method

This inspection checks whether a method without any `@Pattern` or derived annotation overrides an annotated method from its base classes. This is not necessary for the error-highlighting inside the editor, however the runtime-check instrumentation doesn't pick up annotations from base-class methods.

A Quick-Fix is provided to add an annotation that matches the one from the base-class method. This ensures that the runtime-check instrumentation works correctly.

IPython/Jupyter Notebook Support

In this section:

- IPython/Jupyter Notebook Support
 - [Prerequisite](#)
 - [Notebook support](#)
 - [Configuring an interpreter for a notebook](#)
 - [Creating and opening ipynb files](#)
 - [File *.ipynb in the editor](#)
- [Running IPython/Jupyter Notebook Cells](#)

Prerequisite

Prior to working, make sure that the following prerequisite is met:

Jupyter Notebook is properly installed on your computer. To learn how to install Jupyter Notebook, refer to the [official documentation](#).

Notebook support

PyCharm supports [Jupyter Notebook](#), recognizes *.ipynb files, and allows you to edit them.

Jupyter Notebook files are marked with  icon.

A Notebook file <file name>.ipynb shows document cells including code, text, equations etc.

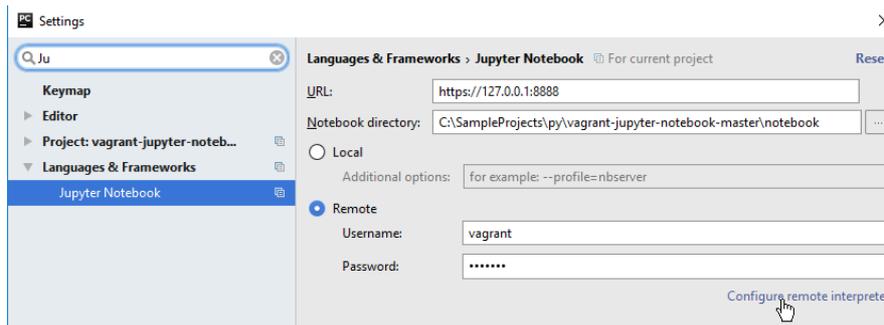
Notebook support includes:

- Support for [local](#) and [remote](#) interpreters. See section [IPython/Jupyter Notebook Support](#) for details.
- Coding assistance:
 - Error and syntax highlighting.
 - [Code completion](#).
 - Ability to create [line comments](#) ([Ctrl+Slash](#)).
- Search and replace facilities: [in a file](#), [in path](#).
- Dedicated kernel view: [Jupyter Notebook](#) tab in the [Run tool window](#).
- Ability to [run cells](#).

Configuring an interpreter for a notebook

To configure a remote connection to a notebook, follow these steps:

1. Open the [Jupyter Notebook](#) page of the Settings/Preferences dialog.
2. In the URL field, specify the https URL with https address of the user's remote notebook. Refer to the pages [Notebook server](#) and [Jupyter Hub](#).
3. Choose the Remote radio-button:



4. Fill in the username (for JupyterHub) and password.
5. Click the link [Configure remote interpreter](#). You'll find yourself at the [Project Interpreter](#) page.
6. Configure the remote interpreter, as described in the section [Configuring Remote Interpreters via Deployment Configuration](#).

Creating and opening ipynb files

To create an *.ipynb file:

1. Do one of the following:
 - Right-click the target directory in the Project tool window, and choose New on the context menu.
 - Press [Alt+Insert](#)
2. Choose the option Jupyter Notebook.
3. In the dialog box that opens, type the file name.

To open the existing *.ipynb files, follow the [same steps](#) as for the files of the other types.

File *.ipynb in the editor

The *.ipynb files feature a toolbar with the following buttons:

ItemTooltipDescription

	Save and checkpoint	Click this button to forcibly save all changes to the notebook in its current state (even if the calculations not finished yet).
	Insert cell below	Click this button to add an empty cell under the current one.
	Cut cell	Click this button to delete the current cell and place it to the clipboard.
	Copy cell	Click this button to create copy of the current cell in the clipboard.
	Paste cell below	Click this button to the paste contents of the clipboard below the current cell.
	Run cell	Click this button to execute the current cell .
	Interrupt kernel	Click this button to stop the current kernel.
	Restart kernel	Click this button to restart the current kernel.
Styles drop-down list		Select the desired presentation style from the drop-down list.

Running IPython/Jupyter Notebook Cells

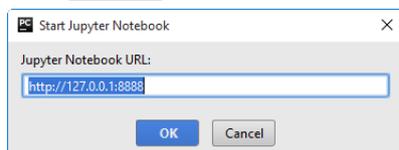
In this section:

- [Launching IPython Notebook](#)
- [Running cells](#)
- [Progress indication](#)

Launching IPython Notebook

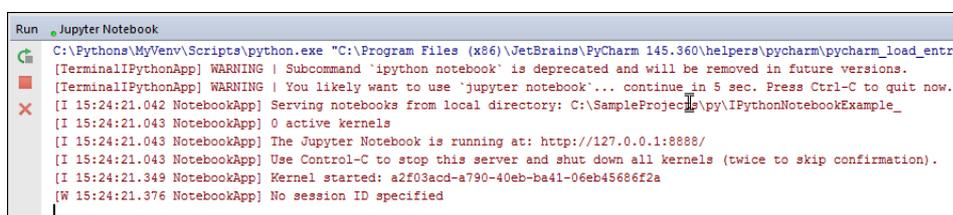
To run an IPython Notebook, follow these steps

1. Open the desired *.ipynb file for editing.
2. Press `Shift+F10`. PyCharm shows Start Jupyter Notebook dialog box where you have to specify the Notebook server URL:



If this URL has been entered in the [Jupyter Notebook](#) page of the Settings dialog, then this value will be specified by default. If you need a different server URL, you have to type it manually.

PyCharm shows the notebook kernel in the [Jupyter Notebook](#) tab of the Run tool window:



When running another notebook, the corresponding kernel starts:

```
[I 15:47:36.786 NotebookApp] Kernel started: 543dbe86-d327-41c7-8017-5188f62e37d5
[W 15:47:36.861 NotebookApp] No session ID specified
[I 15:53:33.894 NotebookApp] Kernel started: 44baa146-aa6d-4f89-a7ef-e9546e61403c
[W 15:53:33.901 NotebookApp] No session ID specified
```

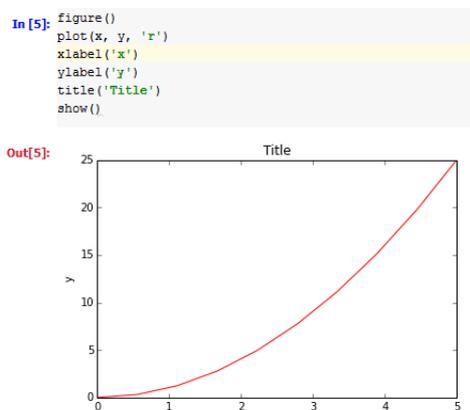
Tip Mind that IPython Notebook is now deprecated!

Running cells

To run a cell, follow these steps

1. Select the desired cell in the document.
2. On the document toolbar, click .

PyCharm executes the code of the selected cell, and shows the result below:



To run all cells in a Notebook

- Press `Shift+F10`.

Progress indication

Note that each cell has its number to the left:

```
In [2]: from __future__ import division
        from IPython.display import display
        from sympy.interactive import printing
        printing.init_printing(use_latex='mathjax')
```

When a cell is being executed, the number changes to an asterisk:

```
In [*]: from __future__ import division
        from IPython.display import display
        from sympy.interactive import printing
        printing.init_printing(use_latex='mathjax')
```

When ready, the cell number increases:

```
In [3]: from __future__ import division
        from IPython.display import display
        from sympy.interactive import printing
        printing.init_printing(use_latex='mathjax')
```

JavaScript-Specific Guidelines

This feature is supported in the Professional edition only.

With PyCharm, you can develop modern web, mobile, and desktop applications with **JavaScript** and **Node.js**. PyCharm supports **JavaScript** and **TypeScript** programming languages, **React** and **Angular** frameworks and provides tight integration with various tools for web development.

On this page:

- [Prerequisites](#)
- [JavaScript support](#)
- [Developing an application that contains JavaScript: general overview](#)
- [In this section](#)

Prerequisites

Before you start working with JavaScript, make sure that the JavaScript Support plugin is enabled. The plugin is bundled with PyCharm and is activated by default. If the plugin is not activated, enable it on the [Plugins](#) page of the [Settings / Preferences Dialog](#) as described in [Enabling and Disabling Plugins](#).

JavaScript support

JavaScript files are marked with  icon.

JavaScript support in PyCharm includes:

- Switching between JavaScript language versions:
 - [ECMAScript 3](#)
 - [ECMAScript 5.1](#)
 - [JavaScript 1.8.5](#)
 - [ECMAScript 6](#)
 - [React JSX](#)
 - [Flow](#)
- Full coding assistance:
 - Smart [code completion](#) for keywords, labels, variables, parameters, user-defined or built-in functions, and JavaScript namespaces.
 - Error and syntax highlighting.
 - Code [formatting](#) and [folding](#).
 - Numerous code [inspections](#) and [quick-fixes](#).
 - Support of the [strict mode](#) standard.
- [Code Generation](#)
 - Generating code stubs based on [file templates](#) during file creation.
 - Inserting, expanding, and generating JavaScript code blocks using [live templates](#).
 - Creating various applications elements via JavaScript and AJAX [intention actions](#).
 - Possibility to create [line and block comments](#) ([Ctrl+Slash](#)) / ([Ctrl+Shift+Slash](#)).
 - [Unwrapping and removing statements](#).
- Possibility to build and view [type](#), [method](#) and [call](#) hierarchies.
- Refactoring, both JavaScript-specific and available for all the supported languages, see [Refactoring Source Code](#) and [Extract Parameter in JavaScript](#).
- Numerous ways to [navigate](#) through the source code, among them [Navigating with Structure Views](#), [Show/Goto Implementation](#) ([Ctrl+Alt+B](#)) from overridden method / subclassed class, etc.
- Advanced facilities for [searching through the source code](#).
- Support of the [JSDoc](#) format and [generating documentation comments](#).
- [Viewing reference](#) information:
 - Definitions, inline documentation, parameter hints.
 - [JSDoc comments](#).
 - Lookup in [external JavaScript libraries](#).
- Running and debugging.
 - Launching applications directly from PyCharm by opening the starting application HTML page in the [default PyCharm browser](#).
 - A dedicated [debug](#) configuration for launching debugging sessions directly from PyCharm.
 - A JavaScript-aware [debugger](#) that lets you execute applications step by step, evaluate expressions, examine related information and find runtime bugs.
 - Support for [JavaScript breakpoints](#).
- Tight integration with related frameworks and technologies.
- Support for the [JSON](#) (JavaScript Object Notation) format:
 - A JSON file [type template](#) mapped to `.json` file extension.
 - JSON code [formatting](#) and [folding](#).

Developing an application that contains JavaScript: general overview

Developing an application that contains JavaScript, generally, includes performing the following steps:

1. [Create a project](#) to implement your application.
2. On the [JavaScript](#) page of the [Settings](#) dialog box, choose the JavaScript language version that you are going to use in your project.

3. Install the project dependencies via the **Node Package Manager** by doing one of the following:

- 1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing `File | Settings for Windows and Linux or PyCharm | Preferences` for macOS, and click `Node.js` and `NPM` under `Languages & Frameworks`.
 - 2. On the `Node.js` and `NPM` page that opens, the `Packages` area shows all the `Node.js`-dependent packages that are currently installed on your computer, both at the `global` and at the `project` level. Click `+`.
 - 3. In the `Available Packages` dialog box that opens, select the required package.
 - 4. Optionally specify the product version and click `Install Package` to start installation.
- Right-click the `package.json` file in the editor or in the `Project tool` window and choose `Run 'npm install'` on the context menu.
 - Open the built-in `Terminal` (`View | Tool Windows | Terminal`) and run the `npm install <required_package>` commands.
 - Install the required packages on the `Node.js and NPM` page of the [Settings / Preferences Dialog](#).

For details, see [Installing and Removing External Software Using Node Package Manager](#).

4. **Populate the project.** Use the following PyCharm facilities, where applicable:

- [Coding assistance](#).
- [Code Generation](#).
- Various types of [navigation](#) and [search](#) through the source code.
- [Viewing reference](#) and [generating documentation comments](#).
- [Look-up in external libraries](#).

5. Improve the quality and maintainability of your code using various types of [refactorings](#), built-in code quality tools, and integration with other linters, see [Using JavaScript Code Quality Tools](#).

6. Run your application by [opening its starting HTML page in the PyCharm default browser](#).

7. **Debug** your application in the [Google Chrome](#) browser.

The JavaScript debugging functionality is incorporated in PyCharm, so just [configure the debugger](#), whereupon you can start the [debugging session](#) and [proceed as usual](#): set the breakpoints, step through them, stop and resume the program, and examine it when suspended.

In this section

- [Configuring JavaScript Libraries](#)
- [Creating JSDoc Comments](#)
- [Viewing JavaScript Reference](#)
- [JavaScript-Specific Refactorings](#)
- [Testing JavaScript](#)
- [Running JavaScript in Browser](#)
- [Minifying JavaScript](#)
- [Using AngularJS](#)
- [Using Angular](#)
- [Using the Flow Type Checker](#)
- [Using JavaScript Code Quality Tools](#)
- [Using Meteor](#)
- [Using PhoneGap/Cordova](#)

Configuring JavaScript Libraries

This feature is supported in the Professional edition only.

On this page:

- [Basics](#)
 - [Predefined and custom libraries](#)
 - [Visibility and scope](#)
- [Viewing the libraries associated with a file](#)
- [Downloading and installing a JavaScript-related library from PyCharm](#)
- [Configuring a custom JavaScript library](#)
 - [Removing a library file](#)
 - [Updating the contents of a library](#)
 - [Deleting a library](#)
 - [Specifying the scope to use a library in](#)

Basics

When working in PyCharm, you can use various JavaScript libraries and frameworks with full coding assistance at disposal. The only thing you need is "introduce" the required libraries to PyCharm so they are recognized and references to them are successfully resolved.

Predefined and custom libraries

Right from PyCharm, you can download and install a number of the most popular JavaScript libraries, such as: [Dojo](#), [ExtJS](#), [jQuery](#), [jQuery UI](#), [Prototype](#), and others. Once installed, these libraries, by default, are visible in all files within the project.

You can also download any other JavaScript libraries and frameworks or even develop JavaScript libraries yourself, if necessary. To use such downloaded or self-developed libraries, you need to [set them up as custom PyCharm JavaScript libraries](#). Configured libraries can be used in [code completion](#), highlighting, [navigation](#), and [Documentation Lookup](#).

To enable Documentation Lookup for symbols defined in an external library or framework, [provide a link](#) to its documentation. Upon pressing  with the cursor positioned at the symbol in question, PyCharm invokes this link and opens the documentation page in browser.

For [jQuery](#), [jQuery UI](#), [Ext JS](#), [Prototype](#), and [Dojo](#) library files, PyCharm automatically detects links to the library documentation.

To get all the above mentioned coding assistance for a library or a framework, you need to [configure it as a PyCharm library](#) and [specify the library scope](#) by associating the library with your project or its part, so you can reference the library from the files within this scope, get code completion, and retrieve definitions and documentation.

Please note, that configuring a framework as a PyCharm library and associating it with a project only ensures coding assistance in the **development environment**, that is, while you are working in PyCharm. To use a framework or library in the **production environment**, make sure the relevant version of the framework is available on the server.

Visibility and scope

Each PyCharm library is characterized by its **visibility** status and **scope**.

The **scope** of a library defines the set of files and folders in which the library is considered as **library**, that is, it is write-protected, excluded from check for errors and refactoring, but only affects the completion list and highlighting.

The **visibility** status of a library determines whether it can be used in one project (**Project**) or can be re-used at the IDE level (**Global**).

- Once configured, a **Global library** can be associated with any of PyCharm projects. The library itself can be located wherever you need, its settings are stored with other PyCharm settings in the dedicated directories under the PyCharm home directory.

The **advantage** of configuring a framework as a global library is that you can store such library in one location and re-use it in unlimited number of your projects without copying the library under the project root every time.

The **disadvantage** of this approach is that to enable team work on a project all the team members should have the library stored on their machines in the same location relative to the project root.

- A **Project library** is **visible** only within one single project. Therefore a project library can be associated only with this project or its part. This means that project libraries cannot be re-used, so if you later try to use a framework configured as a project library with another project, you will have to configure the library anew.

The **advantage** of configuring a JavaScript framework as a project library is that you can share the library definition among other team members through the [project settings](#) so each developer does not have to configure the library separately.

- **Predefined** libraries bring JavaScript definitions (also known as "stubs") for the standard DOM, HTML5 and EcmaScript API, as well as for Node.js global objects. These libraries make the basis for coding assistance in accordance with the API provided by the corresponding JavaScript engine. By enabling a certain predefined library you can ensure that your code fits the target environment.

Predefined libraries are by default enabled in the **scope** of the entire project. A predefined library can be disabled or [associated with another scope](#) but it cannot be [deleted](#).

Viewing the libraries associated with a file

1. Open the file in the editor.
2. Click the Hecto icon  on the Status bar. The pop-up window lists the libraries associates with the current file. To change the list, click the Libraries in scope links and edit the scope settings in the [Manage Scope](#) dialog box that opens.

Downloading and installing a JavaScript-related library from PyCharm

PyCharm provides a dedicated user interface for downloading and installing the most popular **official** JavaScript libraries. Using this interface, you can download and install [Dojo](#), [ExtJS](#), [jQuery](#), [jQuery UI](#), [Prototype](#), and other libraries.

Besides the above listed **official** libraries, you can download [stubs for TypeScript definition files](#).

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS. Expand the Languages and Frameworks node, and then click Libraries under JavaScript.
2. The [JavaScript Libraries](#) page that opens shows a list of all the already available libraries. Click Download.
3. In the Download Library dialog that opens, choose the group of libraries in the drop-down list. The available options are Official libraries and TypeScript community stubs. Depending on your choice, PyCharm displays a list of available libraries. Select the one to be downloaded and installed, and click Download and Install. You return to the [JavaScript Libraries](#) page where the new library is added to the list. Click OK to save the settings.

Configuring a custom JavaScript library

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS. Expand the Languages and Frameworks node, and then click Libraries under JavaScript.
2. The [JavaScript Libraries](#) page that opens shows a list of all the already available libraries. Click Add.
3. In the [New Library](#) dialog box that opens, specify the name of the library, the framework to configure the library from, and framework version to use.
4. Specify the library **visibility**:
 - To enable associating the library with the current project only, choose Current project.
 - To make the library available in any PyCharm project, choose Global.
5. Create a list of files to be included in the library:
 1. Click the Add button `+` next to the list of library files and choose Attach Files or Attach Directory on the context menu, depending of whether you need separate files or an entire folder.
 2. Select the required file, files, or entire directory in the dialog box that opens. PyCharm returns to the New Library dialog box where the Name read-only field shows the name of the selected library file or the names of relevant library files from the selected directory.
 3. In the Type field, specify which version of library you have downloaded and are going to add.
 - Choose Debug if you are adding a library file with uncompressed code. This version is helpful in the development environment, especially for debugging.
 - Choose Release if you are adding a library file with [minified](#) code. This version is helpful in the production environment because the file size is significantly smaller.

It is recommended that you always have a debug version on hand along with the minified one. Minified code is hard to read and hard for PyCharm to handle. When a debug version is available, PyCharm automatically detects and ignores the minified file and retrieves definitions and documentation from the debug version.

4. Specify the URL addresses to access the documentation for library files.
 - To add a link to the documentation for a library, select the corresponding library file, click the Specify button in the Documentation URLs area, and specify the documentation URL in the dialog box that opens.

Tip For [jQuery](#), [jQuery UI](#), [Ext JS](#), [Prototype](#), and [Dojo](#) libraries, PyCharm automatically detects the link to the library documentation and suggests it in the Enter documentation URL text box.

- To remove a link, select it in the Documentation URLs and click the Remove button.

Removing a library file

In the [New Library/Edit Library](#) dialog box, select the required library file and click the Remove button `-`.

Updating the contents of a library

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS. Expand the Languages and Frameworks node, and then click Libraries under JavaScript.
2. The [JavaScript Libraries](#) page that opens shows a list of all the already available libraries. Select the required library and click Edit.
3. In the [Edit Library](#) dialog box that opens, **add** library files, **remove** library files, and **change links to documentation** as necessary.

Deleting a library

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS. Expand the Languages and Frameworks node, and then click Libraries under JavaScript.
2. The [JavaScript Libraries](#) page that opens shows a list of all the already available libraries. Select the required library and click Remove.

Specifying the scope to use a library in

The **scope** of a library defines the set of files and folders in which the library is considered as **library**, that is, it is write-protected, excluded from check for errors and refactoring, but only affects the completion list and highlighting.

By default, all [predefined libraries](#) and [libraries downloaded from PyCharm](#) provide completion, resolution, highlighting and are treated as libraries in any file within the project. In other words, their usage scope is the whole project.

[Libraries that you create yourself](#) are not considered libraries in any of the files unless you specify their usage scope explicitly.

To specify the scope for a library:

Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for

1. Open the [Settings / Preferences Dialog](#) by pressing [\(CTRL/ALT+S\)](#) or by choosing [File | Settings](#) for Windows and Linux or [PyCharm | Preferences](#) for macOS. Expand the Languages and Frameworks node, and then click Libraries under JavaScript.
2. The [JavaScript Libraries](#) page that opens shows a list of all the already available libraries. Click [Manage Scopes](#).
3. In the [JavaScript Libraries. Usage Scope](#) dialog box that opens, specify the custom JavaScript libraries to use in files and folders within your project. To appoint a library for a file or folder, select the required item in the File/Directory field and choose the relevant library from the Library drop-down list. The contents of the list depend on the [visibility](#) type of the configured libraries. **Global** libraries are on the list in all PyCharm projects. **Project** libraries are on the list only within the project they were originally configured in.

Creating JSDoc Comments

This feature is supported in the Professional edition only.

To make your code easy to use by other developers it is considered good practice to provide an HTML documentation of its Application Programming Interface (API). Such documentation can be generated automatically by the [JSDoc](#) tool. All you need, is supply your code with **documentation comments** in accordance with the [JSDoc standard](#). The tool retrieves information from your comments and renders it in HTML using a built-in template.

You can find a detailed description of the JSDoc syntax with examples and explanation of their use in the article [An Introduction to JSDoc](#).

PyCharm creates stubs of JSDoc comments on typing the opening tag `/**` and pressing `Enter`. If this feature is [applied to a method or a function](#), `@param` tags are created. In any other places PyCharm adds an empty documentation stub.

[TODO patterns](#) and [Closure Compiler](#) annotations inside documentation comments are also recognized and are involved in code completion, intention actions, and other types of coding assistance.

Documentation comments in your source code are available for the [Quick Documentation Lookup](#) and open for review on pressing `Ctrl+Q`.

PyCharm checks syntax in the comments and treats it according to the [Code Inspections](#) settings.

On this page:

– [Example of JavaScript comment](#)

Example of JavaScript comment

Consider the following function:

```
function loadDocs(myParam1, myParam2){}
```

Type the opening documentation comment and press `Enter` to generate the documentation comment stub:

```
/**
 * @param myParam1
 * @param myParam2
 */
```

Enabling documentation comments

1. [Open Settings/Preferences dialog box](#), expand the Editor node, then expand General node, and click the [Smart Keys](#) page.
2. In the Enter section, select or clear Insert documentation comment stub check box.
3. Then, scroll to the Insert type placeholders in the documentation comment stub option and select or clear the check box as required. Refer to [the option description](#) for details.

Creating a JSDoc comment block

1. Place the caret before the method or function declaration.
2. Type the opening block comment `/**` and press `Enter`.
3. Describe the listed parameters and return values.

Documentation comment can be created with the dedicated action Fix Doc Comment. It can be invoked by means of [Find Action](#) command.

Press `Ctrl+Shift+A`, with the caret somewhere within a class, method, function, or field, which should be documented, and enter the action name Fix Doc String. The missing documentation stub with the corresponding tags is added. For example:

```
function loadDocs(myParam1, myParam2){}
```

Type the opening documentation comment and press `Enter` to generate the documentation comment stub:

```
/**
 * @param myParam1
 * @param myParam2
 */
```

The next case lays with fixing problems in the existing documentation comments.

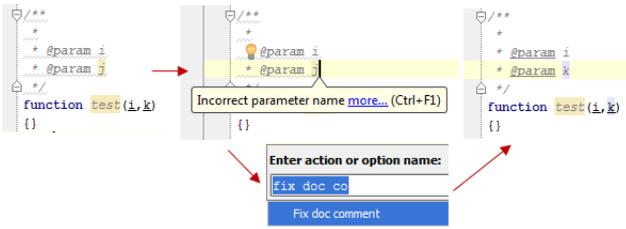
For example, if a method signature has been changed, PyCharm highlights a tag that doesn't match the method signature, and suggests a quick fix.

For Python, PyCharm suggests an inspection Ignore unresolved reference:

```
def isPrime(n1):  
    """  
    """  
    :param n1:  
    """
```

Function 'isPrime' does not have a parameter 'n' more... (Ctrl-F1)

For JavaScript, PyCharm suggests an intention action UpdateJSDoc comment. You can also press `Ctrl+Shift+A`, and type the action name:



Tip The action Fix doc comment has no keyboard shortcut bound with it. You can [configure keyboard shortcut](#) of your own.

Viewing JavaScript Reference

This feature is supported in the Professional edition only.

Besides [common referential procedures](#) that are available in the context of any supported language, PyCharm provides the following ways of retrieving JavaScript-specific reference:

- [Viewing Inline Documentation](#)
- [Documentation Look-up in External JavaScript Libraries](#)

This feature is supported in the Professional edition only.

Quick Documentation Lookup helps you get quick information for any symbol, provided that this symbol has been supplied with Documentation comments in the applicable format. PyCharm recognizes inline documentation created in accordance with the [JavaScript Documentation Tool](#) format.

On this page:

- [Viewing documentation for a symbol at caret](#)
- [Changing font style of the quick documentation](#)
- [Toolbar of the quick documentation lookup](#)

To view documentation for a symbol at caret, do one of the following

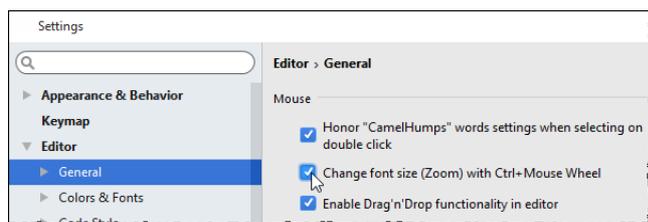
- On the main menu, choose View | Quick Documentation Lookup.
- Press `Ctrl+Q`.
- Provided that the check box [Show quick doc on mouse move](#) in the editor settings is selected, just move your mouse pointer over the desired symbol.

When you explicitly invoke code completion, then quick documentation for an entry selected in the suggestion list can be displayed automatically. The behavior of quick documentation lookup is configured in the [Code Completion](#) page of the Settings/Preferences dialog.

Tip Sequential pressing `Ctrl+Q` toggles the focus of the Quick Documentation pop-up window and [Documentation Tool Window](#).

To change the font size of quick documentation, do one of the following

- Click  in the upper-right corner of the quick documentation window, and move the slider.
 - Rotate the mouse wheel while keeping the `Ctrl` key pressed.
- Note that for this feature to work, you have to enable it in the [General](#) page of the editor settings.



Refer to the section [Zooming in the Editor](#) for details.

Toolbar of the quick documentation lookup

The Quick Documentation Lookup window helps navigate to the related symbols via hyperlinks, and provides a toolbar for moving back and forth through the already navigated pages, changing font size, and viewing documentation in an external browser.

When pinned, the Quick Documentation Lookup turns into [Documentation Tool Window](#), with the corresponding sidebar icon, and more controls.

IconShortcutDescription

	Left or Right	Switch to the previous or next documentation page (e.g. after using hyperlinks).
Note On an macOS computer, you can also use the three-finger right-to-left and left-to-right swipe gestures.		
	Shift+F1	View external documentation in the default browser.
	F4	Switch to the item (e.g. source) that corresponds to the documentation page currently shown.
		Turn the Auto-update from source option on or off. When the option is on, the information in the tool window is synchronized with your navigation in the editor and other places in the UI.
		Click this icon to show font size slider. Move the slider to increase or decrease the font size in the quick documentation window as required.

Documentation Look-up in External JavaScript Libraries

This feature is supported in the Professional edition only.

Besides [common referential procedures](#) and [viewing JSDoc comments](#), you can also retrieve documentation for symbols defined in external JavaScript libraries and frameworks.

When this functionality is enabled, upon pressing `Shift+F1` with the cursor positioned at the symbol in question, PyCharm invokes the corresponding link and opens the documentation page in the [default PyCharm browser](#).

- The default PyCharm browser is configured on the [Web Browsers](#) page of the [Settings](#) dialog.
- For more details on specifying the browser to use by default, see [Configuring Browsers](#).

To enable the external JavaScript library lookup functionality

1. Download the required libraries or frameworks.
2. [Configure the downloads as libraries](#) at the PyCharm level.
3. [Specify links to external documentation](#).

To view documentation on a symbol defined in an external JavaScript library

- Position the cursor at the symbol in question and choose `View | External Documentation`, or press `Shift+F1`.

JavaScript-Specific Refactorings

This feature is supported in the Professional edition only.

In addition to the full range of [common refactorings](#), PyCharm provides the following JavaScript-specific refactorings:

- [Change Signature in JavaScript](#)
- [Extract Parameter in JavaScript](#)
- [Extract Variable in JavaScript](#)

Change Signature in JavaScript

This feature is supported in the Professional edition only.

On this page:

- [Introduction](#)
- [Examples](#)
- [Changing function signature](#)

Introduction

In JavaScript, you can use the Change Signature refactoring to:

- Change the function name.
- Add new parameters and remove the existing ones. Note that you can also add a parameter using a dedicated [Extract Parameter](#) refactoring.
- Reorder parameters.
- Change parameter names.
- Propagate new parameters through the method call hierarchy.

Examples

The following table shows 4 different ways of performing the same Change Signature refactoring.

In all the cases, the function `result()` is renamed to `generate_result()` and a new parameter `input` is added to this function.

The examples show how the function call, the calling function (`show_result()`) and other code fragments may be affected depending on the refactoring settings.

BeforeAfter

<pre>// This is the function whose signature will be changed: function result() { // some code here } function show_result() { // Here is the function call: alert('Result: ' + result()); } // Now we'll rename the function and // add one (required) parameter.</pre>	<pre>// The function has been renamed to generate_result. // The new parameter input has been added. function generate_result(input) { // some code here } function show_result() { // The function call changed accordingly: alert('Result: ' + generate_result(100)); } // When performing the refactoring, 100 was specified as // the parameter value.</pre>
<pre>// This is the function whose signature will be changed: function result() { // some code here } function show_result() { // Here is the function call: alert('Result: ' + result()); } // Now we'll rename the function and add one parameter. // This time, we'll specify that the parameter is optional.</pre>	<pre>// The function has been renamed to generate_result. // The new optional parameter input has been added. function generate_result(input) { input = input 100; // some code here } function show_result() { // The function call changed accordingly: alert('Result: ' + generate_result(100)); } // When performing the refactoring, 100 was specified as // the parameter value.</pre>
	<pre>// The function has been renamed to generate_result. // The new parameter input has been added. function generate_result(input) { // some code here } // Note the new function parameter: function show_result(input) { // The function call changed accordingly: alert('Result: ' + generate_result(input)); } }</pre>

```
// This is the function whose signature will be changed:

function result() {
    // some code here
}

// This function will also change its signature:

function show_result() {

    // Here is the function call:

    alert('Result: ' + result());
}

// Now we'll rename the function and add one required
// parameter. We'll also ask PyCharm to propagate
// the new parameter through the calling function show_result()
// to the function call.
```

```
// This is the function whose signature will be changed:
function result() {

    // some code here
}

// This function will also change its signature:

function show_result() {

    // Here is the function call:

    alert('Result: ' + result());
}

// Now we'll rename the function and add one optional
// parameter. We'll also ask PyCharm to propagate
// the new parameter through the calling function show_result()
// to the function call.
```

```
// The function has been renamed to generate_result.
// The new optional parameter input has been added.

function generate_result(input) {
    input = input || 100;
    // some code here
}

// Note the new function parameter:

function show_result(input) {
    input = input || 100;

    // The function call changed accordingly:

    alert('Result: ' + generate_result(input));
}

// When performing the refactoring, 100 was specified as
// the parameter value.
```

Changing function signature

To change a function signature

- In the editor, place the cursor within the name of the function whose signature you want to change.
- Do one of the following:
 - Press `Ctrl+F6`.
 - Choose Refactor | Change Signature in the main menu.
 - Select Refactor | Change Signature from the context menu.
- In the [Change Signature dialog](#), make the necessary changes to the function signature and specify which other, related changes are required. You can:
 - Change the function name. To do that, edit the text in the Name field.
 - Manage the function parameters using the table of parameters and the buttons to the right of it:
 - To add a new parameter, click `+` (`Alt+Insert`) and specify the properties of the new parameter in the corresponding fields. When adding parameters, you may want to [propagate these parameters](#) to the functions that call the current function.
 - To remove a parameter, click any of the cells in the corresponding row and click `-` (`Alt+Delete`).
 - To reorder the parameters, use `↑` (`Alt+Up`) and `↓` (`Alt+Down`). For example, if you wanted to make a certain parameter the first in the list, you would click any of the cells in the row corresponding to that parameter, and then click `↑` the required number of times.
 - To change the name of a parameter, make the necessary edits in the corresponding table cell.
 - Propagate new function parameters (if any) along the hierarchy of the functions that call the current function. (There may be the functions that call the function whose signature you are changing. These functions, in their turn, may be called by other functions, and so on. You can propagate the changes you are making to the parameters of the current function through the hierarchy of the calling functions and also specify which calling functions should be affected and which shouldn't.)

To propagate the new parameters:

 - Click `☒`.
 - In the left-hand pane of the Select Methods to Propagate New Parameters dialog, expand the necessary nodes and select the check boxes next to the functions you want the new parameters to be propagated to. To help you select the necessary functions, the code for the calling function and the function being called is shown in the right-hand part of the dialog (in the Caller Method and Callee Method panes respectively).

As you switch between the functions in the left-hand pane, the code in the right-hand part changes accordingly.

 - Click OK.

4. To perform the refactoring right away, click Refactor.

To [see the expected changes](#) and make the necessary adjustments prior to actually performing the refactoring, click Preview.

Extract Parameter in JavaScript

This feature is supported in the Professional edition only.

The Extract Parameter refactoring is used to add a new parameter to a method declaration and to update the method calls accordingly.

– [Examples](#)

– [Extracting a parameter in JavaScript](#)

Examples

The following table shows two different ways of extracting a function parameter.

In the first of the examples a new parameter is extracted as an optional parameter. So the corresponding function call doesn't change.

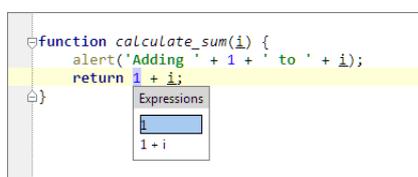
In the second of the examples the parameter is extracted as a required parameter. So the corresponding function call changes accordingly.

BeforeAfter

<pre>// A new parameter will be added to this // function to replace the 1's: function calculate_sum(i) { alert('Adding ' + 1 + ' to ' + i); return 1 + i; } function show_sum() { // Here is the function call: alert('Result: ' + calculate_sum(5)); } // When adding a new parameter we'll specify // that it should be an optional one.</pre>	<pre>// The new parameter i2 has been added // as an optional parameter: function calculate_sum(i, i2) { i2 = i2 1; alert('Adding ' + i2 + ' to ' + i); return i2 + i; } function show_sum() { // The function call has not changed: alert('Result: ' + calculate_sum(5)); }</pre>
<pre>// A new parameter will be added to this // function to replace the 1's: function calculate_sum(i) { alert('Adding ' + 1 + ' to ' + i); return 1 + i; } function show_sum() { // Here is the function call: alert('Result: ' + calculate_sum(5)); } // When adding a new parameter we'll specify // that it should be a required one.</pre>	<pre>// The new parameter i2 has been added // as a required parameter: function calculate_sum(i, i2) { alert('Adding ' + i2 + ' to ' + i); return i2 + i; } function show_sum() { // The function call changed accordingly: alert('Result: ' + calculate_sum(5, 1)); }</pre>

Extracting a parameter in JavaScript

1. In the editor, place the cursor within the expression to be replaced by a parameter.
2. Do one of the following:
 - Press **Ctrl+Alt+P**.
 - Choose Refactor | Extract | Parameter on the main menu.
 - Choose Refactor | Extract | Parameter from the context menu.
3. If more than one expression is detected for the current cursor position, the Expressions list appears. If this is the case, select the required expression. To do that, click the expression. Alternatively, use the **Up** and **Down** arrow keys to navigate to the expression of interest, and then press **Enter** to select it.



4. In the [Extract Parameter](#) dialog:
 1. Specify the parameter name in the Name field.
 2. The Value field, initially, contains the expression that you have selected. Normally, you don't need to change this. (Depending on whether the

option Optional parameter is selected or not, this will be the value assigned to the new parameter in the function body or passed to the function in the function calls.)

3. If, when extracting a new parameter, you don't want to change the function calls, select the Optional parameter check box. If you do so, the value specified in the Value field will be assigned to the new parameter in the function body. The calls to the function (if any) won't change. If you want to pass the value (specified in the Value field) to the new parameter through the existing function calls, clear the Optional parameter check box. If you do so, all the function calls will change according to the new function signature; no explicit assignment of the parameter value will be added to the function body.
4. If more than one occurrence of the expression is found within the function body, you can choose to replace only the selected occurrence or all the found occurrences with the references to the new parameter. Use the Replace all occurrences check box to specify your intention.
5. Click OK.

```
function calculate_sum(i, i2) {  
  i2 = i2 || 1;  
  alert('Adding ' + i2 + ' to ' + i);  
  return i2 + i;  
}
```

Extract Variable in JavaScript

This feature is supported in the Professional edition only.

In JavaScript, you can replace an expression with a variable or a constant. For JavaScript 1.7 or a later version, there is also an option of extracting a local variable.

To perform this refactoring, you can use:

- [In-place refactoring](#). In this case you specify the new name right in the editor.
- [Refactoring dialog](#), where you specify all the required information. To make such a dialog accessible, you have to clear the check box [Enable in-place mode](#) in the editor settings.

JavaScript examples

Before/After

<pre>Parentizer.method('toString', function () { return '(' + this.getValue() + ')'; })</pre>	<p>GLOBAL VARIABLE</p> <pre>Parentizer.method('toString', function () { var string = '(' + this.getValue() + ')'; return string; })</pre>
<pre>var browserName = "N/A"; if (navigator.appName.indexOf("Netscape") != -1) { browserName = "NS"; } else if (navigator.appName.indexOf("Microsoft") != -1) { browserName = "MSIE"; } else if (navigator.appName.indexOf("Opera") != -1) { browserName = "O"; }</pre>	<p>LOCAL VARIABLE</p> <pre>Parentizer.method('toString', function () { let string = '(' + this.getValue() + ')'; return string; })</pre>
<pre>var browserName = "N/A"; if (navigator.appName.indexOf("Netscape") != -1) { browserName = "NS"; } else if (navigator.appName.indexOf("Microsoft") != -1) { browserName = "MSIE"; } else if (navigator.appName.indexOf("Opera") != -1) { browserName = "O"; }</pre>	<pre>var browserName = "N/A"; var appName = navigator.appName; if (appName.indexOf("Netscape") != -1) { browserName = "NS"; } else if (appName.indexOf("Microsoft") != -1) { browserName = "MSIE"; } else if (appName.indexOf("Opera") != -1) { browserName = "O"; }</pre>

To extract a variable using in-place refactoring

1. In the editor, select the expression to be replaced with a variable. You can do that yourself or use the [smart expression selection](#) feature to let PyCharm help you. So, do one of the following:

- Highlight the expression. Then choose Refactor | Extract | Variable on the main menu or on the context menu. Alternatively, press **Ctrl+Alt+V**.

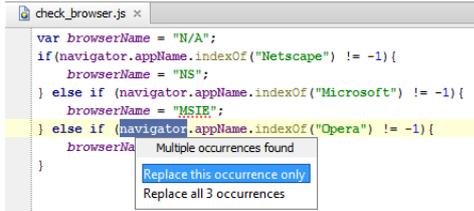
- Place the cursor before or within the expression. Choose Refactor | Extract Variable on the main menu or on the context menu. or press **Ctrl+Alt+V**.

In the Expressions pop-up menu, select the expression. To do that, click the required expression. Alternatively, use the Up and Down arrow keys to navigate to the expression of interest, and then press **Enter** to select it.

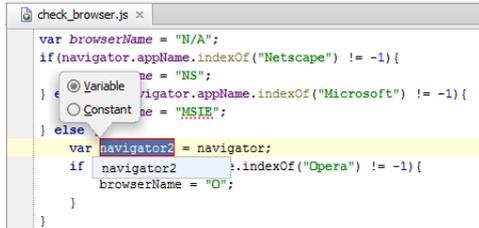
Note The Expressions pop-up menu contains all the expressions appropriate for the current cursor position in the editor.

When you navigate through the suggested expressions in the pop-up, the code highlighting in the editor changes accordingly.

- If more than one occurrence of the selected expression is found, select Replace this occurrence only or Replace all occurrences in the Multiple occurrences found pop-up menu. To select the required option, just click it. Alternatively, use the **Up** and **Down** arrow keys to navigate to the option of interest, and press **Enter** to select it.



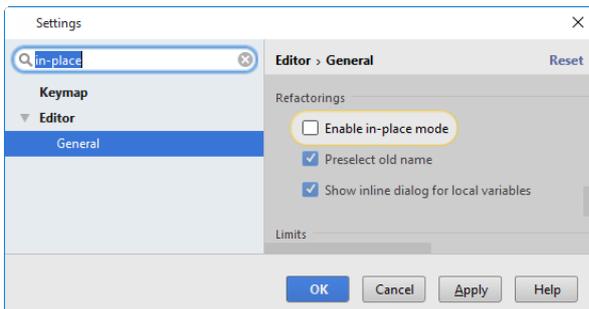
- Specify whether you want to replace selected expression with a variable, or a constant. To do that, click the desired radio button in the pop-up balloon:



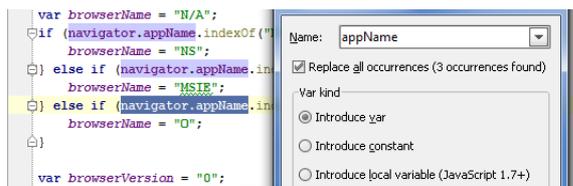
- Specify the name of the variable. Do one of the following:
 - Select one of the suggested names from the pop-up list. To do that, double-click the suitable name. Alternatively, use the **Up** and **Down** arrow keys to navigate to the name of interest, and **Enter** to select it. When finished, press **Escape**.
 - Edit the name by typing. The name is shown in the box with red borders and changes as you type. When finished, press **Escape**.

To extract a variable using the dialog box

If the Enable in place refactorings check box is cleared in the Editor settings, the Extract Variable refactoring is performed by means of the dialog box.



- Select the desired expression, and invoke Extract Variable refactoring as described [above](#).
- If more than one expression is detected for the current cursor position, the Expressions list appears. If this is the case, select the required expression. To do that, click the expression. Alternatively, use the **Up** and **Down** arrow keys to navigate to the expression of interest, and then press **Enter** to select it.
- In the **Extract Variable Dialog**:
 - Specify the variable name next to Name. You can select one of the suggested names from the list or type the name in the Name box.
 - If more than one occurrence of the selected expression is found, you can select to replace all the found occurrences by selecting the corresponding check box. If you want to replace only the current occurrence, clear the Replace all occurrences check box.
 - Specify the type of variable to be extracted: a (global) variable, a constant, or a local variable. To do that, click the required option in the Var kind area.
(Note that JavaScript is not supported in PyCharm Community Edition.)
 - Click OK.



Testing JavaScript

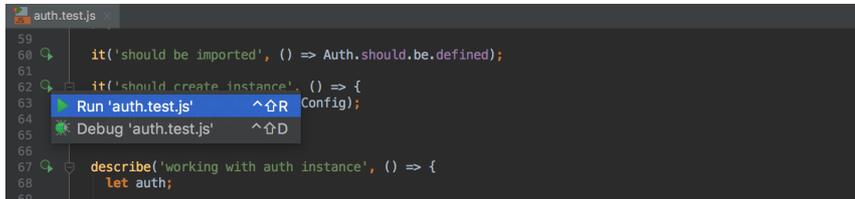
This feature is supported in the Professional edition only.

In PyCharm, you can run JavaScript test cases, methods, or entire test files against a local or remote test server using [Cucumber.js](#), [Jest](#), [JSTestDriver](#), [Karma](#), [Mocha](#), or [Protractor](#).

You can quickly jump from source code to the related test file with the Go to test action (`Ctrl+Shift+T` or `Navigate | Test`). For example, from `auth.js` you can jump to `auth.test.js`.

You can also see whether a test has passed or failed right in the editor, thanks to the **test status** icons  in the left gutter.

For **Mocha** and **Jest**, you can run and debug tests and suites right from the editor: click  or  in the left gutter and choose Run `<test_file_name>` or Debug `<test_file_name>` from the pop-up list.



With **JSTestDriver** and **Karma**, you can also monitor how much of your code is [covered with tests](#). PyCharm displays this statistics in a dedicated tool window and marks covered and uncovered lines visually right in the editor.

In this part:

- [Enabling JavaScript Unit Testing Support](#)
- [Creating JavaScript Unit Tests](#)
- [Running JavaScript Unit Tests](#)

Enabling JavaScript Unit Testing Support

This feature is supported in the Professional edition only.

With PyCharm, you can run and debug unit tests using one of the following **test runners**:

- [JSTestDriver](#)
- [Karma](#)
- [Cucumber.js](#)

To write unit tests that are recognized and processed by these test runners, use the following JavaScript unit testing frameworks:

- [JSTestDriver Assertion](#) framework. The framework libraries are bundled with the JSTestDriver plugin.
- [Jasmine](#).
- [QUnit](#).
- [Mocha](#)

Preparing to Use Cucumber.js Test Runner

This feature is supported in the Professional edition only.

The [Cucumber.js](#) test runner supports executing unit tests against the [Node.js](#) server. [Cucumber.js](#) runs tests that are called [features](#) and are written in the [Gherkin](#) language. Each [feature](#) is described in a separate file with the extension `feature`. [Feature](#) files are marked with the  icon.

The easiest way to install the Cucumber.js test runner is to use the [Node Package Manager \(npm\)](#), which is a part of [Node.js](#). See [Installing and Removing External Software Using Node Package Manager](#) for details.

Depending on the desired location of the Cucumber.js test runner executable file, choose one of the following methods:

- Install the test runner [globally](#) at the PyCharm level so it can be used in any PyCharm project.
- Install the test runner in a specific project and thus restrict its use to this project.
- Install the test runner in a project as a [development dependency](#).

In either installation mode, make sure that the parent folder of the Cucumber.js test runner is added to the `PATH` variable. This enables you to launch the test runner from any folder.

PyCharm provides user interface both for [global](#) and [project](#) installation as well as supports installation through the command line.

On this page:

- [Preparing to install Cucumber.js](#)
- [Installing Cucumber.js globally](#)
- [Installing Cucumber.js in a project](#)
- [Installing Cucumber.js as a development dependency](#)

Preparing to install Cucumber.js

1. Download and install [Node.js](#). The runtime environment is required for two reasons:

- The Cucumber.js test runner is started through [Node.js](#).
- [NPM](#), which is a part of the runtime environment, is also the easiest way to download the Cucumber.js test runner.

If you are going to use the command line mode, make sure the path to the parent folder of the [Node.js](#) executable file and the path to the `npm` folder are added to the `PATH` variable. This enables you to launch the Cucumber.js test runner and `npm` from any folder.

2. Install and enable the [NodeJS](#) repository plugin as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).

3. Install and enable the [Cucumber.js](#) and [Gherkin](#) plugins. The plugins are not bundled with PyCharm, but they can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#). Once enabled, the plugins are available at the IDE level, that is, you can use them in all your PyCharm projects.

Installing Cucumber.js globally

[Global](#) installation makes the test runner available at the PyCharm level so it can be used in any PyCharm project. Moreover, during installation the parent folder of the test runner is automatically added to the `PATH` variable, which enables you to launch the test runner from any folder.

Run the installation from the command line in the [global](#) mode: launch the embedded Terminal (View | Tool Windows | Terminal or by hovering your mouse pointer over  in the lower left corner of PyCharm and choosing Terminal from the menu and type `npm install -g cucumber` at the command line prompt. The `-g` key makes the test runner run in the [global](#) mode. Because the installation is performed through [NPM](#), the Cucumber.js test runner is installed in the `npm` folder. Make sure this parent folder is added to the `PATH` variable. This enables you to launch the test runner from any folder.

For more details on the [NPM](#) operation modes, see [npm documentation](#). For more information about installing the Cucumber.js test runner, see <https://npmjs.org/package/cucumber>. Run [NPM](#) from PyCharm using the Node.js and NPM page of the Settings dialog box.

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Node.js and NPM under Languages & Frameworks.
2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the [global](#) and at the [project](#) level. Click `+`.
3. In the Available Packages dialog box that opens, select the `cucumber` package.
4. Select the Options check box and type `-g` in the text box next to it.
5. Optionally specify the product version and click Install Package to start installation.

Installing Cucumber.js in a project

[Local](#) installation in a specific project restricts the use of the test runner to this project.

Run the installation from the command line: launch the embedded Terminal (View | Tool Windows | Terminal or by hovering your mouse pointer over  in the lower left corner of PyCharm and choosing Terminal from the menu and type `npm install cucumber` at the command line prompt. Run [NPM](#) from PyCharm using the Node.js and NPM page of the Settings dialog box.

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Node.js and NPM under Languages & Frameworks.
2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the [global](#) and at the [project](#) level. Click `+`.
3. In the Available Packages dialog box that opens, select the `cucumber` package.
4. Optionally specify the product version and click Install Package to start installation.

Project level installation is helpful and reliable in [template-based projects](#) of the type [Node Boilerplate](#) or [Node.js Express](#), which already have the `node_modules` folder. The latter is important because [NPM](#) installs the [Cucumber.js](#) test runner in a `node_modules` folder. If your project already contains such folder, the [Cucumber.js](#) test runner is installed there.

Projects of other types or [empty](#) projects may not have a `node_modules` folder. In this case [npm](#) goes upwards in the folder tree and installs the [Cucumber.js](#) test runner in the first detected `node_modules` folder. Keep in mind that this detected `node_modules` folder may be [outside](#) your current project root.

Finally, if no `node_modules` folder is detected in the folder tree either, the folder is created right under the current project root and the [Cucumber.js](#) test runner is installed there.

In either case, make sure that the parent folder of the [Cucumber.js](#) test runner is added to the `PATH` variable. This enables you to launch the test runner from any folder.

Installing Cucumber.js as a development dependency

Because [Cucumber.js](#) is a test framework, which is of no need for those who are going to re-use your application, it is helpful to have it excluded from download for the future. This is done by marking the tool as a [development dependency](#), which actually means adding the tool in the `devDependencies` section of the `package.json` file.

With [PyCharm](#), you can have the test runner marked as a [development dependency](#) right during installation. Do one of the following:

- Run the installation from the command line in the [global](#) mode: launch the embedded Terminal (View | Tool Windows | Terminal or by hovering your mouse pointer over  in the lower left corner of [PyCharm](#) and choosing Terminal from the menu and type `npm install --dev cucumber` at the command line prompt.
- Run [NPM](#) from [PyCharm](#) using the [Node.js](#) and [NPM](#) page of the Settings dialog box.
 1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or [PyCharm](#) | Preferences for macOS, and click [Node.js](#) and [NPM](#) under Languages & Frameworks.
 2. On the [Node.js](#) and [NPM](#) page that opens, the Packages area shows all the [Node.js](#)-dependent packages that are currently installed on your computer, both at the [global](#) and at the [project](#) level. Click [+](#).
 3. In the Available Packages dialog box that opens, select the `cucumber` package.
 4. Select the Options check box and type `--dev` in the text box next to it.
 5. Optionally specify the product version and click Install Package to start installation.

After installation, the test runner is added to the `devDependencies` section of the `package.json` file.

Preparing to Use JSTestDriver Test Runner

This feature is supported in the Professional edition only.

On this page:

- [Configuring the JSTestDriver test runner](#)

Configuring the JSTestDriver test runner

You do not need to download the **JSTestDriver** framework manually. The server and the assertion framework are provided through the **JSTestDriver** plugin. The plugin is not bundled with PyCharm, but it can be installed from the **JetBrains plugin repository** as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).

This plugin does the following:

- Runs the **JSTestDriver server** that captures an opened browser to execute tests in.
- During the test creation, detects the unit testing framework the test code complies with, whereupon suggests the **Add <test framework> support intention action**, provided that the framework is recognized as a **PyCharm JavaScript library** and is thus available in the IDE.

Preparing to Use Karma Test Runner

This feature is supported in the Professional edition only.

The [Karma](#) test runner supports executing unit tests against the [Node.js](#) server. The tests themselves can use [Jasmine](#), [QUnit](#), or [Mocha](#) libraries in Karma-specific edition.

Besides running unit tests, with [Karma](#) you can measure how much of your code is covered with tests. For more details, see [Monitoring Code Coverage for JavaScript](#). To run tests with coverage, you need an additional [Karma](#)-related package.

The easiest way to install the Karma test runner is to use the [Node Package Manager \(npm\)](#), which is a part of [Node.js](#). See [Installing and Removing External Software Using Node Package Manager](#) for details.

Depending on the desired location of the Karma test runner executable file, choose one of the following methods:

- Install the test runner **globally** at the PyCharm level so it can be used in any PyCharm project.
- Install the test runner in a specific project and thus restrict its use to this project.
- Install the test runner in a project as a [development dependency](#).

In either installation mode, make sure that the parent folder of the Karma test runner is added to the `PATH` variable. This enables you to launch the test runner from any folder.

PyCharm provides user interface both for **global** and **project** installation as well as supports installation through the command line.

On this page:

- [Preparing to install the Karma test runner](#)
- [Installing Karma globally](#)
- [Installing Karma in a project](#)
- [Configuring testing frameworks in a project](#)

Preparing to install the Karma test runner

1. Download and install [Node.js](#). The runtime environment is required for two reasons:

- The Karma test runner is started through [Node.js](#).
- [NPM](#), which is a part of the runtime environment, is also the easiest way to download the Karma test runner.

If you are going to use the command line mode, make sure the path to the parent folder of the [Node.js](#) executable file and the path to the `npm` folder are added to the `PATH` variable. This enables you to launch the Karma test runner and `npm` from any folder.

2. Install and enable the [NodeJS](#) repository plugin as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).
3. Install and enable the Karma repository plugin. The plugin is not bundled with PyCharm, but it can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).

Installing Karma globally

Global installation makes a test runner available at the PyCharm level so it can be used in any PyCharm project. Moreover, during installation the parent folder of the test runner is automatically added to the `PATH` variable, which enables you to launch the test runner from any folder.

– Run the installation from the command line in the **global** mode:

1. Launch the embedded Terminal (View | Tool Windows | Terminal or by hovering your mouse pointer over  in the lower left corner of PyCharm and choosing Terminal from the menu
2. Switch to the directory where [NPM](#) is stored or define a `PATH` variable for it so it is available from any folder, see [Installing NodeJS](#).
3. Type the following command at the command line prompt:

```
npm install -g karma
```

The `-g` key makes the test runner run in the **global** mode. Because the installation is performed through [NPM](#), the Karma test runner is installed in the `npm` folder. Make sure this parent folder is added to the `PATH` variable. This enables you to launch the test runner from any folder.

For more details on the [NPM](#) operation modes, see [npm documentation](#). For more information about installing the Karma test runner, see <https://npmjs.org/package/karma>.

– Run [NPM](#) from PyCharm using the Node.js and NPM page of the Settings dialog box.

1. Open the [Settings / Preferences Dialog](#) by pressing `(Ctrl+Alt+S)` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Node.js and NPM under Languages & Frameworks.
2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the **global** and at the **project** level. Click `+`.
3. In the Available Packages dialog box that opens, select the required package to install.
4. Select the Options check box and type `-g` in the text box next to it.
5. Optionally specify the product version and click Install Package to start installation.

Installing Karma in a project

Local installation in a specific project restricts the use of a test runner to this project.

Run the installation from the command line:

Run the installation from the command line.

1. Launch the embedded Terminal (View | Tool Windows | Terminal or by hovering your mouse pointer over  in the lower left corner of PyCharm and choosing Terminal from the menu
2. Switch to the project root folder and type the following command at the command line prompt:

```
npm install karma
```

– Run **NPM** from PyCharm using the Node.js and NPM page of the Settings dialog box.

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Node.js and NPM under Languages & Frameworks.
2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the **global** and at the **project** level. Click **+**.
3. In the Available Packages dialog box that opens, select the required package.
4. Optionally specify the product version and click Install Package to start installation.

Project level installation is helpful and reliable in [template-based projects](#) of the type **Node Boilerplate** or **Node.js Express**, which already have the `node_modules` folder. The latter is important because **NPM** installs the Karma test runner in a `node_modules` folder. If your project already contains such folder, the Karma test runner is installed there.

Projects of other types or **empty** projects may not have a `node_modules` folder. In this case **npm** goes upwards in the folder tree and installs the Karma test runner in the first detected `node_modules` folder. Keep in mind that this detected `node_modules` folder may be **outside** your current project root.

Finally, if no `node_modules` folder is detected in the folder tree either, the folder is created right under the current project root and the Karma test runner is installed there.

In either case, make sure that the parent folder of the Karma test runner is added to the `PATH` variable. This enables you to launch the test runner from any folder.

Configuring testing frameworks in a project

1. Download the **Jasmine**, **QUnit**, or **Mocha** framework. The easiest way is to use the **Node.js Package Manager**:

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Node.js and NPM under Languages & Frameworks.
 2. On the [Node.js and NPM](#) page that opens, click Install in the Packages area.
 3. In the Available Packages dialog box that opens, select `karma-jasmine`, `karma-qunit`, or `karma-mocha` package and click Install Package. Close the dialog box when ready. PyCharm returns to the Node.js page, where the selected package is added to the Packages list. Click OK.
2. To enable PyCharm to resolve references to the downloaded framework and provide code completion and other types of coding assistance, [configure the framework as a PyCharm JavaScript library](#).

Preparing to Use Mocha Test Framework

This feature is supported in the Professional edition only.

On this page:

- [Introduction](#)
- [Preparing to install the Mocha test framework](#)
- [Installing Mocha globally](#)
- [Installing Mocha in a project](#)

Introduction

The [Mocha test framework](#) supports executing unit tests against the **Node.js** server. The tests themselves can use **BDD**, **TDD**, **Exports**, and **QUnit interfaces**.

The easiest way to install the Mocha test framework is to use the **Node Package Manager (npm)**, which is a part of [Node.js](#). See [Installing and Removing External Software Using Node Package Manager](#) for details.

Depending on the desired location of the Mocha test framework executable file, choose one of the following methods:

- Install the test framework **globally** at the PyCharm level so it can be used in any PyCharm project.
- Install the test framework in a specific project and thus restrict its use to this project.
- Install the test framework in a project as a [development dependency](#).

In either installation mode, make sure that the parent folder of the Mocha test framework is added to the `PATH` variable. This enables you to launch the test framework from any folder.

PyCharm provides user interface both for **global** and **project** installation as well as supports installation through the command line.

Preparing to install the Mocha test framework

1. Download and install [Node.js](#). The runtime environment is required for two reasons:

- The Mocha test framework is started through **Node.js**.
- **NPM**, which is a part of the runtime environment, is also the easiest way to download the Mocha test framework.

If you are going to use the command line mode, make sure the path to the parent folder of the **Node.js** executable file and the path to the `npm` folder are added to the `PATH` variable. This enables you to launch the Mocha test framework and `npm` from any folder.

2. Install and enable the **NodeJS** repository plugin as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).

Installing Mocha globally

Global installation makes a test framework available at the PyCharm level so it can be used in any PyCharm project. Moreover, during installation the parent folder of the test framework is automatically added to the `PATH` variable, which enables you to launch the test framework from any folder.

– Run the installation from the command line in the **global** mode:

1. Launch the embedded Terminal (View | Tool Windows | Terminal or by hovering your mouse pointer over  in the lower left corner of PyCharm and choosing Terminal from the menu
2. Switch to the directory where **NPM** is stored or define a `PATH` variable for it so it is available from any folder, see [Installing NodeJs](#).
3. Type the following command at the command line prompt:

```
npm install -g mocha
```

The `-g` key makes the test framework run in the **global** mode. Because the installation is performed through **NPM**, the Mocha test framework is installed in the `npm` folder. Make sure this parent folder is added to the `PATH` variable. This enables you to launch the test framework from any folder.

For more details on the **NPM** operation modes, see [npm documentation](#). For more information about installing the Mocha test framework, see <https://npms.org/package/mocha>.

– Run **NPM** from PyCharm using the Node.js and NPM page of the Settings dialog box.

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Node.js and NPM under Languages & Frameworks.
2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the **global** and at the **project** level. Click `+`.
3. In the Available Packages dialog box that opens, select the required package to install.
4. Select the Options check box and type `-g` in the text box next to it.
5. Optionally specify the product version and click Install Package to start installation.

Installing Mocha in a project

Local installation in a specific project restricts the use of a test framework to this project.

– Run the installation from the command line:

1. Launch the embedded Terminal (View | Tool Windows | Terminal or by hovering your mouse pointer over  in the lower left corner of PyCharm and choosing Terminal from the menu
2. Switch to the project root folder and type the following command at the command line prompt:

```
npm install mocha
```

– Run **NPM** from PyCharm using the Node.js and NPM page of the Settings dialog box.

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Node.js and NPM under Languages & Frameworks.
2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the **global** and at the **project** level. Click **+**.
3. In the Available Packages dialog box that opens, select the required package.
4. Optionally specify the product version and click Install Package to start installation.

Project level installation is helpful and reliable in [template-based projects](#) of the type **Node Boilerplate** or **Node.js Express**, which already have the `node_modules` folder. The latter is important because **NPM** installs the Mocha test framework in a `node_modules` folder. If your project already contains such folder, the Mocha test framework is installed there.

Projects of other types or **empty** projects may not have a `node_modules` folder. In this case **npm** goes upwards in the folder tree and installs the Mocha test framework in the first detected `node_modules` folder. Keep in mind that this detected `node_modules` folder may be **outside** your current project root.

Finally, if no `node_modules` folder is detected in the folder tree either, the folder is created right under the current project root and the Mocha test framework is installed there.

In either case, make sure that the parent folder of the Mocha test framework is added to the `PATH` variable. This enables you to launch the test framework from any folder.

Creating JavaScript Unit Tests

This feature is supported in the Professional edition only.

PyCharm provides integration with JavaScript unit testing frameworks thus providing rich coding assistance for writing JavaScript unit tests. To get framework-aware coding assistance, download the desired framework, configure it as a PyCharm JavaScript library, and define its visibility and scope.

The **visibility** status of a library determines whether it can be used in one project (**Project**) or can be re-used at the IDE level (**Global**).

The **scope** of a library defines the set of files and folders in which the library is considered as **library**, that is, it is write-protected, excluded from check for errors and refactoring, but only affects the completion list and highlighting.

For details, see [Configuring JavaScript Libraries](#).

To create JavaScript unit tests, perform these general steps

- Create a folder `test` at the same level as the `src` folder.
- Populate the `test` folder. For each production file, create a separate test file and name it as follows:

```
<name of production file>.<Test>.js
```

- Mark the folder where the tests are stored as **test folder**.
- If necessary, have PyCharm detect and enable the required testing framework on-the-fly.

Running JavaScript Unit Tests

This feature is supported in the Professional edition only.

JavaScript unit tests are run in a browser against a test server which actually handles the testing process.

This server allows you to capture a browser to run tests in, loads the test targets to the captured browser, controls the testing process, and exchanges data between the browser and PyCharm, so you can view test results without leaving the IDE.

The test server does not necessarily have to be on your machine, it can be launched right from PyCharm through a **test runner**. Currently, PyCharm supports integration with two **test runners**: **JSTestDriver** and [Karma](#).

Testing JavaScript with Cucumber.js

This feature is supported in the Professional edition only.

JavaScript unit tests are run in a browser against a test server which actually handles the testing process.

This server allows you to capture a browser to run tests in, loads the test targets to the captured browser, controls the testing process, and exchanges data between the browser and PyCharm, so you can view test results without leaving the IDE.

The test server does not necessarily have to be on your machine, it can be launched right from PyCharm through the [Cucumber.js](#) test runner. **Cucumber.js** runs tests that are called [features](#) and are written in the [Gherkin](#) language. Each **feature** is described in a separate file with the extension `feature`.

On this page:

- [Before you start](#)
- [Installing Cucumber.js](#)
- [Creating a Cucumber.js run configuration](#)
- [Launching tests](#)

Before you start

1. Download and install [Node.js](#). The runtime environment is required for two reasons:

- The Cucumber.js test runner is started through [Node.js](#).
- **NPM**, which is a part of the runtime environment, is also the easiest way to download the Cucumber.js test runner.

If you are going to use the command line mode, make sure the path to the parent folder of the **Node.js** executable file and the path to the `npm` folder are added to the `PATH` variable. This enables you to launch the Cucumber.js test runner and `npm` from any folder.

2. Install and enable the **NodeJS** repository plugin as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).

3. Install and enable the **Cucumber.js** and **Gherkin** plugins. The plugins are not bundled with PyCharm, but they can be installed from the **JetBrains plugin repository** as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#). Once enabled, the plugins are available at the IDE level, that is, you can use them in all your PyCharm projects.

Installing Cucumber.js

You can install **Cucumber.js** in the current project (locally), or globally, or as a development dependency.

- **Local** installation in a specific project restricts the use of the test runner to this project.
 - **Global** installation makes the test runner available at the PyCharm level so it can be used in any PyCharm project. Moreover, during installation the parent folder of the test runner is automatically added to the `PATH` variable, which enables you to launch the test runner from any folder.
 - Because Cucumber.js is a test framework, which is of no need for those who are going to re-use your application, it is helpful to have it excluded from download for the future. This is done by marking the tool as a [development dependency](#), which actually means adding the tool in the `devDependencies` section of the `package.json` file.
1. launch the embedded Terminal (View | Tool Windows | Terminal or by hovering your mouse pointer over  in the lower left corner of PyCharm and choosing Terminal from the menu and type one of the following commands at the command line prompt:
- `npm install cucumber` to install **Cucumber.js** in your project.
 - `npm install -g cucumber` to install **Cucumber.js** globally. The `-g` key makes the test runner run in the **global** mode. Because the installation is performed through **NPM**, the Cucumber.js test runner is installed in the `npm` folder. Make sure this parent folder is added to the `PATH` variable. This enables you to launch the test runner from any folder.
 - `npm install --dev cucumber` to install **Cucumber.js** as a development dependency. After installation, the test runner is added to the `devDependencies` section of the `package.json` file.

See [NPM documentation](#) for more details about [installing the Cucumber.js test runner](#) and [npm operation modes](#).

You can also install the `cucumber` package on the Node.js and NPM page of the Settings dialog box as described in [Installing and Removing External Software Using Node Package Manager](#).

Creating a Cucumber.js run configuration

1. Open the [Run/Debug Configuration](#) dialog box by doing one of the following:
 - On the main menu, choose Run | Edit Configurations.
 - Open the test file in the editor, and then choose Create <file name> on the context menu.
 - Select the test file in the Project tool window, and then choose Create <file name> on the context menu of the selection.
2. Click the Add button  on the toolbar and select the Cucumber.js configuration type. The [Run/Debug Configuration: Cucumber.js](#) dialog box opens.
3. In the Feature file or directory text box, specify the tests to run. **Cucumber.js** runs tests that are called [features](#) and are written in the [Gherkin](#) language. Each **feature** is described in a separate file with the extension `feature`.
 - To have one **feature** executed, specify the path to the corresponding `.feature` file.
 - To have a bunch of **features** executed, specify the folder where the `.feature` files to run are stored.
4. In the Executable path text box, specify the location of the **cucumber** executable file, `.cmd`, `.bat`, or other depending on the operating system used. The location depends on the installation mode, see [Installing the Cucumber.js test runner](#).
5. Optionally, specify the command line arguments to be passed to the **Cucumber.js** executable file, such as `-r (--require LIBRARY|DIR)`, `-t (--tags TAG_EXPRESSION)`, or `--coffee`. For details, see **Cucumber's** native built-in help available through the `cucumber-js --help` command.
6. Apply the changes and close the dialog box.

Launching tests

1. To launch the tests according to a run configuration, select the Cucumber.js run/debug configuration from the list on the main toolbar and click the Run button  to the right of the list.
2. The test server starts automatically without any steps from your side. View and analyze messages from the test server in the Run tool window.
3. Monitor test execution in the Test Runner tab of the Run tool window as described in [Monitoring and Managing Tests](#).

Testing JavaScript with JSTestDriver

This feature is supported in the Professional edition only.

On this page:

- [Introduction](#)
- [Configuring the JSTestDriver test runner](#)
- [Configuring a testing framework in a project](#)
- [Creating a test runner configuration file manually](#)
- [Starting the PyCharm default JSTestDriver test server](#)
- [Capturing a browser to execute tests in](#)
- [Creating a JSTestDriver run configuration](#)
- [Launching unit tests](#)
- [Monitoring code coverage](#)

Introduction

JavaScript unit tests are run in a browser against a test server which actually handles the testing process.

This server allows you to capture a browser to run tests in, loads the test targets to the captured browser, controls the testing process, and exchanges data between the browser and PyCharm, so you can view test results without leaving the IDE.

The test server does not necessarily have to be on your machine, it can be launched right from PyCharm through the **JSTestDriver test runner**.

The test server loads tests to the browser according to the [configuration files](#) that define which test and corresponding production files should be loaded and in which order. JSTestDriver server treats `*.jstd` and `*.conf` files as test runner configuration files. Create one or several configuration files in advance manually and then specify them in the [run configuration](#).

Configuring the JSTestDriver test runner

You do not need to download the **JSTestDriver** framework manually. The server and the assertion framework are provided through the **JSTestDriver** plugin. The plugin is not bundled with PyCharm, but it can be installed from the **JetBrains plugin repository** as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).

This plugin does the following:

- Runs the **JSTestDriver server** that captures an opened browser to execute tests in.
- During the test creation, detects the unit testing framework the test code complies with, whereupon suggests the **Add <test framework> support intention action**, provided that the framework is recognized as a **PyCharm JavaScript library** and is thus available in the IDE.

Configuring a testing framework in a project

1. Download the framework of your choice and [configure it as a PyCharm JavaScript library](#).
2. Do one of the following:
 - [Add the project folder to the library scope](#).
 - Enable the framework support on-the-fly [during test creation](#). Write the test as required, position the cursor at it, and press `Alt+Enter`. Then choose the **Add <test framework> support intention action** from the list.
 - To use **Jasmine**, add `jasmine-jstd-adapter` to the configuration file. Open `jsTestDriver.conf` and type the following code in it:

```
load:  
lib/jasmine/jasmine.js  
lib/jasmine-jstd-adapter/JasmineAdapter.js
```

Creating a test runner configuration file manually

A test runner configuration file defines which test and corresponding production files should be loaded and in which order. Any file with the extension `*.jstd` or `*.conf` is treated as a test runner configuration file.

1. In the Project tree, select the parent folder of the `src` (production) and the `test` folders, and choose New | File on the context menu.
2. In the New File dialog box, that opens, type a name of the configuration file with the extension `jstd` or `conf`.
3. Open the new file in the editor and specify the full path to the current folder and paths to the files to load relative to the current folder. Use wildcards to specify file name patterns. The required format is [YAML](#), for more details, see [description of test runner configuration files](#).

Starting the PyCharm default JSTestDriver test server

If you are going to use a test server running on another machine or listening to another port, start it according to the server-specific instructions and specify its URL address in the Server area of the [Run/Debug Configuration: JSTestDriver](#) dialog box.

1. Make sure the JSTestDriver [plugin is enabled](#) and at least one test runner configuration file is available in the project.
2. Open the **JSTestDriver Server** tool window by doing one of the following:
 - On the main menu, choose View | Tool Windows | JSTestDriver Server.
 - Click the Run button  on the toolbar, and then click the Start a local server link in the pop-up window that opens.

3. In the [JSTestDriver Server](#) tool window, that opens, click the Run a local server toolbar button .

To stop the server when you are through with unit testing, click the Stop the local server toolbar button .

Capturing a browser to execute tests in

1. Start the [JSTestDriver Server](#) if it is not running yet, and then switch to the JSTestDriver Server tool window.

2. To start a local browser with the Remote Console of the JSTestDriver, do one of the following:

- Click the icon that indicates the browser of your choice.
- If the browser is already opened, copy the contents of the Capture a browser using the URL read-only field and paste the URL in the address bar.

In either case, the icon that indicates the chosen browser becomes active.

You can have several browsers captured. However if you are going to debug your unit tests, you will have to appoint a browser for debugging in the Debug tab of the [Run/Debug Configuration: JSTestDriver](#) dialog box.

3. Switch to the JSTestDriver Server tool window and click the icon that indicates the browser you just opened. PyCharm displays a message informing you that it is ready for executing tests.

Creating a JSTestDriver run configuration

1. Open the [Run/Debug Configuration](#) dialog box by doing one of the following:

- On the main menu, choose Run | Edit Configurations.
- Open the test file in the editor, and then choose Create <file name> on the context menu.
- Select the test file in the Project tool window, and then choose Create <file name> on the context menu of the selection.

2. Click the Add button  on the toolbar and select the JSTestDriver configuration type.

3. In the dialog box that opens, specify the test scope, configuration parameters, and activities to perform before test execution.

4. Apply the changes and close the dialog box.

Launching unit tests

1. To launch the tests according to a run configuration, select the JSTestDriver run/debug configuration from the list on the main toolbar and click the Run button  to the right of the list.

2. To launch a single test, open the test file in the editor and click the blue arrow icon  in the gutter area next to the text to run.

Note The arrows appear only if the test framework used in your tests is associated with the project, so PyCharm can recognize the tests.

3. Monitor test execution in the Test Runner tab of the Run tool window as described in [Monitoring and Managing Tests](#).

Monitoring code coverage

With [JSTestDriver](#), you can also monitor how much of your code is [covered with tests](#). PyCharm displays this statistics in a dedicated tool window and marks covered and uncovered lines visually right in the editor. To monitor code coverage, you need to slightly update the run/debug configuration and then launch the tests in the special [Run with Coverage](#) mode.

1. Create a [JSTestDriver](#) run/debug configuration [as described above](#).

2. To have specific files excluded from coverage analysis, specify the paths to them in the Coverage tab of the [Run/Debug Configuration: JSTestDriver](#) dialog box.

3. [Start the JSTestDriver server](#) and [capture a browser](#) to run the tests in.

4. On the main toolbar, select the [JSTestDriver](#) run configuration in the Run/Debug Configurations drop-down list and click the Run with Coverage button .

5. Monitor the code coverage in the [Coverage](#) tool window.

Testing JavaScript with Karma

This feature is supported in the Professional edition only.

On this page:

- [Introduction](#)
- [Before you start](#)
- [Installing Karma](#)
- [Generating a Karma configuration file](#)
- [Creating a Karma run configuration](#)
- [Launching unit tests](#)
- [Monitoring code coverage](#)

Introduction

JavaScript unit tests are run in a browser against a test server which actually handles the testing process.

This server allows you to capture a browser to run tests in, loads the test targets to the captured browser, controls the testing process, and exchanges data between the browser and PyCharm, so you can view test results without leaving the IDE.

The test server does not necessarily have to be on your machine, it can be launched right from PyCharm through the [Karma](#) test runner.

Karma executes unit tests according to a `karma.conf.js` configuration file which is generated semi-automatically in the interactive mode.

Before you start

1. Download and install [Node.js](#). The runtime environment is required for two reasons:

- The Karma test runner is started through **Node.js**.
- **NPM**, which is a part of the runtime environment, is also the easiest way to download the Karma test runner.

If you are going to use the command line mode, make sure the path to the parent folder of the **Node.js** executable file and the path to the `npm` folder are added to the `PATH` variable. This enables you to launch the Karma test runner and `npm` from any folder.

2. Install and enable the Karma repository plugin. The plugin is not bundled with PyCharm, but it can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).

Installing Karma

You can install **Karma** in the current project (locally), or globally, or as a development dependency.

- **Local** installation in a specific project restricts the use of the test runner to this project.
- **Global** installation makes the test runner available at the PyCharm level so it can be used in any PyCharm project. Moreover, during installation the parent folder of the test runner is automatically added to the `PATH` variable, which enables you to launch the test runner from any folder.
- Because Karma is a test framework, which is of no need for those who are going to re-use your application, it is helpful to have it excluded from download for the future. This is done by marking the tool as a [development dependency](#), which actually means adding the tool in the `devDependencies` section of the `package.json` file.

See [NPM documentation](#) for more details about [installing the Karma test runner](#) and [npm operation modes](#).

You can also install the `karma` package on the Node.js and NPM page of the Settings dialog box as described in [Installing and Removing External Software Using Node Package Manager](#).

Generating a Karma configuration file

Karma executes unit tests according to a `karma.conf.js` configuration file which is generated semi-automatically in the interactive mode. The instruction below ensures successful creation of a consistent configuration file `karma.conf.js` which in its turn ensures successful execution of the tests in your project.

For more details, see <http://karma-runner.github.io/0.10/config/configuration-file.html>.

To create a Karma configuration file:

1. In the command line mode, switch to your project directory.
2. Type the following command at the command line prompt:

```
karma init
```

If **Karma** does not start, check the installation: the parent folder or the **Karma** executable file should be specified in the `PATH` variable.

3. Answer the questions to specify the following basic settings:

- The testing framework to use.
- The browsers to be captured automatically.
- The patterns that define the location of test files to be involved in testing or excluded from it, for example, `src/*.js` and `test/*.js`. For more details, see <http://karma-runner.github.io/0.10/config/files.html>.

Creating a Karma run configuration

1. Open the [Run/Debug Configuration](#) dialog box by doing one of the following:

- On the main menu, choose Run | Edit Configurations.

- Open the test file in the editor, and then choose Create <file name> on the context menu.
 - Select the test file in the Project tool window, and then choose Create <file name> on the context menu of the selection.
2. Click **+** on the toolbar and select the Karma configuration type.
 3. In the dialog box that opens, specify the location of the **Node.js** and **Karma** executable files and the path to the `karma.conf.js` configuration file.
 4. Apply the changes and close the dialog box.

Launching unit tests

1. To launch the tests according to a run configuration, select the Karma run/debug configuration from the list on the main toolbar and click the Run button  to the right of the list.
2. The Karma test server starts automatically without any steps from your side. View and analyze messages from the test server in the Karma Server tab of the Run tool window.
3. Monitor test execution in the Test Runner tab of the Run tool window as described in [Monitoring and Managing Tests](#).

Monitoring code coverage

With **Karma**, you can also monitor how much of your code is [covered with tests](#). PyCharm displays this statistics in a dedicated tool window and marks covered and uncovered lines visually right in the editor.

Coverage for tests run in **Karma** is calculated by the [Istanbul](#) coverage tool which is shipped within the `karma-coverage` package. Therefore to monitor coverage with **Karma** you need to install the `karma-coverage` package in the same folder with **Karma** and then update the `karma.conf.js` configuration file manually.

1. Install the `karma-coverage` package: open the Terminal tool window (View | Tool Windows | Terminal) and at the command line prompt type `npm install karma karma-coverage --save-dev`. Alternatively, you can install the `karma-coverage` package on the [Node.js and NPM page](#) as described in [Installing and Removing External Software Using Node Package Manager](#).

The `karma-coverage` package should be installed in the same folder with **Karma** (`karma (library home)` package), no matter whether **Karma** is installed globally or in your project. Keep this restriction in mind when installing `karma-coverage` manually in the command line mode. If you are installing `karma-coverage` through the **Node Package Manager** interface, the proper installation folder will be chosen automatically.

2. Open the `karma.conf.js` configuration file and add the following information to it manually:
 1. Locate the `reporters` definition and add `coverage` to the list of values in the following format:

```
reporters: ['progress', 'coverage'],
```

2. Add a `preprocessors` definition and specify the coverage scope according to the following format:

```
preprocessors: {
  '**/*.js': ['coverage']
}
```

3. Create a **Karma** run/debug configuration [as described above](#).
4. On the main toolbar, select the **Karma** run configuration in the Run/Debug Configurations drop-down list and click the Run with Coverage button .
5. Monitor the code coverage in the [Coverage](#) tool window. Note that when the tests are executed on `Karma` the Coverage tool window does not contain the Generate Coverage Report toolbar button  because a coverage report is actually generated on the disk every time **Karma** tests are run. The format of a coverage report can be configured in the configuration file, for example:

```
// karma.conf.js
module.exports = function(config) {

  config.set({ ...

  // optionally, configure the reporter

  coverageReporter: { type : 'html', dir : 'coverage/' }

  ...

});};
```

The following `type` values are acceptable:

- `html` produces a bunch of HTML files with annotated source code
- `lcovonly` produces an `lcov.info` file
- `lcov` produces HTML + `.lcov` files. This format is applied by default.
- `cobertura` produces a `cobertura-coverage.xml` file for easy Hudson integration
- `text-summary` produces a compact text summary of coverage, typically to the console
- `text` produces a detailed text table with coverage for all files

Testing JavaScript with Mocha

This feature is supported in the Professional edition only.

The [Mocha test framework](#) supports executing unit tests against the **Node.js** server. The tests themselves can use **BDD**, **TDD**, **Exports**, and **QUnit interfaces**.

On this page:

- [Before you start](#)
- [Installing Mocha](#)
- [Quick test launch](#)
- [Creating a Mocha run/debug configuration](#)
- [Launching tests via a run/debug configuration](#)

Before you start

1. Make sure the [Node.js](#) runtime environment is installed on your computer.
2. Make sure the **NodeJS** repository plugin is installed and enabled. The plugin is not bundled with PyCharm, but it can be installed from the **JetBrains plugin repository** as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).

Installing Mocha

Open the built-in PyCharm Terminal (press `Alt+F12` or choose View | Tool Windows | Terminal on the main menu) and at the command line prompt type `npm install --global mocha` for global installation or `npm install --save-dev mocha` to install **Mocha** as a **development dependency**.

See [NPM documentation](#) for more details about [installing the Mocha test runner](#) and [npm operation modes](#). You can also install the `mocha` package on the Node.js and NPM page of the Settings dialog box as described in [Installing and Removing External Software Using Node Package Manager](#).

Quick test launch

You can run and debug **Mocha** tests and suites right from the editor: click  or  in the left gutter and choose Run <test_file_name> or Debug <test_file_name> from the pop-up list.



You can also see whether a test has passed or failed right in the editor, thanks to the **test status** icons  in the left gutter.

Creating a Mocha run/debug configuration

1. Open the [Run/Debug Configuration](#) dialog box by doing one of the following:
 - On the main menu, choose Run | Edit Configurations.
 - Open the test file in the editor, and then choose Create <file name> on the context menu.
 - Select the test file in the Project tool window, and then choose Create <file name> on the context menu of the selection.
- Click the Add button  on the toolbar and select the Mocha configuration type. The [Run/Debug Configuration: Mocha](#) dialog box opens.
2. Specify the [Node interpreter](#) to use and the location of the **mocha** package.
3. Specify the [working directory](#) of the application. All references in test scripts will be resolved relative to this folder, unless such references use full paths. By default, the field shows the **project root folder**. To change this predefined setting, choose the desired folder from the drop-down list, or type the path manually, or click the Browse button  and select the location in the dialog box, that opens.
4. Specify the tests to run:
 - All in directory: choose this option to run all the tests from files stored in a folder. In the Test directory field, specify the folder with the tests. To have PyCharm look for tests in the subfolders under the specified directory, select the Include subdirectories check box.
 - File pattern: choose this option to have PyCharm look for tests in all the files with the names that match a certain mask and specify the mask in the Test file pattern field.
 - Test file: choose this option to run only the tests from one file and specify the path to this file in the Test file field.
 - Suite: choose this option to run individual suites from a test file. In the Test file field, specify the file where the required suites are defined. Click the Suite name field and configure a list of suites to run. To add a suite to the list, click  and type the name of the required suite. To remove a suite, select it in the list and click .
 - Test: choose this option to run individual tests from a test file. In the Test file field, specify the file where the required tests are defined. Click the Test name field and configure a list of tests to run. To add a test to the list, click  and type the name of the required test. To remove a test, select it in the list and click .
5. Choose the [interface](#) used in the test to run.

Launching tests via a run/debug configuration

1. To launch the tests according to a run configuration, select the Mocha run/debug configuration from the list on the main toolbar and click the Run button  to the right of the list.

2. The test server starts automatically without any steps from your side. View and analyze messages from the test server in the Run tool window.
3. Monitor test execution in the Test Runner tab of the Run tool window as described in [Monitoring and Managing Tests](#).

[Code Coverage](#) shows how much of your code is covered with tests and marks covered and uncovered lines visually right in the editor.

In this section:

- [Prerequisites](#)
- [Enabling running tests with coverage in Karma](#)
- [Measuring code coverage for tests executed on JSTestDriver](#)
- [Measuring code coverage for tests executed on Karma](#)

Prerequisites

1. The **JSTestDriver** or **Karma** test runner is installed and configured.
2. Additional third-party testing frameworks, for example, [Jasmine](#), [QUnit](#), or [Mocha](#), are downloaded and integrated with PyCharm.
3. A runner-specific configuration file is generated or written manually.

For more details, see [Testing JavaScript with Karma](#) or [Testing JavaScript with JSTestDriver](#) depending on your choice.

Enabling running tests with coverage in Karma

Coverage for tests run in **Karma** is calculated by the [istanbul](#) coverage tool. The tool is shipped within the `karma-coverage` package. Therefore running tests with coverage in **Karma** requires that you download the `karma-coverage` package and manually update the `karma.conf.js` configuration file.

The `karma-coverage` package should be installed in the same folder with **Karma** (`karma (library home)` package), no matter whether **Karma** is installed globally or in your project. Keep this restriction in mind when installing `karma-coverage` manually in the command line mode. If you are installing `karma-coverage` through the **Node Package Manager** interface, the proper installation folder will be chosen automatically.

1. Install the `karma-coverage` package through the **Node Package Manager (npm)**.
 1. Open the **Settings / Preferences Dialog** by pressing `Ctrl+Alt+S` or by choosing `File | Settings for Windows and Linux or PyCharm | Preferences for macOS`, and click `Node.js and NPM` under `Languages & Frameworks`.
 2. On the **Node.js and NPM** page that opens, click `Install` in the `Packages` area.
 3. In the `Available Packages` dialog box that opens, select the `karma-coverage` package, click `Install Package`, and close the dialog box when ready. PyCharm returns to the `Node.js` page, where the `karma-coverage` package is added to the `Packages` list. Click `OK`.

For details, see [Installing and Removing External Software Using Node Package Manager](#).

2. Open the `karma.conf.js` configuration file and add the following information to it manually:
 1. Locate the `reporters` definition and add `coverage` to the list of values in the following format:

```
reporters: ['progress', 'coverage'],
```

2. Add a `preprocessors` definition and specify the coverage scope according to the following format:

```
preprocessors: {  
  '**/*.js': ['coverage']  
}
```

Measuring code coverage for tests executed on JSTestDriver

1. Prepare tests manually or [have tests generated](#) by PyCharm.
2. Create a JSTestDriver configuration file `*.jstd` or `*.conf`. For details, see [JSTestDriver Configuration file](#).
3. [Create a run configuration](#) of the type `JSTestDriver`.
To have specific files excluded from coverage analysis, specify the paths to them in the `Coverage` tab of the [Run/Debug Configuration: JSTestDriver](#) dialog box.
4. [Start the JSTestDriver server](#) and [capture a browser](#) to run the tests in.
5. On the main toolbar, select the **JSTestDriver** run configuration in the `Run/Debug Configurations` drop-down list and click the `Run with Coverage` button .
6. Monitor the code coverage in the [Coverage](#) tool window.

Measuring code coverage for tests executed on Karma

1. Prepare tests manually or [have tests generated](#) by PyCharm.
2. Create a Karma configuration file `karma.conf`. For details, see [Testing JavaScript with Karma](#).
3. [Create a run configuration](#) of the type `Karma`.
4. On the main toolbar, select the **Karma** run configuration in the `Run/Debug Configurations` drop-down list and click the `Run with Coverage` button .
5. Monitor the code coverage in the [Coverage](#) tool window. Note that when the tests are executed on `Karma` the `Coverage` tool window does not contain the `Generate Coverage Report` toolbar button  because a coverage report is actually generated on the disk every time **Karma** tests

are run. The format of a coverage report can be configured in the configuration file, for example:

```
// karma.conf.js
module.exports = function(config) {

  config.set({ ...

  // optionally, configure the reporter

  coverageReporter: { type : 'html', dir : 'coverage/' }

  ...

  });};
```

The following `type` values are acceptable:

- `html` produces a bunch of HTML files with annotated source code
- `lcovonly` produces an `lcov.info` file
- `lcov` produces HTML + `.lcov` files. This format is applied by default.
- `cobertura` produces a `cobertura-coverage.xml` file for easy Hudson integration
- `text-summary` produces a compact text summary of coverage, typically to the console
- `text` produces a detailed text table with coverage for all files

Debugging JavaScript Unit Tests

Besides simply executing unit tests on the **JSTestDriver** or **Karma** without the ability to analyze the problems that arise, PyCharm provides the ability to debug unit tests just like JavaScript source code.

On this page:

- [Prerequisites](#)
- [Debugging unit tests executed on JSTestDriver](#)
- [Debugging unit tests executed on Karma](#)

Prerequisites

1. The **JSTestDriver** or **Karma** test runner is installed and configured.
2. Additional third-party testing frameworks, for example, [Jasmine](#), [QUnit](#), or [Mocha](#), are downloaded and integrated with PyCharm.
3. A runner-specific configuration file is generated or written manually.

For more details, see [Testing JavaScript with Karma](#) or [Testing JavaScript with JSTestDriver](#) depending on your choice.

Debugging unit tests executed on JSTestDriver

1. [Prepare the unit tests](#) to run.
2. In the unit tests, set the [breakpoints](#) for PyCharm to postpone test execution at.
3. Create a JSTestDriver configuration file `*.jstd` or `*.conf`. For details, see [JSTestDriver Configuration file](#).
4. [Start a PyCharm default JSTestDriver test server](#).
5. [Capture a browser to execute the tests in](#).
6. [Create a JSTestDriver run configuration](#). If you have captured two browsers, specify the browser for debugging the tests: in the Debug tab of the [Run/Debug Configuration: JSTestDriver](#) dialog box, choose the relevant browser from the Debug drop-down list. The available options are Chrome and Dartium.
7. Choose Run | Debug on the main menu, or click Debug button  on the toolbar, or press `Shift+F9`.
8. In the [Debug](#) tool window that opens, proceed as usual: [step through the program](#), [stop and resume](#) program execution, [examine it when suspended](#), etc.

Debugging unit tests executed on Karma

1. Prepare tests manually or [have tests generated](#) by PyCharm.
2. Create a Karma configuration file `karma.conf`. For details, see [Testing JavaScript with Karma](#).
3. [Create a run configuration](#) of the type **Karma**.
4. On the main toolbar, select the **Karma** run configuration in the Run/Debug Configurations drop-down list and click Debug button  on the toolbar, or press `Shift+F9`.
5. In the [Debug](#) tool window that opens, proceed as usual: [step through the program](#), [stop and resume](#) program execution, [examine it when suspended](#), etc.

Running JavaScript in Browser

In this section:

– [Running JavaScript](#)

To run a file with JavaScript from PyCharm

No [run configuration](#) is required for launching applications with injected JavaScript from PyCharm. You just open the HTML page with the reference to the required script in the browser and that's it.

This method works if you do not have a **default deployment configuration**. To check that, choose Tools | Deployment | Configuration. In the left-hand pane of the Deployment dialog that opens, the default configuration is shown in bold. To remove this status, select the configuration and click the Use as Default toolbar button , see [Creating a Remote Server Configuration](#) for details.

1. In the editor, open the HTML file with the JavaScript reference. This HTML file does not necessarily have to be the one that implements the starting page of the application.
2. Do one of the following:
 - Choose View | Preview file in on the main menu or press `Alt+F2`. Then select the desired browser from the pop-up menu.
 - Hover your mouse pointer over the code to show the browser icons bar: . Click the icon that indicates the desired browser.

Debugging JavaScript in Firefox

With PyCharm, you can debug your client-side JavaScript in Firefox, version 36 and higher, using the Firefox remote debugging functionality. However it is strongly recommended that you use **Chrome** or any other browser of the **Chrome** family.

On this page:

- [Enabling debugging in Firefox](#)
- [Debugging an application](#)
- [Example](#)

Enabling debugging in Firefox

With PyCharm, you can debug your client-side JavaScript in Firefox, version 36 and higher, using the Firefox remote debugging functionality.

To enable Firefox remote debugging:

1. Open your **Firefox** browser, then open Tools | Developer | Toggle Tools.
2. In the Development Tools pane that opens, click Toolbox Options button  on the toolbar and select the Enable remote debugging check box under Advanced Settings.



Advanced settings

- Disable Cache (when toolbox is open)
- Disable JavaScript *
- Enable chrome and add-on debugging
- Enable remote debugging

* Current session only, reloads the page

3. In the console at the bottom of the browser (the console appeared when you opened the Developer toolbar), type `listen <port number>`.



You can set any port number, however it is recommended that you use 6000 and higher. Later you will specify this port number in the run configuration.

Debugging an application

You can debug an application running on the PyCharm built-in web server on an external server.

To start debugging:

1. Set the **breakpoints** in the JavaScript code, as required.
2. Create a debug configuration of the type **Firefox Remote**:

Choose Run | Edit Configuration on the main menu, click the Add New Configuration button  on the toolbar, and select Firefox Remote from the pop-up list.

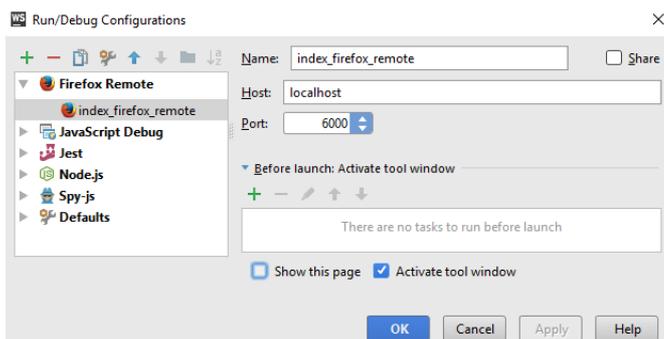
3. In the **Run/Debug Configuration: Firefox Remote** dialog box that opens, type `localhost` in the Host field. In the Port field, type the port that you specified when you **enabled remote debugging in Firefox**. The default value is `6000`.
4. Open your application in Firefox. The browser shows the application after the code execution, that is, the breakpoints you set have no effect yet.
5. Choose the newly created configuration in the Select run/debug configuration drop-down list on the toolbar and click the Debug toolbar button .
6. In the dialog box that opens, click OK to allow incoming connection from the browser.
7. Refresh the application page in the browser: the page shows the results of code execution up to the first breakpoint.
8. In the Debug tool window, proceed as usual: [step through the program](#), [stop and resume](#) program execution, [examine it when suspended](#), [view actual HTML DOM](#), etc.

Example

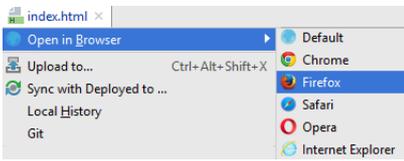
Suppose you have a simple application that consists of two files: `index.html` and `index.js` file, where `index.html` references `index.js`. This example shows how you can debug the application when it is running on the PyCharm built-in server.

To start debugging:

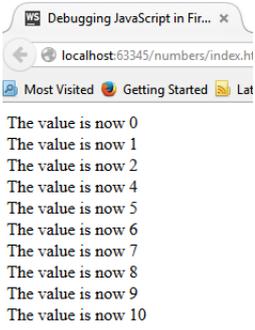
1. Set the breakpoints in `index.js`.
2. Create a **FireFox Remote** debug configuration, type `localhost` and `6000` in the Host and Port fields respectively.



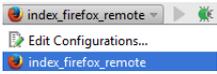
3. Open `index.html` in the editor, choose Open in browser on the context menu, and then choose Firefox from the list:



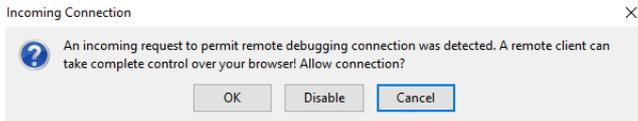
The browser opens showing the application running on the PyCharm port (currently 63345):



4. Select the `index_firefox_remote` configuration from the drop-down list on the toolbar and click the Debug toolbar button :



Click OK in the Incoming Connection dialog box that opens:



Refresh the application page in the browser.

Debugging JavaScript in Chrome

PyCharm provides a built-in debugger for your client-side JavaScript code that works with Chrome. The video and the instructions below walk you through the basic steps to get started with this debugger.

Before you start, install the [JetBrains IDE Support](#) Chrome extension. Find more about that and about additional configuration of the debugger in [Configuring JavaScript Debugger and JetBrains Chrome Extension](#).

On this page:

- [Debugging an application running on the built-in server](#)
 - [Example](#)
- [Debugging client-side JavaScript running on an external web server](#)
- [Debugging asynchronous code](#)
- [Debugging workers](#)

Debugging an application running on the built-in server

PyCharm has a built-in web server that can be used to preview and debug your application. This server is always running and does not require any manual configuration. All the project files are served on the built-in server with the root URL `http://localhost:<built-in server port>/<project root>`, with respect to the project structure.

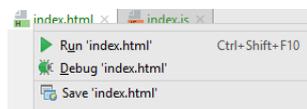
To start debugging:

1. Set the [breakpoints](#) in the JavaScript code, as required.
2. Open the HTML file that references the JavaScript to debug or select the HTML file in the [Project view](#).
3. On the context menu of the editor or the selection, choose `Debug <HTML_file_name>`. PyCharm generates a debug configuration and starts a debugging session through it. The file opens in the default browser, and the [Debug tool window](#) appears.
4. In the Debug tool window, proceed as usual: [step through the program](#), [stop and resume](#) program execution, [examine it when suspended](#), [view actual HTML DOM](#), etc.

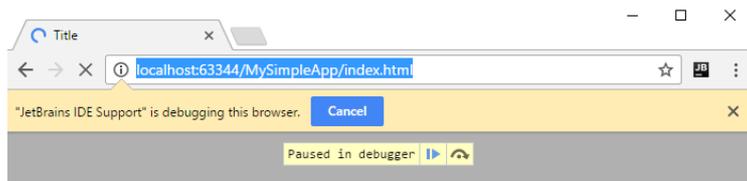
Tip To save the automatically generated configuration for further re-use, choose `Save <HTML_file_name>` on the context menu after the debugging session is over.

Example

Suppose you have a simple application that consists of an `index.html` file and a `MyJavaScript.js` file, where `index.html` references `MyJavaScript.js`. To start debugging this application using the built-in server, open `index.html` in the editor and choose `Debug 'index.html'` on the context menu:



PyCharm creates a [run/debug configuration](#) automatically, and a debugging session starts:



To restart the new run/debug configuration, click  in the upper right-hand corner of the PyCharm window or choose `Run | Debug` on the main menu:



Debugging client-side JavaScript running on an external web server

Often you may want to debug client-side JavaScript running on an external development web server, e.g. powered by Node.js.

To start debugging:

1. Set the [breakpoints](#) in the JavaScript code, as required.
2. Run the application in the [development mode](#). Often you need to run `npm start` for that. When the development server is ready, copy the URL address at which the application is running in the browser - you will need to specify this URL address in the run/debug configuration.

3. Create a debug configuration of the type JavaScript Debug:

Choose Run | Edit Configuration on the main menu, click the Add New Configuration button  on the toolbar, and select JavaScript Debug from the pop-up list.

- In the dialog box that opens, specify the URL address at which the application is running. This URL can be copied from the address bar of your browser as described in Step 2 above. Click OK to save the configuration settings.
- Choose the newly created configuration in the Select run/debug configuration drop-down list on the toolbar and click the Debug toolbar button . The URL address specified in the run configuration opens in the chosen browser and the [Debug tool window](#) appears.
- In the Debug tool window, proceed as usual: [step through the program](#), [stop and resume](#) program execution, [examine it when suspended](#), [view actual HTML DOM](#), etc.

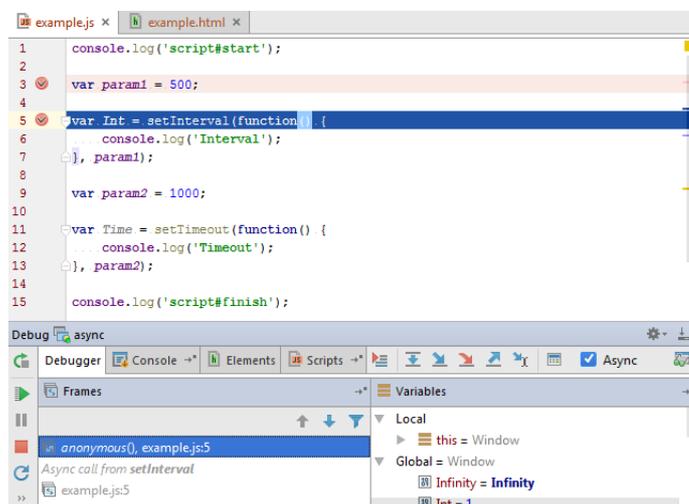
See [Debugging React Applications](#) and [Debugging Angular Applications](#) for examples.

Debugging asynchronous code

PyCharm supports debugging asynchronous client-side JavaScript code, currently this functionality is available only in Chrome. The asynchronous debugging mode is toggled through the Async check box on the tool bar of the Debugger tab in the `Debug` tool window. The check box is displayed only during a JavaScript debugging session.

- When this check box is selected, PyCharm recognizes breakpoints inside asynchronous code and stops at them and lets you step into asynchronous code. As soon as a breakpoint inside an asynchronous function is hit or you step into asynchronous code, a new element `Async call from <caller>` is added in the Frames pane of the Debugger tab. PyCharm displays a full call stack, including the caller and the entire way to the beginning of the asynchronous actions.
- When the check box is cleared, PyCharm does not recognize and therefore skips breakpoints in the asynchronous code and does not allow you to step into it.

The image below shows an example of a JavaScript debugging session.



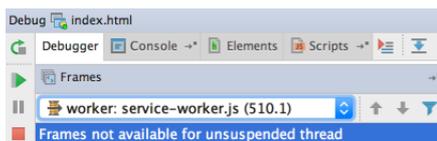
- With the Async check box selected, the debugger will stop at line3(breakpoint), then at line5(breakpoint). On clicking Step into, the debugger will stop at line5 (on `function`), then will move to line6.
- With the Async check box cleared, the debugger will stop at line3(breakpoint), then at line5(breakpoint). On clicking Step into, the debugger will move to line9.

Debugging workers

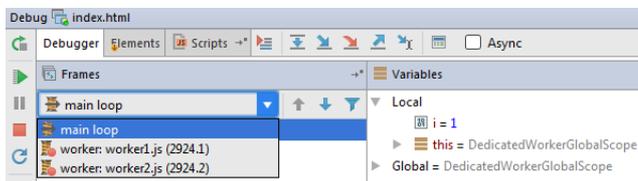
PyCharm supports debugging [Service Workers](#) and [Web Workers](#). PyCharm recognizes breakpoints in each **worker** and shows the debug data for it as a separate thread in the `Frame` pane on the `Debugger` tab of the `Debug Tool Window`.

Note that PyCharm can debug only [dedicated workers](#), debugging for [shared workers](#) is currently not supported.

- Set the breakpoints in the `Workers` to debug.
- If you are using `Service Workers`, make sure the Allow unsigned requests check box on the `Debugger` page is selected. Otherwise your `service workers` may be unavailable during a debug session:



- Create a debug configuration of the type `JavaScript Debug` as described above in [Debugging client-side JavaScript running on an external web server](#).
- Choose the newly created configuration in the Select run/debug configuration drop-down list on the tool bar and click the Debug toolbar button . The HTML file specified in the run configuration opens in the chosen browser and the `Debug tool window` opens with the Frames drop-down list showing all the `Workers`:



To examine the data (variables, watches, etc.) for a **Worker**, select its thread in the list and view its data in the [Variables](#) and [Debug Tool Window. Watches](#) panes. When you select another **Worker**, the contents of the panes are updated accordingly.

Minifying JavaScript

The term **minification** or **compression** in the context of JavaScript means removing all unnecessary characters, such as **spaces**, **new lines**, **comments** without affecting the functionality of the source code.

These characters facilitate working with the code at the development and debugging stage by improving the code readability. However at the production stage these characters become extraneous: being insignificant for code execution, they increase the size of code to be transferred. Therefore it is considered good practice to remove them before deployment.

PyCharm supports integration with the following JavaScript minification tools:

- [YUI Compressor](#)
- [UglifyJS](#)
- [Closure Compiler](#)

In PyCharm, minifier configurations are called **File Watchers**. For each supported minifier, PyCharm provides a predefined **File Watcher** template. Predefined **File Watcher** templates are available at the PyCharm level. To run a minifier against your project files, you need to create a project-specific **File Watcher** based on the relevant template, at least, specify the path to the minifier to use on your machine.

On this page:

- [Installing and configuring the Closure Compiler minification tool](#)
- [Installing and configuring the YUI Compressor JS or UglifyJS minification tool](#)
- [Creating a file watcher](#)
- [Minifying the code](#)

Installing and configuring the Closure Compiler minification tool

1. Download and install the [Node.js](#) runtime environment because the JavaScript minifier is started through **Node.js**.
2. Install and enable the **NodeJS** repository plugin as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).
3. Make sure the **File Watchers** plugin is enabled. The plugin is bundled with PyCharm and activated by default. If it is not, enable the plugin. See [Enabling and Disabling Plugins](#) for details.
4. Download the compiled tool at <http://dl.google.com/closure-compiler/compiler-latest.zip>.
5. Extract the archive to the folder under the **Node.js**. This ensures that **Node.js**, which is required for starting the tool, will be specified in the path to it.

Installing and configuring the YUI Compressor JS or UglifyJS minification tool

1. Download and install [Node.js](#). The runtime environment is required for two reasons:
 - The JavaScript minifier is started through **Node.js**.
 - **NPM**, which is a part of the runtime environment, is also the easiest way to download the JavaScript minifier.

If you are going to use the command line mode, make sure the path to the parent folder of the **Node.js** executable file and the path to the `npm` folder are added to the `PATH` variable. This enables you to launch the JavaScript minifier and `npm` from any folder.

2. Install and enable the **NodeJS** repository plugin as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).
3. Make sure the **File Watchers** plugin is enabled. The plugin is bundled with PyCharm and activated by default. If it is not, enable the plugin. See [Enabling and Disabling Plugins](#) for details.
4. Download and install the JavaScript minification tool. The easiest way is to use the **Node Package Manager (npm)**, which is a part of [Node.js](#).
 1. Switch to the directory where the **Node Package Manager (npm)** is stored or define a `path` variable for it so it is available from any folder.
 2. Depending on the tool you want to use, type one of the following commands at the command line prompt:
 - To have the **YUI Compressor JS** installed, type:

```
npm install yuicompressor
```

- To have the **UglifyJS** installed, type:

```
npm install uglify-js
```

If you use the **Node Package Manager (npm)**, the tools are installed under **Node.js** so **Node.js**, which is required for starting the tool, will be specified in the path to it.

Creating a file watcher

PyCharm provides a common procedure and user interface for creating **File Watchers** of all types. The only difference is in the predefined templates you choose in each case.

1. To start creating a **File Watcher**, open the Settings/Preferences dialog box by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS on the main menu, and then click File Watchers under the Tools node. The [File Watchers page](#) that opens, shows the list of **File Watchers** that are already configured in the project.
2. Click the Add button `+` or press `Alt+Insert`. Depending on the tool you are going to use, choose the appropriate predefined template from the pop-up list:
 - Closure Compiler

- UglifyJS
- YUI Compressor JS

3. In the Program text box, specify the path to the compiler executable file or archive depending on the chosen predefined template:

- `compiler.jar` for Closure Compiler
- `yuicompressor-<version>.jar` for YUI Compressor JS
- `uglifyjs.cmd` for UglifyJS

If you installed the tool through the **Node Package Manager**, PyCharm locates the required file itself and fills in the field automatically. Otherwise, type the path manually or click the Browse button  and choose the file location in the dialog box that opens.

4. Proceed as described on page [Using File Watchers](#).

Minifying the code

When a **Minification File Watcher** is enabled (see [Enabling and disabling File Watchers](#)), minification starts automatically as soon as a file to which compilation is applicable is changed or saved, see [Configuring the behaviour of the File Watcher](#).

PyCharm creates a separate file with the generated output. The file has the name of the source **JavaScript** file and the extension `min.js`. The location of the generated file is defined in the Output paths to refresh text box of the [New Watcher dialog](#). However, in the Project Tree, it is shown under the source **JavaScript** file which is now displayed as a node.

Using AngularJS

PyCharm provides integration with the [AngularJS framework](#) also known as **Angular 1**. This support involves:

- **AngularJS**-aware code completion for `ng` directives (also including custom directives), controller and application names, and code insights for data bindings inside curly-brace expressions `{{}}`.
- **AngularJS**-specific navigation:
 - Between the name of a controller in HTML and its definition in JavaScript.
 - Between `ngView` or `&routeProvider` and the template.
 - **Go To Symbol** navigation for entities.
- A collection of **AngularJS** built-in live templates.
- Quick documentation look-up by pressing `Ctrl+Q`, see [Viewing Inline Documentation](#) for details.
- [AngularJS ui-router](#) diagram.

Before you start

1. Install the [Node.js](#) runtime environment.
2. Make sure the **AngularJS** plugin is installed and enabled. The **AngularJS** plugin is bundled with PyCharm and activated by default. If the plugin is not activated, enable it on the [Plugins](#) page of the [Settings / Preferences Dialog](#) as described in [Enabling and Disabling Plugins](#). The **NodeJS** plugin is not bundled with PyCharm, but it can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).

Introduction

You can get **AngularJS** support in a PyCharm project in the following ways:

- [Open an existing project with AngularJS sources](#).
- [Generate an AngularJS-specific project stub](#) from the PyCharm Welcome Screen. PyCharm generates the **AngularJS**-specific project structure with all the required configuration files based on the [AngularJS seed project](#)
- [Create an empty project](#), and then install **AngularJS** in it either manually, by downloading the **AngularJS** framework, or using the **Bower** package manager. [If you choose manual installation](#), you will need to configure **AngularJS** as a PyCharm JavaScript library. [If you use Bower](#), PyCharm will do this configuration automatically, without any steps from your side.

Creating an AngularJS project using a seed project

1. Choose File | New Project on the main menu or click the Create New Project button on the Welcome screen. The [New Project Dialog](#) dialog box opens.
2. In the left-hand pane, choose AngularJS.
3. In the right-hand pane, in the Location text box, specify the path to the project folder where the project-related files will be stored.
4. When you click Create, PyCharm generates the **AngularJS**-specific project structure with all the required configuration files based on the [AngularJS seed project](#)
5. Download the **AngularJS** dependencies that contain the **AngularJS** code and the tools that support development and testing: launch the embedded Terminal (View | Tool Windows | Terminal or by hovering your mouse pointer over  in the lower left corner of PyCharm and choosing Terminal from the menu and type `npm install` at the command line prompt. Learn more about the installation of dependencies in the [Install Dependencies](#) section of the `readme.md` file.

Configuring AngularJS support in a project

If you already have **AngularJS** sources in your project (for example, in the `bower_components` folder), just open your project and start working. If these sources are **excluded from project**, then you only need to configure **AngularJS** as an [External JavaScript library](#).

If you do not have any previously downloaded **AngularJS** sources, [create an empty project](#), and then install **AngularJS** in it either [manually, by downloading the AngularJS framework](#), or [using the Bower package manager](#).

Creating an empty PyCharm project

1. Choose File | New Project on the main menu or click the Create New Project button on the Welcome screen. The [New Project Dialog](#) dialog box opens.
2. In the left-hand pane, choose Empty Project.
3. In the Location text box, specify the path to the project folder where the project-related files will be stored.
4. When you click Create, PyCharm creates and opens an **empty** project.

Configuring AngularJS coding assistance in an empty project manually

1. Download the **AngularJS** framework at <http://angularjs.org/>.
2. Open the empty project where you will use **AngularJS**.
3. Configure **AngularJS** as a PyCharm JavaScript library, to let PyCharm recognize **AngularJS**-specific structures and provide full coding assistance:
 1. [Open the Settings dialog box](#), and click [JavaScript Libraries](#).
 2. In the Libraries area, click the Add button.
 3. In the [New Library](#) dialog box that opens, specify the name of the library.
 4. Click the Add button  next to the list of library files and choose Attach Files or Attach Directory on the context menu, depending of whether you need separate files or an entire folder.
 5. Select the `Angular.js` or `Angular.min.js`, or an entire directory in the dialog box that opens. PyCharm returns to the New Library dialog box where the Name read-only field shows the name of the selected files or folder.
 6. In the Type field, specify which version you have downloaded and are going to add.

- If you added `Angular.js`, choose Debug. This version is helpful in the development environment, especially for debugging.
- If you added the [minified](#) `Angular.min.js`, choose Release. This version is helpful in the production environment because the file size is significantly smaller.

Learn more at [Configuring JavaScript Libraries](#).

Installing AngularJS in an empty project through Bower

1. Open the empty project where you will use **AngularJS**.
2. Download, install, and configure [Node.js](#) as described in [Configuring Node.js Interpreters](#).
3. Install [Bower](#) as described in [Using Bower Package Manager](#).
4. Launch the embedded Terminal (View | Tool Windows | Terminal or by hovering your mouse pointer over  in the lower left corner of PyCharm and choosing Terminal from the menu and type `bower install angular` at the command line prompt to install the package in the current project.

You can also install the `angular` package on the [Bower](#) page of the Settings dialog as described in [Installing and Removing Bower Packages](#).

Using AngularJS Router state diagrams

You can see a diagram illustrating the relations between views, states, and templates in **AngularJS** applications that use [ui-router](#).

To generate and view a diagram, open the desired file in the editor, and then choose Diagrams | Show AngularJS ui-router State Diagram on the context menu. PyCharm generates a diagram and shows it in a separate editor tab.

To navigate from an element in the diagram to the code that implements this element, select it and choose Jump to Source on the context menu.

Using Angular

PyCharm provides integration with the [Angular framework](#) also known as **Angular 2**.

On this page:

- [Before you start](#)
- [Introduction](#)
- [Installing Angular CLI](#)
- [Generating an Angular application stub using Angular CLI](#)
- [Creating an empty PyCharm project](#)
- [Installing Angular in an empty project through NPM](#)
- [Generating Angular structures](#)
- [Using Angular language service](#)

Before you start

1. Download, install, and configure [Node.js](#) as described in [Configuring Node.js Interpreters](#).
2. Make sure the **AngularJS** plugin is installed and enabled. The **AngularJS** plugin is bundled with PyCharm and activated by default. If the plugin is not activated, enable it on the [Plugins](#) page of the [Settings / Preferences Dialog](#) as described in [Enabling and Disabling Plugins](#). The **NodeJS** plugin is not bundled with PyCharm, but it can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).

Introduction

You can get **Angular** support in a PyCharm project in the following ways:

- **Open an existing project with Angular sources.** If you already have **Angular** sources in your project (for example, in the `node_modules` folder), just open your project and start working.
- **Install Angular CLI** and generate an **Angular-specific project stub using Angular CLI**. PyCharm generates the **Angular**-specific project structure with all the required configuration files based on the [AngularJS seed project](#)
- **Create an empty project**, and then install **Angular** in it using the **npm** package manager.

Installing Angular CLI

To generate an **Angular** project in PyCharm, you need to install the `angular-cli` package **globally** so it can be used in any PyCharm project. Launch the embedded Terminal (View | Tool Windows | Terminal or by hovering your mouse pointer over  in the lower left corner of PyCharm and choosing Terminal from the menu and type `npm install -g @angular/cli` at the command line prompt.

You can also install the package on the [Node.js and NPM](#) as described in [Installing and Removing External Software Using Node Package Manager](#).

Generating an Angular application stub using Angular CLI

1. Choose File | New Project on the main menu or click the Create New Project button on the Welcome screen. The [New Project Dialog](#) dialog box opens.
2. In the left-hand pane, choose Angular CLI.
3. In the right-hand pane:
 1. In the Location text box, specify the path to the project folder where the project-related files will be stored.
 2. In the Node Interpreter field, specify the NodeJS interpreter to use. Choose a configured interpreter from the drop-down list or choose Add to configure a new one, see [Configuring Node.js Interpreters](#)
 3. In the Angular CLI field, specify the path to the `angular-cli` package.
4. When you click Create, PyCharm generates the **Angular**-specific project structure with all the required configuration files based on the [AngularJS seed project](#)

Creating an empty PyCharm project

1. Choose File | New Project on the main menu or click the Create New Project button on the Welcome screen. The [New Project Dialog](#) dialog box opens.
2. In the left-hand pane, choose Empty Project.
3. In the Location text box, specify the path to the project folder where the project-related files will be stored.
4. When you click Create, PyCharm creates and opens an **empty** project.

Installing Angular in an empty project through NPM

1. Open the empty project where you will use **Angular**.
2. Launch the embedded Terminal (View | Tool Windows | Terminal or by hovering your mouse pointer over  in the lower left corner of PyCharm and choosing Terminal from the menu and type `npm install @angular/core` at the command line prompt. That will install the core **Angular** package with the critical runtime parts of the framework. You may also need to install other packages that are parts of **Angular**, see the [list of packages](#).

Generating Angular structures

In an **Angular CLI** project, you can have specific structures generated automatically.

1. On the main menu, choose File | New | Angular CLI.
2. In the pop-up list that opens, click the relevant type of structure.
3. In the dialog box that opens, specify the name of the structure to be generated and the path to it relative to the `src/app` folder of your project. If you want to generate a structure in a separate folder, create this folder first. This does not apply to components, which are by default generated in separate folders

unless the `--flat` option is specified.

If necessary, specify additional options, for example, `--flat` to have a new component generated directly in the specified location without creating a separate folder.

Using Angular language service

PyCharm supports integration with the [Angular language service](#) developed by the [Angular](#) team to improve code analysis and completion for [Angular-TypeScript](#) projects. Note that the [Angular language service](#) works only with the projects that use [Angular 2.3.1](#) or higher and [TypeScript](#) version compatible with it. Also make sure you have a `tsconfig.json` file in your project.

To install the `@angular/language-service` package:

1. Open the Terminal tool window (View | Tool Windows | Terminal or `Alt+F12`).
2. Change the current folder to the project root and at the command line prompt run `npm install @angular/language-service --save-dev`.

The [Angular language service](#) is activated by default so PyCharm starts it automatically together with the [TypeScript service](#) and shows all the errors and warnings in your TypeScript and HTML files both in the editor and in the [TypeScript Compiler Tool Window](#). To activate or disable the service:

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Typescript under Languages & Frameworks.
2. On the [TypeScript](#) page that opens, select the Enable TypeScript Compiler check box and click Configure next to it.
3. In the Service options dialog box that opens, select or clear the Use Angular service check box.

Using the Flow Type Checker

PyCharm provides basic support of [Flow](#) static type checker that brings type annotations to **JavaScript**. This support involves recognition and syntax highlighting of **Flow** structures on all operating systems. If you are using **Unix** or **macOS**, you can also run **Flow** in the command-line mode.

On this page:

- [Before you start](#)
- [Installing Flow](#)
- [Installing Flow globally](#)
- [Installing Flow in a project](#)
- [Configuring Flow in PyCharm](#)

Before you start

1. Install and enable the **NodeJS** repository plugin as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).
2. Download and install the [Node.js](#) runtime environment.
3. Configure the Node.js interpreter in PyCharm:
 1. Open the by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Node.js and NPM under Languages & Frameworks.
 2. On the [Node.js and NPM](#) page that opens, specify the location of the desired Node.js interpreter.

See [Configuring Node.js Interpreters](#) for details.

Installing Flow

The easiest way to install the Flow type checker is to use the **Node Package Manager (npm)**, which is a part of [Node.js](#). See [Installing and Removing External Software Using Node Package Manager](#) for details.

Depending on the desired location of the Flow type checker executable file, choose one of the following methods:

- Install the type checker **globally** at the PyCharm level so it can be used in any PyCharm project.
- Install the type checker in a specific project and thus restrict its use to this project.
- Install the type checker in a project as a [development dependency](#).

In either installation mode, make sure that the parent folder of the Flow type checker is added to the `PATH` variable. This enables you to launch the type checker from any folder.

PyCharm provides user interface both for **global** and **project** installation as well as supports installation through the command line.

Installing Flow globally

Global installation makes a type checker available at the PyCharm level so it can be used in any PyCharm project. Moreover, during installation the parent folder of the type checker is automatically added to the `PATH` variable, which enables you to launch the type checker from any folder.

- Run the installation from the command line in the **global** mode:
 1. Launch the embedded Terminal (View | Tool Windows | Terminal or by hovering your mouse pointer over  in the lower left corner of PyCharm and choosing Terminal from the menu
 2. Switch to the directory where **NPM** is stored or define a `PATH` variable for it so it is available from any folder, see [Installing NodeJs](#).
 3. Type the following command at the command line prompt:

```
npm install -g flow-bin
```

The `-g` key makes the type checker run in the **global** mode. Because the installation is performed through **NPM**, the Flow type checker is installed in the `npm` folder. Make sure this parent folder is added to the `PATH` variable. This enables you to launch the type checker from any folder.

For more details on the **NPM** operation modes, see [npm documentation](#). For more information about installing the Flow type checker, see <https://npmjs.org/package/flow-bin>.

- Run **NPM** from PyCharm using the Node.js and NPM page of the Settings dialog box.
 1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Node.js and NPM under Languages & Frameworks.
 2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the **global** and at the **project** level. Click **+**.
 3. In the Available Packages dialog box that opens, select the required package to install.
 4. Select the Options check box and type `-g` in the text box next to it.
 5. Optionally specify the product version and click Install Package to start installation.

Installing Flow in a project

Local installation in a specific project restricts the use of a type checker to this project.

- Run the installation from the command line:
 1. Launch the embedded Terminal (View | Tool Windows | Terminal or by hovering your mouse pointer over  in the lower left corner of PyCharm and choosing Terminal from the menu

2. Switch to the project root folder and type the following command at the command line prompt:

```
npm install flow-bin
```

– Run **NPM** from PyCharm using the Node.js and NPM page of the Settings dialog box.

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Node.js and NPM under Languages & Frameworks.
2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the **global** and at the **project** level. Click **+**.
3. In the Available Packages dialog box that opens, select the required package.
4. Optionally specify the product version and click Install Package to start installation.

Project level installation is helpful and reliable in [template-based projects](#) of the type **Node Boilerplate** or **Node.js Express**, which already have the `node_modules` folder. The latter is important because **NPM** installs the Flow type checker in a `node_modules` folder. If your project already contains such folder, the Flow type checker is installed there.

Projects of other types or **empty** projects may not have a `node_modules` folder. In this case **npm** goes upwards in the folder tree and installs the Flow type checker in the first detected `node_modules` folder. Keep in mind that this detected `node_modules` folder may be **outside** your current project root.

Finally, if no `node_modules` folder is detected in the folder tree either, the folder is created right under the current project root and the Flow type checker is installed there.

In either case, make sure that the parent folder of the Flow type checker is added to the `PATH` variable. This enables you to launch the type checker from any folder.

For more details, see [Getting Started with Flow](#).

Configuring Flow in PyCharm

To have PyCharm recognize **Flow** structures, provide correct syntax highlighting, report errors properly, and avoid false-positive error highlighting, change the **JavaScript** language level to **Flow**, add a `.flowconfig` configuration file to your project, and supply every file to be checked with a `// @flow` comment on top.

1. To change the language level to **Flow**:

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click JavaScript under Languages & Frameworks.
2. On the **JavaScript** page that opens, choose Flow from the JavaScript Language Version drop-down list.
3. In the Flow executable field, specify the path to the **Flow** executable file.
4. In the Use Flow server for: area, specify the basis for coding assistance by selecting or clearing the following check boxes:
 - Type checking: When this check box is selected, syntax and error highlighting is provided based on the data received from the Flow server. When the check box is cleared, only the basic internal PyCharm highlighting is available.
 - Navigation, code completion, and type hinting: When this check box is selected, suggestion lists for reference resolution and code completion contain both suggestions retrieved from integration with Flow and suggestions calculated by PyCharm. When the check box is cleared, references are resolved through PyCharm calculation only.
 - Code highlighting and built-in inspections: Select this checkbox to power native **Flow** code highlighting and built-in inspections. Please note that enabling this option may cause performance problems. By default, the check box is cleared.

The check boxes are available only when the path to the **Flow** executable file is specified.

2. To have a `.flowconfig` configuration file generated in your project: launch the embedded Terminal (View | Tool Windows | Terminal or by hovering your mouse pointer over  in the lower left corner of PyCharm and choosing Terminal from the menu and run the `flow init` command at the command line prompt.

3. To have a file checked with the **Flow** checker, add a `// @flow` comment at the top of it: just type `flow` and press `Tab` and PyCharm will expand it into `// @flow`.

Using Grunt Task Runner

In this section:

- [Basics](#)
- [Preparing to install Grunt](#)
- [Installing Grunt](#)
 - [Installing the Grunt command line interface](#)
 - [Installing Grunt in a project](#)
- [Running Grunt tasks](#)
 - [Running Grunt tasks from the tasks tree](#)
 - [Building a tasks tree](#)
 - [Running tasks](#)
 - [Running and debugging tasks according to a run configuration](#)
 - [Creating a Grunt.js run/debug configuration](#)
 - [Running a task according to a run configuration](#)
 - [Debugging Grunt tasks](#)
 - [Running a Grunt task as a Before-Launch task](#)
- [Running Grunt tasks automatically](#)

Basics

PyCharm provides integration with the [Grunt JavaScript Task Runner](#). Grunt support involves:

- Parsing `Gruntfile.js` files, recognizing definitions of tasks and targets.
- Building trees of tasks and targets.
- Navigation between a task or a target in the tree and its definition in the `Gruntfile.js` file.
- Running and debugging tasks and targets.
- Configuring the task execution mode and output.

Preparing to install Grunt

1. Download and install [Node.js](#). The runtime environment is required for two reasons:
 - The Grunt task runner is started through **Node.js**.
 - **NPM**, which is a part of the runtime environment, is also the easiest way to download the Grunt task runner.

If you are going to use the command line mode, make sure the path to the parent folder of the **Node.js** executable file and the path to the `npm` folder are added to the `PATH` variable. This enables you to launch the Grunt task runner and `npm` from any folder.

2. Install and enable the **NodeJS** plugin. The plugin is not bundled with PyCharm, but it can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).

Installing Grunt

To use **Grunt** in a PyCharm project, you need two packages:

- A **globally** installed `grunt-cli` package (Grunt command line interface) for executing Grunt commands.
- A `grunt` package installed in the project to build the project tasks tree and provide coding assistance while editing the `Gruntfile.js` or `Gruntfile.coffee` file. Learn more about `Gruntfile.json` at <http://gruntjs.com/getting-started#preparing-a-new-grunt-project>.

These packages can be installed either in the command line mode or through the PyCharm user interface. In either case, installation is performed via `npm`.

Installing the Grunt command line interface

The **Grunt Command Line Interface** is implemented in the `grunt-cli` package which should be installed **globally** to make the functionality available in any PyCharm project and from any folder. Do one of the following:

- Run the installation from the command line in the **global** mode:
 1. Launch the embedded Terminal (View | Tool Windows | Terminal or by hovering your mouse pointer over  in the lower left corner of PyCharm and choosing Terminal from the menu
 2. Switch to the directory where **NPM** is stored or define a `PATH` variable for it so it is available from any folder, see [Installing NodeJs](#).
 3. Type the following command at the command line prompt:

```
npm install -g grunt-cli
```

The `-g` key makes the task runner run in the **global** mode. Because the installation is performed through **NPM**, the Grunt task runner is installed in the `npm` folder. Make sure this parent folder is added to the `PATH` variable. This enables you to launch the task runner from any folder.

For more details on the **NPM** operation modes, see [npm documentation](#). For more information about installing the Grunt task runner, see <https://npmjs.org/package/grunt-cli>.

- Run **NPM** from PyCharm using the Node.js and NPM page of the Settings dialog box.
 1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for

macOS, and click Node.js and NPM under Languages & Frameworks.

2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the **global** and at the **project** level. Click **+**.
3. In the Available Packages dialog box that opens, select the required package to install.
4. Select the Options check box and type `-g` in the text box next to it.
5. Optionally specify the product version and click Install Package to start installation.

Installing Grunt in a project

At the project level, the basic **Grunt** functionality is implemented in the `grunt` package. The package is responsible for building project trees of Grunt tasks and for coding assistance in editing the `Gruntfile.js` file. Do one of the following:

– Run the installation from the command line:

1. Launch the embedded Terminal (View | Tool Windows | Terminal or by hovering your mouse pointer over  in the lower left corner of PyCharm and choosing Terminal from the menu
2. Switch to the project root folder and type the following command at the command line prompt:

```
npm install grunt
```

– Run **NPM** from PyCharm using the Node.js and NPM page of the Settings dialog box.

1. Open the **Settings / Preferences Dialog** by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Node.js and NPM under Languages & Frameworks.
2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the **global** and at the **project** level. Click **+**.
3. In the Available Packages dialog box that opens, select the required package.
4. Optionally specify the product version and click Install Package to start installation.

Running Grunt tasks

Grunt tasks are launched in the following ways:

- From tasks trees in the dedicated **Grunt Tool Window** which opens when you invoke **Grunt** from a `Gruntfile.js` file. As soon as you invoke **Grunt**, it starts building a tree of tasks according to the `Gruntfile.js` on which it was invoked. If a task has **targets**, the task is displayed as a node and the targets are listed under it.
- As a before-launch task, from another run configuration.

The result of executing a task is displayed in the **Run tool window**. The tool window shows the Grunt output, reports the errors occurred, lists the packages or plugins that have not been found, etc. The name of the last executed task is displayed on the title bar of the tool window.

Running Grunt tasks from the tasks tree

Running Grunt tasks right from the tasks tree is easy and fast, the only disadvantage of this approach is that such important additional options as **force execution** and **verbose mode** are not available in this mode. As a result, you cannot have PyCharm, for example, ignore warnings or provide a detailed log.

Building a tasks tree

- If the Grunt tool window is not opened yet, do one of the following:
 - Select the required `Gruntfile.js` file in the Project tool window and choose Show Grunt Tasks on the context menu of the selection.
 - Open the required `Gruntfile.js` file in the editor and choose Show Grunt Tasks on the context menu of the editor.

In either case, the Grunt tool window opens showing the tasks tree built according to the selected or opened `Gruntfile.js` file.

- If the Grunt tool window is already opened, click  on the toolbar and choose the required `Gruntfile.js` file from the list. PyCharm adds a new node and builds a tasks tree under it. The title of the node shows the path to the `Gruntfile.js` file according to which the tree is built.
- To re-build a tree, switch to the required node and click  on the toolbar.

By default, the tasks in a tree are listed in the order in which they are defined in `Gruntfile.js` (option Definition order). To have them listed in the alphabetic order, click the  toolbar button, then choose Sort by on the menu, and then choose Name.

Running tasks

- To run a task or a target, do one of the following:
 - To run the task, select the root node in the tree and choose on the context menu of the selection.
 - To navigate to the definition of a task or target, select the required task or target in the tree and choose on the context menu of the selection.

The task or target execution output will be displayed in the **Run tool window**. The name of the target is shown in the format `<task name>:<target name>`. The tool window shows the Grunt output, reports the errors occurred, lists the packages or plugins that have not been found, etc. The name of the last executed task is displayed on the title bar of the tool window.

- To run several tasks or targets, use the multiselect mode: hold `Shift` (for adjacent items) or `Ctrl` (for non-adjacent items) keys and select the required tasks or targets, then choose Run on the context menu of the selection.

Running and debugging tasks according to a run configuration

Besides using **temporary** run configurations that PyCharm creates automatically, you can create and launch your own **Grunt.js** run configurations. For details about run configurations, see [Run/Debug Configuration](#) and [Creating and Editing Run/Debug Configurations](#).

Creating a Grunt.js run/debug configuration

1. Choose Run | Edit Configuration on the main menu
2. Click the Add New Configuration button **+** on the toolbar, and select Grunt.js from the pop-up list.
3. In the [Run/Debug Configuration: Grunt.js](#) dialog box that opens, specify the name of the run configuration, the tasks to run (use blank spaces as separators), the location of the `Gruntfile.js` file to retrieve the definitions of the tasks from, and the path to the `Grunt` package installed **locally**, under the project root.
Specify the location of the Node executable file and the NodeJS-specific options to be passed to this executable file, see [Node parameters](#) for details.

If applicable, specify the [environment variables](#) for the NodeJS executable file.

To have **Grunt** ignore warnings and continue executing the launched task until the task is completed successfully or an error occurs, select the `-f` check box. Otherwise, the task execution is stopped by the first reported warning. To have the **verbose** mode applied and thus have a full detailed log of a task execution displayed, select the `-v` check box.

Running a task according to a run configuration

Select the run configuration from the list on the main tool bar and then choose Run | Run <configuration name> on the main menu or click the Run toolbar button **▶**. The output is displayed in the [Run tool window](#).

Debugging Grunt tasks

1. Create a **Grunt.js** run/debug configuration, [see above](#).
2. Open the `Gruntfile.js` file in the editor and set the breakpoints in it where necessary.
3. To start a debugging session, select the required debug configuration from the list on the main tool bar and then choose Run | Debug <configuration name> on the main menu or click the Debug toolbar button **🐞**.
4. In the [Debug tool window](#) that opens, analyze the suspended task execution, step through the task, etc. as described in [Examining Suspended Program](#) and [Stepping Through the Program](#).

Running a Grunt task as a Before-Launch task

1. Open the [Run/Debug Configurations](#) dialog by choosing Run | Edit Configurations on the main menu, and select the required configuration from the list or create it anew by clicking **+** and choosing the relevant run configuration type.
2. In the dialog box that opens, click **+** in the Before launch area and choose Run Grunt task from the drop-down list.
3. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool.
Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package.

Running Grunt tasks automatically

If you have some tasks or targets that you run on a regular basis, you can add the corresponding run configurations to a list of **startup tasks**. The tasks will be executed automatically on the project start-up.

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Startup Tasks under Tools.
2. On the [Startup Tasks](#) page that opens, click **+** on the toolbar.
3. From the drop-down list, choose the required **Grunt** run configuration. The configuration is added to the list.
If no applicable configuration is available in the project, click **+** and choose Edit Configurations. Then define a configuration with the required settings in the [Run/Debug Configuration: Grunt.js](#) page that opens. When you save the new configuration it is automatically added to the list of startup tasks.

Using Gulp Task Runner

In this section:

- [Basics](#)
- [Preparing to install Gulp.js](#)
- [Installing Gulp.js](#)
- [Running Gulp.js tasks](#)
 - [Running Gulp.js tasks from the tasks tree](#)
 - [Running tasks from Gulpfile.js](#)
 - [Running and debugging tasks according to a run configuration](#)
 - [Running a Gulp task as a Before-Launch task](#)
- [Running Gulp.js tasks automatically](#)

Basics

PyCharm provides integration with the [Gulp.js Task Runner](#). Gulp.js support involves:

- Parsing `Gulpfile.js` files, recognizing definitions of tasks.
- Building trees of tasks.
- Navigation between a task in the tree and its definition in the `Gulpfile.js` file.
- Running and debugging tasks.
- Configuring the task execution mode and output.
- Creating a Gulp.js run configuration automatically.

Preparing to install Gulp.js

To prepare installing Gulp.js, follow these steps:

1. Download and install [Node.js](#). The runtime environment is required for two reasons:
 - The Gulp.js task runner is started through [Node.js](#).
 - [NPM](#), which is a part of the runtime environment, is also the easiest way to download the Gulp.js task runner.

If you are going to use the command line mode, make sure the path to the parent folder of the [Node.js](#) executable file and the path to the `npm` folder are added to the `PATH` variable. This enables you to launch the Gulp.js task runner and `npm` from any folder.

2. Install and enable the [NodeJS](#) plugin. The plugin is not bundled with PyCharm, but it can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).

Installing Gulp.js

To run Gulp in a PyCharm project you need a Gulp package installed **locally**, under the project root.

You can additionally have a **global** installation of Gulp. In this case, the path to the Gulp installation will be added to the `PATH` environment variable. This will allow you to invoke Gulp from any folder when working in the command line mode.

The basic Gulp.js functionality is implemented in the `gulp.js` package. The package is responsible for building project trees of Gulp tasks and for coding assistance in editing the `Gulpfile.js` file. The `gulp` package can be installed either in the command line mode or through the PyCharm user interface. In either case, installation is performed via `npm`.

Installing Gulp.js in the command line mode

- Launch the embedded Terminal (View | Tool Windows | Terminal or by hovering your mouse pointer over  in the lower left corner of PyCharm and choosing Terminal from the menu and do one of the following:
 - To install Gulp.js in the current project, switch to the project root folder and type the following command at the command line prompt:

```
npm install gulp
```

- To install Gulp.js globally, type:

```
npm install --global gulp
```

Installing Gulp.js through the PyCharm interface

1. Run [NPM](#) from PyCharm using the Node.js and NPM page of the Settings dialog box: Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Node.js and NPM under Languages & Frameworks.
2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the **global** and at the **project** level. Click .

3. In the Available Packages dialog box that opens, select the `gulp` package.
Optionally:
 - Specify the version to install.
 - To have the package installed **globally**, select the Options check box and type `--global` in the text box.
4. Click Install Package to start installation.

Running Gulp.js tasks

Gulp.js tasks can be run either from the tasks tree in the dedicated **Gulp Tool Window**, from the `Gulpfile.js` file, by launching a **Gulp.js run configuration**, or as a before-launch task from another run configuration.

Running Gulp.js tasks from the tasks tree

Gulp.js starts building a **tasks tree** as soon as you invoke **Gulp.js** by choosing Show Gulp Tasks on the context menu of a `Gulpfile.js` in the Project tool window or of a `Gulpfile.js` opened in the editor. The tree is built according to the `Gulpfile.js` file on which **Gulp.js** was invoked. If you have several `Gulpfile.js` files in your project, you can build a separate tasks tree for each of them and run tasks without abandoning the previously built tasks trees. Each tree is shown under a separate node.

Technically, PyCharm invokes **Gulp.js** and processes `Gulpfile.js` according to the **default Gulp.js run configuration**. However this is done silently and does not require any steps from your side.

Building a tasks tree

- If the Gulp tool window is not opened yet, do one of the following:
 - Select the required `Gulpfile.js` file in the Project tool window and choose Show Gulp Tasks on the context menu of the selection.
 - Open the required `Gulpfile.js` file in the editor and choose Show Gulp Tasks on the context menu of the editor.

In either case, the Gulp tool window opens showing the tasks tree built according to the selected or opened `Gulpfile.js` file.

- If the Gulp tool window is already opened, click  on the toolbar and choose the required `Gulpfile.js` file from the list. PyCharm adds a new node and builds a tasks tree under it. The title of the node shows the path to the `Gulpfile.js` file according to which the tree is built.
- To re-build a tree, switch to the required node and click  on the toolbar.

By default, the tasks in a tree are listed in the order in which they are defined in `Gulpfile.js` (option Definition order). To have them listed in the alphabetic order, click the  toolbar button, then choose Sort by on the menu, and then choose Name.

If your `Gulpfile.js` is written in **ECMA6**, by default PyCharm does not recognize this format and fails to build a tasks tree. To solve this problem, update the **default Gulp.js run configuration**:

1. Choose Run | Edit Configuration on the main menu
2. Under the Defaults node, click Gulp.js.
3. In the **Run/Debug Configuration: Gulp.js** dialog box that opens, type `--harmony` in the Node options text box and click OK.

Running a task

- To run a **Gulp.js** task from a tree of tasks, do one of the following:
 - To run the task, select the root node in the tree and choose on the context menu of the selection.
 - To navigate to the definition of a task, select the required task in the tree and choose on the context menu of the selection.

The task execution output will be displayed in the **Run tool window**. The tool window shows the Gulp.js output, reports the errors occurred, lists the packages or plugins that have not been found, etc. The name of the last executed task is displayed on the title bar of the tool window.

- To run several tasks, use the multiselect mode: hold `Shift` (for adjacent items) or `Ctrl` (for non-adjacent items) keys and select the required tasks, then choose Run on the context menu of the selection.

Running tasks from Gulpfile.js

You can run tasks right from the `Gulpfile.js` file opened in the editor without previously building a tree of tasks.

To run tasks from Gulpfile.js, do one of the following

- Position the cursor at the definition of the task to run and choose Run <task name> on the context menu of the selection.
PyCharm creates and launches a **temporary** run configuration with the name of the selected task.

The task execution output will be displayed in the **Run tool window**. The tool window shows the Gulp.js output, reports the errors occurred, lists the packages or plugins that have not been found, etc. The name of the last executed task is displayed on the title bar of the tool window.

- To save an automatically created temporary run configuration, position the cursor at the definition of the task for which it was created and choose Save <task name> on the context menu of the selection. Learn more in [Creating and Saving Temporary Run/Debug Configurations](#).

Running and debugging tasks according to a run configuration

Besides using **temporary** run configurations that PyCharm creates automatically, you can create and launch your own **Gulp.js** run configurations. For details

about run configurations, see [Run/Debug Configuration](#) and [Creating and Editing Run/Debug Configurations](#).

Creating a Gulp.js run configuration

1. Choose Run | Edit Configuration on the main menu
2. Click the Add New Configuration button **+** on the toolbar, and select Gulp.js from the pop-up list.
3. In the [Run/Debug Configuration: Gulp.js](#) dialog box that opens, specify the name of the run configuration, the tasks to run (use blank spaces as separators), the location of the `Gulpfile.js` file to retrieve the definitions of the tasks from, and the path to the `Gulp` package installed locally, under the project root.
Specify the location of the Node executable file and the NodeJS-specific options to be passed to this executable file, see [Node parameters](#) for details.

If applicable, specify the [environment variables](#) for the NodeJS executable file.

If applicable, in the Arguments field, specify the arguments for tasks to be executed with. Use the following format:

```
--<parameter_name> <parameter_value>
```

For example: `--env development` .

For details about passing task arguments, see <https://github.com/gulpjs/gulp/blob/master/docs/recipes/pass-arguments-from-cli.md>

Running a task according to a run configuration

- Select the run configuration from the list on the main tool bar and then choose Run | Run <configuration name> on the main menu or click the Run toolbar button **▶**. The output is displayed in the [Run tool window](#).

Debugging Gulp tasks

1. Create a Gulp.js run/debug configuration, [see above](#).
2. Open the `Gulpfile.js` file in the editor and set the breakpoints in it where necessary.
3. To start a debugging session, select the required debug configuration from the list on the main tool bar and then choose Run | Debug <configuration name> on the main menu or click the Debug toolbar button **🐛**.
4. In the [Debug tool window](#) that opens, analyze the suspended task execution, step through the task, etc. as described in [Examining Suspended Program](#) and [Stepping Through the Program](#).

Running a Gulp task as a Before-Launch task

1. Open the [Run/Debug Configurations](#) dialog by choosing Run | Edit Configurations on the main menu, and select the required configuration from the list or create it anew by clicking **+** and choosing the relevant run configuration type.
2. In the dialog box that opens, click **+** in the Before launch area and choose Run Gulp task from the drop-down list.
3. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool.
Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package.

Running Gulp.js tasks automatically

If you have some tasks that you run on a regular basis, you can add the corresponding run configurations to a list of **startup tasks**. The tasks will be executed automatically on the project start-up.

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Startup Tasks under Tools.
2. On the [Startup Tasks](#) page that opens, click **+** on the toolbar.
3. From the drop-down list, choose the required **Gulp.js** run configuration. The configuration is added to the list.
If no applicable configuration is available in the project, click **+** and choose Edit Configurations. Then define a configuration with the required settings in the [Run/Debug Configuration: Gulp.js](#) page that opens. When you save the new configuration it is automatically added to the list of startup tasks.

Using JavaScript Code Quality Tools

PyCharm provides facilities to run JavaScript-specific code quality inspections through integration with the following code verification tools:

- [JSLint](#). The tool is bundled with PyCharm.
- [JSHint](#). PyCharm comes bundled with the default version of the tool, but you can have an alternative version downloaded.
- [Closure Linter](#). The tool is not bundled with PyCharm, so you need to download it yourself.
- [JSCS](#). The tool is not bundled with PyCharm, but you can install it from the PyCharm interface using the [Node Package Manager \(npm\)](#).
- [ESLint](#). The tool is not bundled with PyCharm, but you can install it from the PyCharm interface using the [Node Package Manager \(npm\)](#).

All these tools register themselves as PyCharm code inspections: they check JavaScript code for most common mistakes and discrepancies without running the application. When a tool is activated, it launches automatically on the edited **JavaScript** file. Discrepancies are highlighted and reported in pop-up information windows, a pop-up window appears when you hover the mouse pointer over a stripe in the Validation sidebar. You can also press `Alt+Enter` to examine errors and apply suggested quick fixes. Learn more about inspections and intention actions at [Code Inspection](#) and [Intention Actions](#).

On this page:

- [Enabling and configuring the built-in JSLint tool](#)
- [Enabling and configuring JSHint](#)
- [Enabling and configuring Closure Linter](#)
- [Using JSCS](#)
- [Using ESLint](#)
- [Using JavaScript Standard Style](#)

Enabling and configuring the built-in JSLint tool

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS.
Expand the JavaScript node under Languages & Frameworks, and then click JSLint under Code Quality Tools.
2. On the [JSLint page](#) that opens, select the Enable check box. After that all the controls in the page become available.
3. Define the set of common mistakes to check the code for. To enable a validation, select the check box next to it.

Enabling and configuring JSHint

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS.
Expand the JavaScript node under Languages & Frameworks, and then click JSHint under Code Quality Tools.
2. On the [JSHint page](#) that opens, select the Enable check box. After that all the controls in the page become available.
3. From the Version drop-down list, choose the version of the tool to apply.
PyCharm comes bundled with **version 1.0.0**, which is used by default. PyCharm provides the ability to download another version, which is not bundled. Actually, the alternative version is downloaded only once, whereupon it is available without download.
4. Configure the behaviour of JSHint:
 - To have the code verified according to the rules from a previously created configuration file, select the Use config files check box.
A configuration file is a JSON file with the extension `.jshintrc` that specifies which JSHint options should be enabled or disabled. PyCharm will look for a `.jshintrc` file in the working directory. If the search fails, PyCharm will search in the parent folder, then again in the parent folder. The process is repeated until PyCharm finds a `.jshintrc` or reaches the project root. To have PyCharm still run verification in this case, specify the default configuration file to use.

Starting with PyCharm version 6.0.1, the **globals** setting in `.jshintrc` is supported. For earlier versions, a **Predefined (, separated)"** tree node is displayed in the bottom of the tree in the UI.
 - To configure verification manually, clear the check box and define the set of common mistakes to check the code for in the Options area. The controls in the area fall into two groups:
 - Enforcing options: select the check boxes in this group to enable very strict behaviour of the verification tool and thus allow only safe JavaScript.
 - Relaxing options: select/clear the check boxes in this area to suppress warnings when certain types of discrepancies are detected.
 - Environments: select/clear these check boxes to specify for which environments you want global properties predefined.To enable a validation, select the check box next to it.

Enabling and configuring Closure Linter

1. [Download and install the Closure Linter tool](#).
2. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS.
Expand the JavaScript node under Languages & Frameworks, and then click Closure Linter under Code Quality Tools.
3. On the [Closure Linter page](#) that opens, select the Enable check box. After that all the controls in the page become available.
4. Specify the path to the **Closure Linter** executable file:
 - `<Python_home>\Scripts\jslint.exe` for Windows
 - `/usr/local/bin/gjslint` for Linux and macOS
5. Specify the path to the previously created configuration file.

Note that the behaviour of **Closure Linter** may slightly differ depending on whether it is run from PyCharm or in the command line mode if a configuration file is

parsed before the `gjslint` process started and all the parsed options are passed as arguments to the `gjslint` process. When **Closure Linter** is running inside PyCharm, the `gjslint` process is launched in the following way:

```
/path/to/gjslint --flagfile /path/to/config_file --recurse=no /path/to/user_source_file.js
```

Therefore, the following command fails if the configuration file has a space instead of a `=`:

```
--jslint_error indentation
```

Using JSCS

The **JSCS** tool is run through **NodeJS**, therefore make sure the **NodeJS** framework is downloaded and installed on your computer. The framework also contains the **Node Package Manager(npm)** through which **JSCS** is installed.

Integration with **NodeJS** and **NPM** is supported through the **NodeJs** plugin. The plugin is not bundled with PyCharm, but it can be installed from the **JetBrains plugin repository** as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).

To install JSCS:

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Node.js and NPM under Languages & Frameworks.
2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the **global** and at the **project** level. Click `+`.
3. In the Available Packages dialog box that opens, select the `jscs` package and click Install Package.
Learn more about installing tools through **NPM** in [Installing and Removing External Software Using Node Package Manager](#).

To activate and configure JSCS:

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS.
Expand the JavaScript node under Languages & Frameworks, and then click JSCS under Code Quality Tools.
2. Select the Enable check box to activate **JSCS**. After that the controls in the dialog box become available.
3. In the Node Interpreter field, specify the NodeJS installation home. Type the path to the NodeJS executable file manually, or click the `...` button and select the location in the dialog box, that opens.
4. In the JSCS Package field, specify the location of the `jscs` package installed in the current project, see [Installing JSCS](#).
5. Appoint the [configuration](#) to use.

By default, PyCharm first looks for a `jscsConfig` property in the `package.json` file of the current project. If no such property is found, PyCharm looks for a `.jscsrc` or a `.jscs.json` configuration file. PyCharm starts the search from the folder where the file to be checked is stored, then searches in the parent folder, and so on until reaches the project root. Accordingly, you have to define the configuration to apply either as a `jscsConfig` property in the `package.json` file or in a `.jscsrc` or a `.jscs.json` configuration file, or in a custom **JSON** configuration file.

You can also apply a predefined set of rules, either independently or in combination with a configuration file. In the latter case, the rules from the configuration file override the predefined rules.

- To have PyCharm look for a `jscsConfig` property in the `package.json` file or for a `.jscsrc` or a `.jscs.json` file, choose the Search for config(s) option.
 - To use a custom file, choose the Configuration File option and specify the location fo the file in the Path field. Choose the path from the drop-down list, or type it manually, or click the `...` button and select the relevant file from the dialog box that opens.
 - To have a predefined set or rules applied, choose the desired set from the Code Style Preset drop-down list.
6. If necessary, from the Code Style Preset drop-down list, choose the set of predefined rules associated with the code style you use.

Using ESLint

The **ESLint** tool is run through **NodeJS**, therefore make sure the **NodeJS** framework is downloaded and installed on your computer. The framework also contains the **Node Package Manager(npm)** through which **ESLint** is installed.

Integration with **NodeJS** and **NPM** is supported through the **NodeJS** plugin. The plugin is not bundled with PyCharm, but it can be installed from the **JetBrains plugin repository** as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).

To install ESLint, open the PyCharm built-in Terminal (View | Tool Windows | Terminal or press `Alt+F12`) and at the command line prompt type `npm install eslint --save-dev` to install the package in the current project or `npm install eslint --global` to install the package globally.

You can also install the package on the Node.js and NPM page as described in [Installing and Removing External Software Using Node Package Manager](#).

Learn more on the [ESLint official website](#).

To activate and configure ESLint:

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS.
Expand the JavaScript node under Languages & Frameworks, and then click ESLint under Code Quality Tools.
2. Select the Enable check box to activate **ESLint**. After that the controls in the dialog box become available.
3. In the Node Interpreter field, specify the NodeJS installation home. Type the path to the NodeJS executable file manually, or click the `...` button and select the location in the dialog box, that opens.
4. In the ESLint Package field, specify the location of the `eslint` package installed in the current project, see [Installing ESLint](#) or the `standard` package

installed in your project or globally, see [Installing JavaScript Standard](#).

5. Appoint the configuration to use.

By default, PyCharm first looks for a `.eslintrc` configuration file. PyCharm starts the search from the folder where the file to be checked is stored, then searches in the parent folder, and so on until reaches the project root. If no `.eslintrc` file is found, **ESLint** uses its default embedded configuration file. Accordingly, you have to define the configuration to apply either in a `.eslintrc` configuration file, or in a custom **JSON** configuration file, or rely on the default embedded configuration.

– To have PyCharm look for a `.eslintrc` file, choose the Search for `.eslintrc` option. If no `.eslintrc` file is found, the default embedded configuration file will be used.

– To use a custom file, choose the Configuration File option and specify the location of the file in the Path field. Choose the path from the drop-down list, or type it manually, or click the  button and select the relevant file from the dialog box that opens.

6. If necessary, in the Additional Rules Directory field, specify the location of the files with additional code verification rules. These rules will be applied after the rules from `.eslintrc` or the above specified custom configuration file and accordingly will override them.

7. If necessary, in the Extra ESLint Options field, specify additional command line options to run **ESLint** with using spaces as separators. See [ESLint Command Line Interface Options](#) for details.

Using JavaScript Standard Style

PyCharm supports [JavaScript Standard Style](#).

To install Standard, open the PyCharm built-in Terminal (View | Tool Windows | Terminal or press `Alt+F12`) and at the command line prompt type `npm install standard --save-dev` to install the package in the current project or `npm install standard --global` to install the package globally.

You can also install the package on the Node.js and NPM page as described in [Installing and Removing External Software Using Node Package Manager](#).

Learn more on the [ESLint official website](#). See also [JavaScript Standard Style: Installation](#).

To configure PyCharm code style options for JavaScript to follow the main rules Standard declares so they are applied when you type the code or reformat it, open the [Code Style. JavaScript](#) page (File | Settings | Editor | Code Style | JavaScript for Windows and Linux or PyCharm | Preferences | Editor | Code Style | JavaScript for macOS), click Set from, and then choose Predefined Style | JavaScript Standard Style. The style will replace your current scheme.

Since Standard is based on ESLint, you can use Standard via PyCharm ESLint integration. To enable linting with Standard open the [ESLint](#) page as described [above](#), select the Enable check box, and specify the location of the `standard` package in the ESLint Package field.

If you open a new project that already uses Standard and has it listed in the project's `package.json` file, PyCharm will detect that and enable linting with Standard automatically.

Using Meteor

PyCharm provides integration with the [Meteor framework](#). Meteor support in PyCharm involves:

- Automatic recognition of **Meteor** projects by detecting the `.meteor` folder and excluding the `.meteor/local` folder from project.
- Attaching the predefined **Meteor** library to the project automatically.
- Coding assistance: syntax highlighting, code completion, resolving references.
- Support of **Spacebars** via **Handlebars** with completion for if and each directives. There is also support for navigation between JavaScript source code and templates, allowing for easy navigation by using `Ctrl+B` to [go to Declaration](#).
- Running and debugging the server and the client sides of applications.

On this page:

- [Before you start](#)
- [Installing Meteor](#)
- [Preparing to develop a Meteor application](#)
 - [Generating a Meteor application stub](#)
 - [Enabling Meteor integration in an existing project](#)
 - [Importing Meteor packages](#)
- [Running and debugging a Meteor application](#)
 - [Creating a Meteor run configuration](#)
 - [Configuring the update policy in the debugging mode](#)
 - [Configuring the update policy for the server side code](#)
 - [Configuring the update policy for the client side code](#)
 - [Running a Meteor application](#)
 - [Debugging a Meteor application](#)

Before you start

Make sure the **Meteor** and the **Handlebars/Mustache** plugins are activated. The **Handlebars/Mustache** enables PyCharm to recognize **Spacebars** templates that are an extension of the **Handlebars/Mustache** templates. As a side effect, HTML files in **Meteor** projects are marked with the **Handlebars/Mustache** icon . The plugins are not bundled with PyCharm, but they can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#). Once enabled, the plugins are available at the IDE level, that is, you can use them in all your PyCharm projects.

Installing Meteor

The installation procedure depends on the operating system you are using:

1. To install **Meteor** on Windows, download the `LaunchMeteor.exe` installer at <https://www.meteor.com/install>.
2. To install **Meteor** on OSX (OSX 10.6 or higher is required) or Linux, type the following command:

```
$ curl https://install.meteor.com | /bin/sh
```

Learn more at <http://docs.meteor.com/>.

Preparing to develop a Meteor application

To start your development, you need a PyCharm project with the **Meteor**-specific structure. You can have an application stub that meets these requirements generated automatically or open an existing **Meteor** project in PyCharm and configure **Meteor** support in it.

Generating a Meteor application stub

1. In the left-hand pane, choose Meteor JS App.
2. In the right-hand pane:
 1. In the Location text box, specify the path to the project folder where the project-related files will be stored.
 2. Specify the location of the **Meteor** executable file (see [Installing Meteor](#)).
 3. From the Template drop-down list, choose the sample to generate. To have a basic project structure generated, choose the Default option.
 4. In the Filename text box, type the name for the mutually related `.js`, `.html`, and `.css` files that will be generated. The text box is available only if the **Default** sample type is selected from the Template drop-down list.

When you click Create, PyCharm generates a skeleton of a **Meteor** application, including an HTML file, a JavaScript file, a CSS file, and a `.meteor` folder with subfolders. The `.meteor/local` folder, which is intended for storing the built application, is automatically marked as **excluded** and is not involved in indexing.

By default, **excluded** files are shown in the project tree. To have the `.meteor/local` folder hidden, click the  button on the toolbar of the Project tool window and remove a tick next to the Show Excluded Files option.

PyCharm also automatically attaches the predefined **Meteor** library to the project, thus enabling syntax highlighting, resolving references, and code completion. See [Configuring JavaScript Libraries](#) for details.

Meteor uses **Spacebars** templates that are an extension of the **Handlebars/Mustache** templates. PyCharm recognizes **Spacebars** templates, but as a side effect marks HTML files in **Meteor** projects with the **Handlebars/Mustache** icon .

Enabling Meteor integration in an existing project

PyCharm automatically detects **Meteor** projects by locating the `.meteor` folder. The `.meteor/local` folder, which is intended for storing the built application,

is automatically marked as **excluded** and is not involved in indexing.

1. Open the desired **Meteor** project in PyCharm by choosing File | Open on the main menu or clicking Open on the Welcome Screen.
2. Open the **Settings / Preferences Dialog** by pressing (`Ctrl+Alt+S`) or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS. Expand the Languages & Frameworks node, and then click Meteor under JavaScript.
3. On the **Meteor** page that opens:
 1. Check the location of the **Meteor** executable file or specify the path to it if PyCharm has not detected the executable file automatically.
 2. To have PyCharm involve the `.meteor/local` folder and its contents in indexing, clear the Automatically exclude ".meteor/local" directory on open project check box. By default, the check box is selected and the `.meteor/local` folder, which is intended for storing the built application, is automatically marked as **excluded** and is not involved in indexing.

By default, **excluded** files are shown in the project tree. To have the `.meteor/local` folder hidden, click the  button on the toolbar of the Project tool window and remove a tick next to the Show Excluded Files option.
 3. Make sure the Automatically import Meteor packages as external library check box is selected.
 - When the check box is selected, PyCharm automatically imports the external packages from the `meteor/packages` file. As a result, PyCharm provides full range coding assistance: resolves references to **Meteor** built-in functions, for example, `check(true)`, and to functions from third-party packages, provides proper syntax and error highlighting, supports debugging with source maps, etc.
 - When this check box is cleared, PyCharm does not automatically import the external packages from the `meteor/packages` file. As a result no coding assistance is provided. To improve the situation, open the `meteor/packages` file in the editor and click the Import packages as library link or run the `meteor --update` command.

By default, the check box is selected.
 4. Click Apply to save the settings without leaving the dialog box.
4. Check that PyCharm has attached the **Meteor** library to the project:
 1. Click Libraries under JavaScript.
 2. On the **JavaScript. Libraries** page that opens, make sure the check box next to the **Meteor** project library in the Libraries list is selected.
 3. Click OK.

Alternatively, choose Use JavaScript Library on the context menu of the editor and make sure a tick is set next to Meteor project library.

By default, **excluded** files are shown in the project tree. To have the `.meteor/local` folder hidden, click the  button on the toolbar of the Project tool window and remove a tick next to the Show Excluded Files option.

PyCharm also automatically attaches the predefined **Meteor library** to the project, thus enabling syntax highlighting, resolving references, and code completion. See [Configuring JavaScript Libraries](#) for details.

Importing Meteor packages

Besides the predefined **Meteor** library that ensures basic **Meteor**-specific coding assistance, you can download additional [packages](#) that are defined in the `.meteor/local/packages` file.

1. Open the `.meteor/local/packages` file in the editor.
2. Click the Import Meteor Packages link in the upper right-hand corner of the screen.
3. In the dialog box that opens, specify the packages to download depending on the type of the application you are going to develop in your project.
 - Client
 - Server
 - Cordova: choose this option to import the packages that support development of **Meteor** applications for iOS and Android, see [Meteor Cordova Phonegap Integration](#) for details.

Running and debugging a Meteor application

With PyCharm, you can debug both the **client-side** and the **server-side** of **Meteor** JavaScript code simultaneously using the PyCharm debugger. The debugger also pauses at the breakpoints set in the sources stored in the `/packages` folder. This functionality is supported both for the client side and for the server side code.

Meteor applications are launched only through a dedicated run configuration. Technically, several **Meteor projects** that implement different applications can be combined within one single PyCharm project. To run and debug these applications independently, create a separate run configuration for each of them with the relevant **working directory**. To avoid port conflicts, these run configurations should use different ports. In the Program Arguments field, specify a separate port for each run configuration in the format `--port=<port_number>`. The results are shown in the console and/or in the browser depending on the settings specified in the Browser / Live Edit tab, see [Creating a run configuration](#) below.

In the debugging mode, PyCharm provides the **Live Edit** functionality which supports both automatic and manual upload of updated files on the server side. For the client side, **Live Edit** does not provide any way to apply the changes. For information on enabling and configuring the **Live Edit** functionality, see [Live Editing of HTML, CSS, and JavaScript](#).

Creating a Meteor run configuration

Technically, PyCharm creates separate run configurations for the server-side and the client-side code, but you specify all your settings in one dedicated **Meteor** run configuration.

1. On the main menu, choose Run | Edit Configurations. In the Edit Configuration dialog box, that opens, click the Add New Configuration toolbar button , and choose Meteor on the context menu.
2. In the Configuration tab of the **Run/Debug Configuration: Meteor** dialog box that opens, specify the name of the configuration and the location of the **Meteor** executable file according to the installation (see [Installing Meteor](#)).

Optionally, in the Program Arguments field, specify the command line additional parameters to be passed to the executable file on start up, if

applicable. These can be, for example, `--dev`, `--test`, or `--prod` to indicate the environment in which the application is running (**development**, **test**, or **production** environments) so different resources are loaded on start up.

In the Working Directory field, specify the folder under which the application files to run are stored. This folder must contain a `.meteor` folder in the root to be treated a **Meteor project**. By default, the field shows the path to the PyCharm project root.

Technically, several **Meteor projects** that implement different applications can be combined within one single PyCharm project. To run and debug these applications independently, create a separate run configuration for each of them with the relevant **working directory**. To avoid port conflicts, these run configurations should use different ports. In the Program Arguments field, specify a separate port for each run configuration in the format `--port=<port_number>`.

3. Switch to the Browser / Live Edit tab and configure the behaviour of the browser:
 1. To view the results of the client-side code execution during running and debugging, select the After Launch check box and choose the browser to open from the drop-down list.
 2. In the text box below, specify the URL address to open the application in. The default value is `http://localhost:3000`.
 3. To enable debugging the client-side code in the browser, select the With JavaScript Debugger check box.
 4. Click OK to save the run/debug configuration.

Configuring the update policy in the debugging mode

In the debugging mode, PyCharm provides the **Live Edit** functionality which supports both automatic and manual upload of updated files on the server side. For the client side, **Live Edit** does not provide any way to apply the changes.

For information on enabling and configuring the **Live Edit** functionality, see [Live Editing of HTML, CSS, and JavaScript](#).

The client-side code can be updated through the native [Meteor hot code pushes functionality](#).

Configuring the update policy for the server side code

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS. Expand the Debugger node under Build, Execution, Deployment, and then click Live Edit. The [Live Edit](#) page that opens.
2. In the Update area, configure the way changes made to the code during a debugging session are applied. This functionality is available only for the **server-side** code. To have the **client side** code updated, select the Enable Meteor 'Hot code push' check box on the [Meteor page](#) of the Settings dialog box.

Note that an update will be performed only if none of the modified files have any syntax errors.

 - Auto in (ms): Choose this option to have the changes applied automatically at certain time interval and specify the delay in ms in the text box. This policy is not available for the client-side code of **Meteor** applications.
 - Manual: Choose this option to apply the changes manually by clicking the Update <run configuration name> JavaScript button  or the Update <run configuration name> button  on the toolbar of the Debug tool window.

It is recommended that you choose this option because applying changes to the **client-side** code is not supported.
 - Restart if hotswap fails:

With changes in HTML, CSS, and JavaScript on the client side, the contents of a Web page in the browser are updated without reloading. For **NodeJS** or **Meteor** applications, PyCharm first tries to update the application incorporating the changes without restarting the NodeJS server.

 - Select this check box to have PyCharm try restarting the server if the changes cannot be applied automatically. After the restart, PyCharm brings you to the execution point where the problem took place.

If even with this option chosen automatic upload still fails, you will have to restart the server manually by clicking the Rerun <run configuration name> button .
 - When the check box is cleared, PyCharm just displays a pop-up window informing you about the failure and suggesting you to restart the server manually.

Configuring the update policy for the client side code

The client-side code can be updated through the native [Meteor hot code pushes functionality](#). The functionality is enabled by default, to activate it:

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS. Expand the Languages & Frameworks node, and then click Meteor under JavaScript.
2. On the [Meteor](#) page that opens, select the Enable Meteor 'Hot code push' check box.

Running a Meteor application

- To run an application, select the required run configuration from the list on the main tool bar and then choose Run | Run <configuration name> on the main menu or click the Run toolbar button .
- If in the run configuration the browser is configured to open automatically upon application start, the browser opens showing the result of application execution. You can view the same result by opening the browser of your choice manually at the URL specified in the run configuration.

Debugging a Meteor application

1. Set the [breakpoints](#) in the code, where necessary.
2. To initiate a debugging session, select the required debug configuration from the list on the main tool bar and then choose Run | Debug <configuration name> on the main menu or click the Debug toolbar button . The Debug tool window opens showing two tabs: one for

debugging the server-side code marked with  and the other for debugging the client-side code marked with .

3. [Explore the suspended program](#) and [step through the program](#).

4. Apply the changes to the application depending on the update policy defined in the run configuration:

- To apply the changes to the client-side code, switch to the  tab and click the Update <run configuration name> JavaScript button  on the toolbar.
- The changes to the server-side code are applied depending on the update policy defined in the run configuration:
 - If you have chosen manual upload, switch to the  tab and click the Update <run configuration name> button  on the toolbar.
 - If you have chosen automatic upload, updates to the server-side code are applied automatically. If automatic upload fails, restart the application by clicking Rerun <run configuration name> button . Choosing the Restart if hotswap fails option on the [Live Edit](#) page may help: PyCharm will attempt to restart the server automatically.

Using PhoneGap/Cordova

In PyCharm, you can develop applications intended for running on various mobile platforms, including [Android](#), using the [PhoneGap](#), [Apache Cordova](#), and [Ionic](#) frameworks.

On this page:

- [Before you start](#)
- [Installing PhoneGap/Cordova/Ionic](#)
 - [Installing PhoneGap in the command line mode](#)
 - [Installing PhoneGap/Cordova/Ionic through the PyCharm interface](#)
- [Preparing to use PhoneGap/Cordova/Ionic in a project](#)
 - [Generating a PhoneGap/Cordova/Ionic application stub](#)
 - [Enabling PhoneGap/Cordova/Ionic integration in an existing project](#)
- [Creating and launching a PhoneGap/Cordova/Ionic run configuration](#)

Before you start

1. Make sure the **PhoneGap/Cordova** and the **NodeJS** plugins are enabled. The plugins are not bundled with PyCharm, but they can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#). Once enabled, the plugins are available at the IDE level, that is, you can use them in all your PyCharm projects.
2. Download and install [Node.js](#) because **NPM**, which is a part of the framework, is also the easiest way to download **PhoneGap** and **Cordova**.
3. Download and install an emulator tool. These tools are specific for the target platform and the operating system you use:
 - <http://cordova.apache.org/docs/en/latest/guide/platforms/ios/index.html>
 - <http://cordova.apache.org/docs/en/latest/guide/platforms/android/index.html>
 - <http://cordova.apache.org/docs/en/latest/guide/platforms/blackberry10/home.html>
 - <http://cordova.apache.org/docs/en/latest/guide/platforms/osx/index.html>
 - <http://cordova.apache.org/docs/en/latest/guide/platforms/ubuntu/index.html>
 - <http://cordova.apache.org/docs/en/latest/guide/platforms/win8/index.html>
 - <http://cordova.apache.org/docs/en/latest/guide/platforms/wp8/home.html>

Installing PhoneGap/Cordova/Ionic

To use **PhoneGap/Cordova**, you need a `phonegap`, or `cordova`, or `ionic` package. The package can be installed either in the command line mode or through the PyCharm user interface. In either case, installation is performed via **npm**.

Installing PhoneGap in the command line mode

- Launch the embedded Terminal (View | Tool Windows | Terminal or by hovering your mouse pointer over  in the lower left corner of PyCharm and choosing Terminal from the menu and type one of the following commands at the command line prompt:

```
npm install -- global phonegap
npm install -- global cordova
npm install -- global ionic
```

With the `--global` option, the package will be installed **globally** so it is accessible from all your PyCharm projects.

Installing PhoneGap/Cordova/Ionic through the PyCharm interface

1. Run **NPM** from PyCharm using the Node.js and NPM page of the Settings dialog box: Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Node.js and NPM under Languages & Frameworks.
2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the **global** and at the **project** level. Click **+**.
3. In the Available Packages dialog box that opens, select the `phonegap`, or `cordova`, or `ionic` package, depending on your preferences and workflow.
Optionally:
 - Specify the version to install.
 - To have the package installed **globally** so it is accessible from all your PyCharm projects, select the Options check box and type `--global` in the text box.
4. Click Install Package to start installation.

Preparing to use PhoneGap/Cordova/Ionic in a project

To start your development, you need a PyCharm project with the **PhoneGap/Cordova/Ionic**-specific structure. You can have an application stub that meets these requirements generated automatically or open an existing **PhoneGap/Cordova/Ionic** project in PyCharm and configure **PhoneGap/Cordova/Ionic** support in it.

Generating a PhoneGap/Cordova/Ionic application stub

1. In the left-hand pane, choose PhoneGap/Cordova App.
2. In the right-hand pane:
 1. In the Location text box, specify the path to the project folder where the project-related files will be stored.
 2. Specify the location of the executable file `phonegap.cmd`, or `cordova.cmd`, or `ionic.cmd` (see [Installing PhoneGap/Cordova/Ionic](#)).

When you click Create, PyCharm generates a skeleton of a **PhoneGap/Cordova/Ionic** application with the framework-specific structure.

Enabling PhoneGap/Cordova/Ionic integration in an existing project

1. Open the desired **PhoneGap/Cordova/Ionic** project in PyCharm by choosing File | Open on the main menu or clicking Open on the Welcome Screen.
2. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS. Expand the Languages & Frameworks node, and then click PhoneGap/Cordova under JavaScript.
3. On the [PhoneGap/Cordova](#) page that opens:
 1. Check the location of the executable file `phonegap.cmd`, or `cordova.cmd`, or `ionic.cmd` or specify the path to it if PyCharm has not detected the executable file automatically.
PyCharm detects the installed version and displays it in the PhoneGap/Cordova Version read-only field.
 2. In the PhoneGap/Cordova Working Directory field, specify the folder under which the **PhoneGap/Cordova/Ionic** application files to run are stored.
 3. In the Plugins area, configure a list of plugins to use in your development by installing required packages. The list shows all the **PhoneGap/Cordova/Ionic** plugins that are currently installed on your computer, both at the **global** and at the **project** level.
 - To install a plugin, click the Install button **+**. In the Available Packages dialog box that opens, select the required package.
To have the plugin installed **globally** so it is accessible from all your PyCharm projects, select the Options check box and type `--global` in the text box. Click Install Package.
 - To remove a plugin, select it in the list and click the Uninstall button **-**.
 - To upgrade a plugin to the latest available version, select the plugin in the list and click the Upgrade button **↑**.

See [Apache Cordova Plugins](#) and [PhoneGap Plugins](#) for information about plugins and their use.

Creating and launching a PhoneGap/Cordova/Ionic run configuration

PhoneGap/Cordova/Ionic applications are executed according to a dedicated run/debug configuration.

1. On the main menu, choose Run | Edit Configurations. In the Edit Configuration dialog box, that opens, click the Add New Configuration toolbar button **+**, and choose PhoneGap/Cordova on the context menu.
2. In the [Run/Debug Configuration: PhoneGap/Cordova](#) dialog box that opens, specify the following:
 1. The name of the configuration.
 2. In the PhoneGap/Cordova Executable Path field, specify the location of the executable file `phonegap.cmd`, `cordova.cmd`, or `ionic.cmd` (see [Installing PhoneGap/Cordova/Ionic](#)).
 3. In the PhoneGap/Cordova Working Directory field, specify the folder under which the **PhoneGap/Cordova/Ionic** application files to run are stored.
 4. From the Command drop-down list, choose the command to run. The contents of the drop-down list, depend on the actually used framework, namely, on the executable file specified in the PhoneGap/Cordova Executable Path field. The available options are:
 - For **PhoneGap**:
 - emulate
 - run
 - prepare
 - serve
 - remote build
 - remote run
 - See <https://www.npmjs.org/package/phonegap> for a list of **PhoneGap**-specific commands with descriptions.
 - For **Cordova**:
 - emulate
 - run
 - prepare
 - serve
 - See <https://www.npmjs.org/package/cordova> for a list of **Cordova**-specific commands with descriptions.
 - For **Ionic**:
 - emulate
 - run
 - prepare
 - serve
 - See <https://www.npmjs.org/package/ionic> for a list of **Ionic**-specific commands with descriptions.
5. From the Platform drop-down list, choose the platform for running on which the application is intended. The available options are:
 - Android
 - ios To emulate this platform, you need to install the [ios-sim command line tool](#) globally. You can do it through the **Node Package Manager (npm)**, see [Installing and Removing External Software Using Node Package Manager](#) or by running the `sudo npm install ios-sim -g` command, depending on your operating system.
 - amazon-fireos

- firefoxos
- blackberry10
- ubuntu
- wp8
- windows8
- browser

Learn more about targeted platforms at http://docs.phonegap.com/en/edge/guide_platforms_index.md.html#Platform%20Guides and http://cordova.apache.org/docs/en/4.0.0/guide_cli_index.md.html#The%20Command-Line%20Interface.

6. For **Cordova** and **Ionic**, specify the targeted virtual or physical Android device to run the application on: select the Specify Target check box and select the required device from the drop-down list. The list shows all the virtual and physical devices that are currently configured on our machine. See http://docs.phonegap.com/en/edge/guide_platforms_android_index.md.html#Android%20Platform%20Guide for details.
If PyCharm displays the following error message: **Cannot detect ios-sim in path**, make sure you have installed the `ios-sim`, see Before you start.
3. To run a **PhoneGap/Cordova/Ionic** application, select the required run configuration from the list on the main tool bar and then choose Run | Run <configuration name> on the main menu or click the Run toolbar button .

Markup Languages and Style Sheets

In this section:

- Markup Languages and Style Sheets
 - [Supported markup and template languages](#)
 - [Parsing Web contents](#)
- [Changing Color Values in Style Sheets](#)
- [CSS-Specific Refactorings](#)
- [Editing HTML Files](#)
- [Emmet Support](#)
- [Generating Instance Document From XML Schema](#)
- [Generating DTD](#)
- [Generating XML Schema From Instance Document](#)
- [Markdown Support](#)
- [Minifying CSS](#)
- [Navigating Between Edit Points](#)
- [Referencing XML Schemas and DTDs](#)
- [Compiling Sass, Less, and SCSS to CSS](#)
- [Compiling Stylus to CSS](#)
- [Using Stylelint Code Quality Tool](#)
- [Using Pug \(Jade\) Template Engine](#)
- [Validating Web Content Files](#)
- [Viewing Styles Applied to a Tag](#)
- [Viewing Images](#)
- [Working with Sass and SCSS in Compass Projects](#)
- [Live Editing of HTML, CSS, and JavaScript](#)
- [Viewing Actual HTML DOM](#)
- [Using JetBrains Chrome Extension](#)

Note In this part you will find information that is specific for the web content files only!

Supported markup and template languages

PyCharm supports editing of files in the following markup and template languages:

- XML
- HTML/XHTML
- **These features are supported in the Professional edition only.**
- CSS
- [Sass, SCSS](#)
- [Slim](#)
- [Less](#)
- [YAML](#)
- [Stylus](#)
- [Compass](#)
- [Handlebars expressions and Mustache templates](#)

The markup languages and style sheets are integrated into PyCharm and can use the most powerful editing features:

- Validation and syntax highlighting.
- **Code completion** (`Ctrl+Space`).
- **Indentation** (`Ctrl+Alt+I`, `Ctrl+Alt+L`).
- **Formatting** (`Ctrl+Alt+L`) according to the [code style](#).
- **Intention actions** (`Alt+Enter`).
- **Viewing code structure** (`Alt+7`).
- **Navigation in the source code** (`Ctrl+B`).
- **Integrated documentation** (`Ctrl+Q`).
- **Search for usages** (`Alt+F7`).
- **Commenting and uncommenting lines** (`Ctrl+Slash`, `Ctrl+Shift+Slash`).
- **Unwrapping and removing tags** (`Ctrl+Shift+Delete`).

All these features work if PyCharm successfully locates the DTD or schema file. In this case, all the files are validated against the DTD or schema, and the editing conveniences become available. Without a DTD or schema, only the well-formedness check is possible.

These features for web contents work same way as for the other source files. Refer to the respective topics of the [Advanced Editing Features](#) part for the detailed descriptions of procedures, and to [Keyboard shortcuts](#) .

Parsing Web contents

PyCharm parses Web contents files according to the following specifications:

- HTML: specification is configurable in the Default HTML language level in the [Schemas and DTDs](#) page of the Settings/Preferences dialog. By default, specification HTML 5.0 from W3C is assumed.
- CSS: specification CSS 3.0. The most common selectors are supported: universal selector `*`, type selectors `.a`, descendant selectors `.a.b`, child selectors `.a .b`, ID selectors `#b`, pseudo-classes and class selectors `DIV.warning`.
- PyCharm uses Xerces 2.11, an XML parser developed by Apache Software Foundation Group.

Changing Color Values in Style Sheets

On this page:

- [Introduction](#)
- [Choosing color value](#)
- [Changing color value](#)

Introduction

In addition to the editing techniques that are common for all file types, PyCharm provides specific techniques for CSS, Sass, SCSS and Less files.

In particular, it is possible to easily change color values without the necessity to memorize and type color codes.

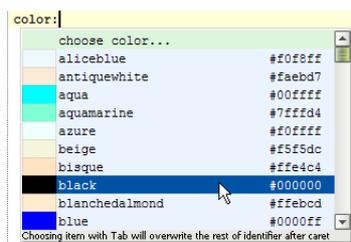
The `color` properties are marked with the icons of the corresponding color in the left gutter area of the editor. Use these icons to view colors and change color values. When you point with your mouse cursor to the color icon, the pop-up window appears, showing the color and its code.



Choosing color value

To choose color value in a style sheet

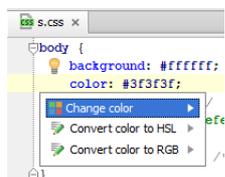
1. Open the desired style sheet for editing.
2. Type `color:` , and then press `Ctrl+Space`.
3. Select the desired color value from the suggestion list, or choose color... to pick a custom one:



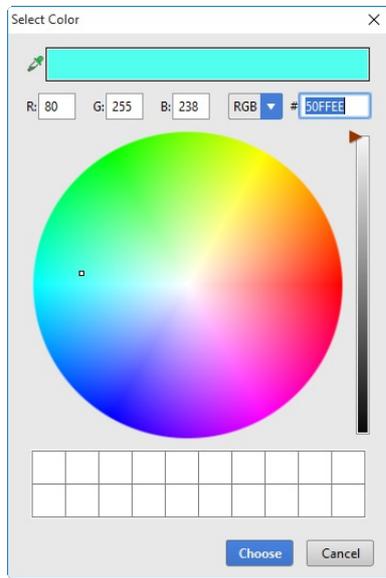
Changing color value

To change color value in a style sheet

1. Open the desired style sheet for editing, and locate color property to be changed.
2. Do one of the following:
 - If the color icon is **shown**, just double-click it in the left gutter of the editor.
 - If the color icon is **not shown**, then press `Alt+Enter`, or click the lightbulb icon to reveal the list of intention actions, and choose Change color intention action:



3. In the Choose color dialog box, pick the desired new color, and click Choose:



CSS-Specific Refactorings

In addition to the full range of [common refactorings](#), PyCharm provides the following CSS-specific refactorings:

- [Extract Variable for Sass](#)

Extract Variable for Sass

On this page:

- [Introduction](#)
- [Example](#)
- [Extracting a variable in-place](#)
- [Extracting a variable using the dialog box](#)

Introduction

You can replace a Sass expression with a local or global variable.

To perform this refactoring, you can use:

- [In-place refactoring](#). In this case you specify the new name right in the editor.
- [Refactoring dialog](#), where you specify all the required information. To make such a dialog accessible, you have to clear the check box [Enable in-place mode](#) in the editor settings.

Example

Before/After

```
$blue: #3bbfce
$margin: 16px

.border
padding: $margin / 2
margin: $margin / 2
border-color: $blue
```

```
$blue: #3bbfce
$margin: 16px;
$var: $margin / 2

.border
padding: $var
margin: $var
border-color: $blue
```

Extracting a variable in-place

To extract a variable using in-place refactoring

1. In the editor, select the expression to be replaced with a variable. You can do that yourself or use the **smart expression selection** feature to let PyCharm help you. So, do one of the following:
 - Highlight the expression. Then choose Refactor | Extract | Variable on the main menu or on the context menu. Alternatively, press `Ctrl+Alt+V`.
 - Place the cursor before or within the expression. Choose Refactor | Extract Variable on the main menu or on the context menu. or press `Ctrl+Alt+V`. In the Expressions pop-up menu, select the expression. To do that, click the required expression. Alternatively, use the Up and Down arrow keys to navigate to the expression of interest, and then press `Enter` to select it.

Note The Expressions pop-up menu contains all the expressions appropriate for the current cursor position in the editor.

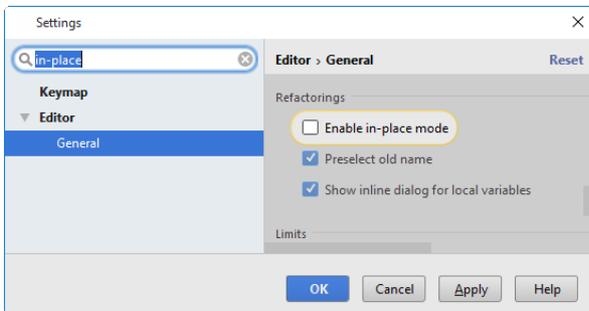
When you navigate through the suggested expressions in the pop-up, the code highlighting in the editor changes accordingly.

2. If more than one occurrence of the selected expression is found, select Replace this occurrence only or Replace all occurrences in the Multiple occurrences found pop-up menu. To select the required option, just click it. Alternatively, use the `Up` and `Down` arrow keys to navigate to the option of interest, and press `Enter` to select it.
3. Select the place in the source code, where the new variable will be declared. The declaration can be global (a variable is available throughout the whole file), or local (a variable is declared immediately before use, and is available in the current block only).
4. Specify the name of the variable. Do one of the following:
 - Select one of the suggested names from the pop-up list. To do that, double-click the suitable name. Alternatively, use the `Up` and `Down` arrow keys to navigate to the name of interest, and `Enter` to select it. When finished, press `Escape`.
 - Edit the name by typing. The name is shown in the box with red borders and changes as you type. When finished, press `Escape`.

Extracting a variable using the dialog box

To extract a variable using the dialog box

If the Enable in place refactorings check box is cleared in the Editor settings, the Extract Variable refactoring is performed by means of the dialog box.

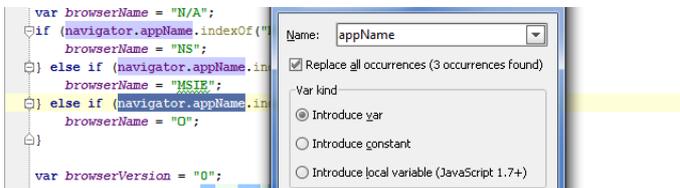


1. Select the desired expression, and invoke Extract Variable refactoring as described [above](#).

2. If more than one expression is detected for the current cursor position, the Expressions list appears. If this is the case, select the required expression. To do that, click the expression. Alternatively, use the `Up` and `Down` arrow keys to navigate to the expression of interest, and then press `Enter` to select it.

3. In the [Extract Variable dialog for Sass](#):

- Specify the variable name. You can select one of the suggested names from the list or type the name in the Name field.
- Specify the place for declaration. Select the desired place (global or local) from the drop-down list.
- If more than one occurrence of the selected expression is found, you can select to replace all the found occurrences by selecting the corresponding check box. If you want to replace only the current occurrence, clear the Replace all occurrences check box.
- Click OK.

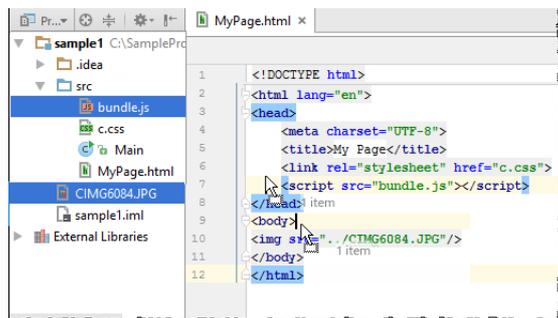


Editing HTML Files

PyCharm enables you to [create HTML files](#), and edit them.

When a file is opened for editing, you can drag a JavaScript or CSS file into the opened HTML file. PyCharm will automatically create a script or a link tag inside the `<head>` tag.

Same way, you can drag an image to add an `` tag with the `width` and `height` attributes anywhere inside the `<body>` tag.



Emmet Support

With PyCharm, you can edit HTML and CSS code faster by applying **Emmet** features. Just type an [Emmet abbreviation](#) in HTML and press Tab to expand it into the markup. **Emmet** also works in the **CSS** and **JSX** context.

PyCharm provides two types of **Emmet** support.

- Native
- Support of additional templates

In this section:

- [Enabling Emmet Support](#)
- [Expanding Emmet Templates with User Defined Templates](#)
- [Configuring Abbreviation Expansion Key](#)
- [Enabling Support of Additional Live Templates](#)
- [Surrounding a Code Block with an Emmet Template](#)

Enabling Emmet Support

In this section:

- [Basics](#)
- [Enabling and configuring native Emmet support in the HTML or XML context](#)
- [Enabling native Emmet support in the JavaScript context](#)
- [Enabling and configuring native Emmet support in the CSS context](#)

Basics

Native Emmet support allows you to generate XML/HTML, JavaScript (JSX Harmony) and CSS structures based on [abbreviations](#). PyCharm supports basic **Emmet** and [Emmet version 1.1](#) features, such as:

- New syntax for writing RGBA colors.
- Implied attributes.
- Default attributes.
- Boolean attributes.
- The **Update Tag** action.

Emmet is supported in HTML/XML, JavaScript (JSX Harmony) and in the CSS contexts. This support is configured separately on the [Emmet. HTML](#), [Emmet. JSX](#) and [Emmet. CSS](#) pages respectively.

Enabling and configuring native Emmet support in the HTML or XML context

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS. Expand the Editor node, and then click XML under Emmet. The [Emmet](#) page opens.
2. To enable the **Emmet** support in the HTML or XML context, select the Enable XML/HTML Emmet check box. When this check boxes is cleared, all the other controls on this page become disabled.
3. To have PyCharm show a pop-up window with a preview of the entered abbreviation before actually expanding it, select the Enable abbreviation preview check box.
4. Specify how Emmet in PyCharm will treat URL addresses by selecting or clearing the Enable automatic URL recognition while wrapping text with <a> tag check box.

- If this check box is cleared and you attempt to wrap an URL address with the <a> tag, PyCharm simply encloses the URL address in `` and positions the cursor inside the double quotes in the `href` attribute. For example, wrapping `http://www.jetbrains.com` will result in `http://www.jetbrains.com`:

```
<a href="|">http://www.jetbrains.com</a>
```

- If this check box is selected and you attempt to wrap an URL address with the <a> tag, PyCharm inserts the URL address inside the double quotes as the value of the `href` attribute and encloses the URL in `<a href="<wrapped URL>">`. For example, wrapping `http://www.jetbrains.com` will result in `http://www.jetbrains.com`. Moreover, PyCharm highlights the wrapped URL green as a recognized URL:

```
<a href="http://www.jetbrains.com">http://www.jetbrains.com</a>
```

Enabling native Emmet support in the JavaScript context

This feature is supported in the Professional edition only.

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS.
2. Under the Languages and Frameworks node, click [JavaScript](#), and select the language level JSX Harmony.
3. Expand the Editor node, and then click JSX under Emmet. The [JSX](#) page opens.
4. To enable the **Emmet** support in the JavaScript context, select the Enable JSX Emmet check box.

Enabling and configuring native Emmet support in the CSS context

This feature is supported in the Professional edition only.

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS. Expand the Editor node, and then click CSS under Emmet. The [Emmet. CSS](#) page opens.
2. To enable the **Emmet** support in the CSS context, select the Enable CSS Emmet check box. When this check boxes is cleared, all the other controls on this page become disabled.
3. Configure the way unknown abbreviations are treated by selecting or clearing the Enable fuzzy search among CSS abbreviations check box: When this check box is selected, every unknown abbreviation will be scored against available template names. The match with the best score will be used to resolve the template. For example, with this option enabled, the following abbreviations can be equal to:

- `ov:h`
- `ov-h`
- `o-h`
- `oh`

4. Configure the way unrecognized properties are treated by selecting or clearing the Enable expansion of unknown properties ('unknown' to 'unknown;') check box:

- When this check box is selected, any entered word will be expanded into the same word followed with a colon and a semicolon;

- When this check box is cleared, only known properties (for example, `color`) will be expanded this way (`color::;`)
5. Configure inserting browser-specific prefixes using the Auto insert CSS vendor prefixes check box: If this check box is selected, the CSS properties listed in the table below are expanded into constructs that contain pre-pending vendor prefixes. Learn more at [Vendor prefixes](#). If this check box is cleared, the entire table of properties is disabled.
6. Configure the use of properties in different browsers using the Properties and vendor prefixes table. The table contains a list of CSS properties and vendor prefixes that correspond to various browsers.
- To enable or disable a property in a browser, select or clear the check box under the browser column.
 - To add a new property to the list, click the Add button `+` or press `Alt+Insert` . Then type the name of the property in the dialog box that opens and enable it in the relevant browsers.
 - To delete one or more properties from the list, select them and press Remove `-` or press `Alt+Delete` .

Expanding Emmet Templates with User Defined Templates

In this section:

- [Introduction](#)
- [Expanding Emmet templates](#)

Introduction

You can expand Emmet templates with your own live templates.

Suppose you have a template entry with the following template text:

```
<entry type="$TYPES$">$END$ <entry>
```

To generate a list of entries, you just need to type `"entry-list<entry[number=$]*5"` and press `TAB`. By default, the `number` attribute will be generated before `type`. To customize the position where it is generated, you need to add the `ATTRS` variable to your template, for example:

```
<entry type="$TYPES$" $ATTRS$>$END$ <entry>
```

The `ATTRS` variable must have an empty string as the default value and should be skipped.

Expanding Emmet templates

Warning! Emmet support for CSS and XSL is provided in the Professional edition only!

To expand an Emmet template with a user-defined template

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing `File | Settings for Windows and Linux or PyCharm | Preferences for macOS`, and click `Live Templates` under `Editor`.
2. On the [Live Templates](#) page that opens, expand the required `Zen Coding` template group, for example, `Zen HTML`, `Zen CSS`, or `Zen XSL`.
3. Select the template to expand. The focus moves to the `Template Text` area where the fields show the settings of the selected template.
4. In the `Template Text` field, add the required text and variables to the template body.
5. Click the `Edit Variables` button. In the [Edit Template Variables](#) dialog box that opens, specify the default variable values in the `Default value` field and select the `Skip if defined` check box, where necessary.

Shortcut key for expanding Emmet selectors and live templates is configurable. You can re-define this default setting for each specific live template.

Configuring a shortcut to expand abbreviations

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Emmet under Editor.
2. On the [Emmet](#) page that opens, choose the desired option from the Expand abbreviation with drop-down list.

Configuring a shortcut to expand a live template with

1. To re-define the expansion key for a live template, open the [Live Templates](#) page, expand one of the Zen Coding nodes, and select the desired template. The focus moves to the [Template Text](#) area.
2. From the Expand with drop-down list, select the key to expand the template with.
This setting does not override the default setting specified for native Emmet support; you will just get the ability to expand the template using either of the specified keys.

Enabling Support of Additional Live Templates

Support of additional templates includes more than 200 different HTML, CSS, and XSL live templates. All of them are listed under the Emmet nodes on the [Live Templates](#) page of the Settings/Preferences dialog box.

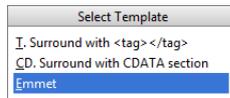
Warning! Emmet support for CSS and XSL is provided in the Professional edition only!

To enable support of additional HTML, CSS, and XSL live templates

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Live Templates under Editor.
2. On the [Live Templates](#) page that opens, select the check boxes next to the relevant template groups.

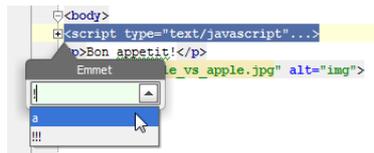
To surround a block of code with an Emmet template, follow these steps

1. Open the desired file for editing and select a block of code to be surrounded.
2. Press `Ctrl+Alt+J`, or choose Code | Surround with | Live Template on the main menu .
3. In the Select Template pop-up menu, choose Emmet:



4. Type the desired Emmet abbreviation, and press `Enter`.

Note the drop-down list to the right. Clicking the down arrow reveals a history list of the recently applied Emmet live templates:



Also mind the color indication. If you type a valid Emmet abbreviation, the background is green. However, when a non-existent abbreviation is entered, the background becomes red:



Warning! Emmet support for JavaScript, CSS and XSL is provided in the Professional edition only!

To generate an XML instance document from an XML Schema

1. With the desired Schema (`.xsd`) file opened in the active editor tab, choose Tools | XML Actions | Generate XML Document from XSD Schema on the main menu.
2. In the [Generate Instance Document from Schema](#) dialog box that opens configure the XML instance document generation procedure:
 - In the Schema Path text box, specify the location of the Schema to base the XML document generation on. By default, the field shows the full path to the current file. Accept this suggestion or click the Browse button  and select the desired file in the [dialog that opens](#).
 - In the Instance Document Name text box, specify the name of the output file to place the generated XML in.

Warning! PyCharm suggests the name of the source XML document with the `.xml` extension. If you type another name, make sure the extension is correct.
 - Specify the location of the generated document. By default, it will be placed in the same directory as the source Schema file. To specify another location, click the Browse button  and select the desired path in the [dialog that opens](#).
 - From the Element Name drop-down list, select the local name of the global element to be used as the root of the generated XML document.
 - Specify whether to take restriction and uniqueness particles into consideration by selecting the corresponding check boxes.

Generating DTD

A [Data Type Definition \(DTD\)](#) is required for running [structure validation checks](#) on a Web content file. PyCharm can scan any XML file for the existing elements and attributes and generate a DTD for it.

To generate a DTD for an XML file

1. Open the desired file in the active editor tab.
2. On the main menu, choose Tools | XML Actions | Generate DTD From XML File. The resulting DTD is added as a section above the first line of the file.

Generating XML Schema From Instance Document

An [XSD \(XML Schema Definition\) Schema](#) is required for running [structure validation checks](#) on a Web content file. PyCharm can scan any XML file for the existing elements and attributes and generate a Schema for it.

To have a Schema generated based on an XML instance document

1. With the desired XML document opened in the active editor tab, choose Tools | XML Actions | Generate XSD Schema from XML File on the main menu.

2. In the [Generate Schema From Instance Document](#) dialog box that opens configure the Schema generation procedure:

- In the Instance Document Path text box, specify the location of the file to be used as the basis for Schema generation. By default, the field shows the full path to the current file. Accept this suggestion or click the Browse button  and select the desired file in the [dialog that opens](#).
- In the Result Schema File Name text box, specify the name of the output file to place the generated Schema in.

Warning! PyCharm suggests the name of the source XML document with the `.xsd` extension. If you type another name, make sure the extension is correct.

- Specify the location of the generated Schema. By default, the generated Schema file will be placed in the same directory as the source XML instance document. To specify another location, click the Browse button  and select the desired path in the [dialog that opens](#).
- From the Design Type drop-down list, select the way to declare elements and complex types.
- From the Detect Simple Content Type drop-down list, select the type to use for leaf text.
- In the Detect Enumerations Limit text box, type the number of occurrences to cause the Schema enumeration appearance.

Tip To suppress Schema enumeration, specify 0.

Markdown Support

Warning! The following is only valid when Markdown Support Plugin is installed and enabled!

On this page:

- [Introduction](#)
- [Prerequisites](#)
- [Changes to the UI](#)
- [Creating a Markdown file](#)
- [Markdown editor](#)

Introduction

PyCharm makes it possible to work with the [Markdown](#) files.

The Markdown files are marked with  icon.

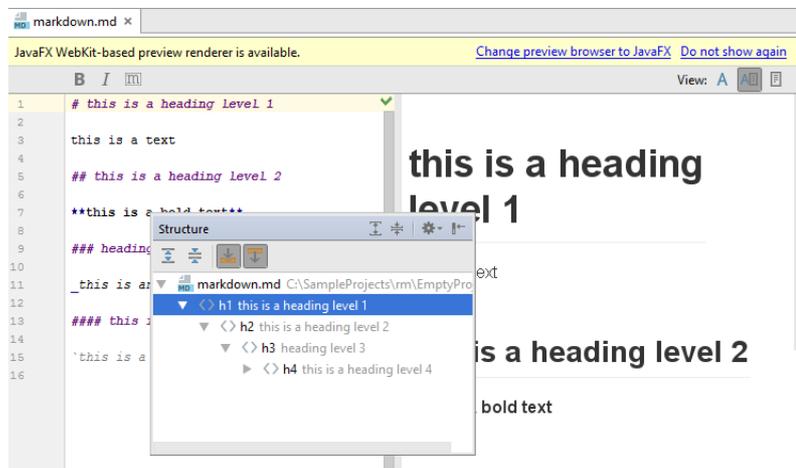
Prerequisites

Before you start working with Markdown, make sure that Markdown Support plugin is [installed and enabled](#). The plugin is not bundled with PyCharm.

Changes to the UI

With the Markdown Support enabled, The page [Markdown](#) appears in the Languages and Frameworks section of the Settings/Preferences dialog.

Note also, that [Structure view](#) shows the headings of the various levels:



Creating a Markdown file

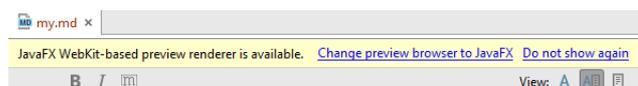
To create a Markdown file, follow these steps

1. Do one of the following:
 - Choose File | New on the main menu.
 - Right-click the target directory where the new file should be created, and choose New on the context menu.
 - Press `Alt+Insert`
2. Choose File.
3. In the New File dialog box, specify the new file name and extension `.md`.

The new file `<name>.md`, marked with  icon, is created and opens for editing.

Markdown editor

The editor of a `<name>.md` file by default shows the following:



Click one of the links to get rid of the banner. It's recommended to choose the link [Change preview browser to JavaFX](#).

The editor is divided into two panes: the editor itself and the preview. Each of the panes can be hidden.

Editor pane

Icon	Action	Description
	Toggle bold mode	Inserts two asterisks before and after the selected text to render bold font.

<i>I</i>	Toggle italic mode	Inserts underscores before and after the selected text to render italic font.
<code>code</code>	Toggle monospaced (code span) mode	Inserts single apostrophes before and after the selected text to render monospaced font.
Preview pane		
A	Show editor only	Shows editor only with Markdown syntax.
A 	Show editor and preview	Shows editor with Markdown syntax and the corresponding preview. The results of editing are immediately reflected in the preview pane.
	Show preview only	Shows preview that renders the Markdown syntax. Editing is not possible, and the buttons B , <i>I</i> and <code>code</code> are disabled.

Minifying CSS

This feature is supported in the Professional edition only.

On this page:

- [Introduction](#)
- [Installing and configuring the YUI Compressor](#)
- [Creating a file watcher](#)
- [Minifying the code](#)

Introduction

The term **minification** or **compression** in the context of CSS means removing all unnecessary characters, such as **spaces**, **new lines**, **comments** without affecting the functionality of the source code.

These characters facilitate working with the code at the development and debugging stage by improving the code readability. However at the production stage these characters become extraneous: being insignificant for code execution, they increase the size of code to be transferred. Therefore it is considered good practice to remove them before deployment.

PyCharm supports integration with the [YUI Compressor](#) CSS minification tool.

In PyCharm, minifier configurations are called **File Watchers**. For each supported minifier, PyCharm provides a predefined **File Watcher** template. Predefined **File Watcher** templates are available at the PyCharm level. To run a minifier against your project files, you need to create a project-specific **File Watcher** based on the relevant template, at least, specify the path to the minifier to use on your machine.

Installing and configuring the YUI Compressor

1. Download and install [Node.js](#). The runtime environment is required for two reasons:
 - The CSS minifier is started through **Node.js**.
 - **NPM**, which is a part of the runtime environment, is also the easiest way to download the CSS minifier.

If you are going to use the command line mode, make sure the path to the parent folder of the **Node.js** executable file and the path to the `npm` folder are added to the `PATH` variable. This enables you to launch the CSS minifier and `npm` from any folder.

2. Install and enable the **NodeJS** repository plugin as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).
3. Download and install the JavaScript minification tool. The easiest way is to use the **Node Package Manager (npm)**, which is a part of [Node.js](#).
 1. Switch to the directory where the **Node Package Manager (npm)** is stored or define a `path` variable for it so it is available from any folder.
 2. Type the following command at the command line prompt:

```
npm install yuicompressor
```

If you use the **Node Package Manager (npm)**, the minifier is installed under **Node.js** so **Node.js**, which is required for starting the tool, will be specified in the path to it.

Creating a file watcher

PyCharm provides a common procedure and user interface for creating **File Watchers** of all types. The only difference is in the predefined templates you choose in each case.

1. To start creating a **File Watcher**, open the Settings/Preferences dialog box by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS on the main menu, and then click File Watchers under the Tools node. The [File Watchers page](#) that opens, shows the list of **File Watchers** that are already configured in the project.
2. Click the Add button  or press `Alt+Insert` and choose the YUI Compressor CSS predefined template from the pop-up list.
3. In the Program text box, specify the path to the `yuicompressor-<version>.jar` file. If you installed the tool through the **Node Package Manager**, PyCharm locates the required file itself and fills in the field automatically. Otherwise, type the path manually or click the Browse button  and choose the file location in the dialog box that opens.
4. Proceed as described on page [Using File Watchers](#).

Minifying the code

When a **Minification File Watcher** is enabled (see [Enabling and disabling File Watchers](#)), minification starts automatically as soon as a file to which compilation is applicable is changed or saved, see [Configuring the behaviour of the File Watcher](#).

PyCharm creates a separate file with the generated output. The file has the name of the source **CSS** file and the extension `min.css`. The location of the generated file is defined in the Output paths to refresh text box of the [New Watcher dialog](#). However, in the Project Tree, it is shown under the source **CSS** file which is now displayed as a node.

Navigating Between Edit Points

In the HTML and XML context, you can navigate between **edit points**, that is, between important points of code where editing is possible.

- To move the cursor to the previous edit point, choose Navigate | Previous Emmet Edit Point, or press `Shift+Alt+Open Bracket`.
- To move the cursor to the next edit point, choose Navigate | Next Emmet Edit Point, or press `Shift+Alt+Close Bracket`.

Referencing XML Schemas and DTDs

Your XML file may reference an external XML schema (XSD) or DTD file, e.g.

```
<root xmlns="http://www.example.org"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.example.org http://www.example.org/xsds/example.xsd">
```

or

```
<!DOCTYPE root SYSTEM "http://www.example.org/dtds/example.dtd">
```

If the referenced URL or the namespace URI is "unfamiliar", it's marked as an error.

For situations like this, PyCharm provides the following [intention actions](#):

- Fetch External Resource. PyCharm downloads the referenced file and associates it with the URL (or the namespace URI). The error highlighting disappears. The XML file is validated according to the downloaded schema or DTD. (The associations of the URLs and the namespace URIs with the schema and DTD files are shown on the [Schemas and DTDs page](#) in the [Settings dialog](#).)
- Manually Setup External Resource. Use this option when you already have an appropriate schema or DTD file available locally. The [Map External Resource dialog](#) will open and you'll be able to select the file for the specified URL or namespace URI. The result of the operation is the same as in the case of fetching the resource.
- Ignore External Resource. The URL or the namespace URI is added to the Ignored Schemas and DTDs list. (This list is shown on the [Schemas and DTDs page](#) in the [Settings dialog](#).) The error highlighting disappears. PyCharm won't validate the XML file, however, it will check if the XML file is well-formed.

There is one more intention action that you may find useful: Add Xsi Schema Location for External Resource. This intention action lets you complete your root XML elements. If the namespace is already specified, PyCharm can add a couple of missing attributes.

For example, if you have a fragment like this:

```
<root xmlns="http://www.example.org">
```

and perform the intention action on the value of the `xmlns` attribute, the result will be:

```
<root xmlns="http://www.example.org"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.example.org >
```

At this step, you'll be able to add the schema URL, and then map the URL (or the namespace URI) onto an appropriate schema file, or add the URL (or the URI) to the Ignored Schemas and DTDs list.

Compiling Sass, Less, and SCSS to CSS

This feature is supported in the Professional edition only.

On this page:

- [Introduction](#)
- [Before you start](#)
- [Installing the Sass/SCSS compiler](#)
- [Installing the Less compiler](#)
 - [Installing the Less compiler globally](#)
 - [Installing the Less compiler in a project](#)
- [Creating a file watcher](#)
- [Compiling the code](#)

Introduction

Sass, Less, and SCSS code is not processed by browsers that work with CSS code. Therefore to be executed, Sass, Less, or SCSS code has to be translated into CSS. This operation is referred to as **compilation** and the tools that perform it are called **compilers**.

PyCharm supports integration with compiler tools that translate Sass, Less, and SCSS code into CSS.

In PyCharm, compiler configurations are called **File Watchers**. For each supported compiler, PyCharm provides a predefined **File Watcher** template. Predefined **File Watcher** templates are available at the PyCharm level. To run a compiler against your project files, you need to create a project-specific **File Watcher** based on the relevant template, at least, specify the path to the compiler to use on your machine.

Before you start

1. Make sure the **Less Support**, **Sass Support**, and **CSS Support** plugins are activated. The plugins are bundled with PyCharm and activated by default. If the plugins are not activated, enable them on the **Plugins** page of the **Settings / Preferences Dialog** as described in [Enabling and Disabling Plugins](#).
2. Install and enable the **NodeJS** repository plugin as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).
3. Download and install the [Node.js](#) runtime environment.
4. Configure the Node.js interpreter in PyCharm:
 1. Open the by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Node.js and NPM under Languages & Frameworks.
 2. On the [Node.js and NPM](#) page that opens, specify the location of the desired Node.js interpreter.

See [Configuring Node.js Interpreters](#) for details.

Installing the Sass/SCSS compiler

Sass and SCSS compilers are managed through the **Ruby Gem manager**.

1. [Download and install Ruby](#).
2. Specify a `path` variable for the folder where the Ruby executable file and the `gem.bat` file are stored. This lets you launch **Ruby** and **Gem Manager** from any folder and ensures that **Ruby** is successfully launched during compilation.
3. Type the following command at the command line prompt:

```
gem install sass
```

The tool is installed to the folder where Ruby executable file and the `gem.bat` file are stored.

Installing the Less compiler

The easiest way to install the Less compiler is to use the **Node Package Manager (npm)**, which is a part of [Node.js](#). See [Installing and Removing External Software Using Node Package Manager](#) for details.

Depending on the desired location of the Less compiler executable file, choose one of the following methods:

- Install the compiler **globally** at the PyCharm level so it can be used in any PyCharm project.
- Install the compiler in a specific project and thus restrict its use to this project.
- Install the compiler in a project as a [development dependency](#).

In either installation mode, make sure that the parent folder of the Less compiler is added to the `PATH` variable. This enables you to launch the compiler from any folder.

PyCharm provides user interface both for **global** and **project** installation as well as supports installation through the command line.

Installing the Less compiler globally

Global installation makes a compiler available at the PyCharm level so it can be used in any PyCharm project. Moreover, during installation the parent folder of the compiler is automatically added to the `PATH` variable, which enables you to launch the compiler from any folder.

- Run the installation from the command line in the **global** mode:
 1. Launch the embedded Terminal (View | Tool Windows | Terminal or by hovering your mouse pointer over  in the lower left corner of PyCharm and choosing Terminal from the menu
 2. Switch to the directory where **NPM** is stored or define a `PATH` variable for it so it is available from any folder, see [Installing NodeJs](#).
 3. Type the following command at the command line prompt:

```
npm install -g less
```

The `-g` key makes the compiler run in the **global** mode. Because the installation is performed through **NPM**, the Less compiler is installed in the `npm` folder. Make sure this parent folder is added to the `PATH` variable. This enables you to launch the compiler from any folder.

For more details on the **NPM** operation modes, see [npm documentation](https://npmjs.org/package/less). For more information about installing the Less compiler, see <https://npmjs.org/package/less>.

- Run **NPM** from PyCharm using the Node.js and NPM page of the Settings dialog box.
 1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Node.js and NPM under Languages & Frameworks.
 2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the **global** and at the **project** level. Click `+`.
 3. In the Available Packages dialog box that opens, select the required package to install.
 4. Select the Options check box and type `-g` in the text box next to it.
 5. Optionally specify the product version and click Install Package to start installation.

Installing the Less compiler in a project

Local installation in a specific project restricts the use of a compiler to this project.

- Run the installation from the command line:
 1. Launch the embedded Terminal (View | Tool Windows | Terminal or by hovering your mouse pointer over  in the lower left corner of PyCharm and choosing Terminal from the menu
 2. Switch to the project root folder and type the following command at the command line prompt:

```
npm install less
```

- Run **NPM** from PyCharm using the Node.js and NPM page of the Settings dialog box.
 1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Node.js and NPM under Languages & Frameworks.
 2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the **global** and at the **project** level. Click `+`.
 3. In the Available Packages dialog box that opens, select the required package.
 4. Optionally specify the product version and click Install Package to start installation.

Project level installation is helpful and reliable in [template-based projects](#) of the type **Node Boilerplate** or **Node.js Express**, which already have the `node_modules` folder. The latter is important because **NPM** installs the Less compiler in a `node_modules` folder. If your project already contains such folder, the Less compiler is installed there.

Projects of other types or **empty** projects may not have a `node_modules` folder. In this case **npm** goes upwards in the folder tree and installs the Less compiler in the first detected `node_modules` folder. Keep in mind that this detected `node_modules` folder may be **outside** your current project root.

Finally, if no `node_modules` folder is detected in the folder tree either, the folder is created right under the current project root and the Less compiler is installed there.

In either case, make sure that the parent folder of the Less compiler is added to the `PATH` variable. This enables you to launch the compiler from any folder.

Creating a file watcher

PyCharm provides a common procedure and user interface for creating **File Watchers** of all types. The only difference is in the predefined templates you choose in each case.

1. Make sure the **File Watchers** plugin is enabled. The plugin is bundled with PyCharm and activated by default. If it is not, enable the plugin. See [Enabling and Disabling Plugins](#) for details.
2. To start creating a **File Watcher**, open the Settings/Preferences dialog box by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS on the main menu, and then click File Watchers under the Tools node. The [File Watchers page](#) that opens, shows the list of **File Watchers** that are already configured in the project.
3. Click the Add button `+` or press `Alt+Insert`. Depending on the tool you are going to use, choose the appropriate predefined template from the pop-up list
 - Less
 - Sass
 - SCSS
4. In the Program text box, specify the path to the compiler executable file or archive depending on the chosen predefined template.
 - `lessc.cmd` for Less If you installed the tool through the **Node Package Manager**, PyCharm locates the required file itself at `<node.js_home>/node_modules/bin/lessc.cmd` and fills in the field automatically. Otherwise, type the path manually or click the Browse button  and choose the file location in the dialog box that opens.
 - `sass.bat` for Sass
 - `scss.bat` for SCSS

If you installed the **Sass** and **SCSS** tools through **Ruby**, PyCharm locates the required files itself at `<ruby_home>/bin/sass.bat` `<ruby_home>/bin/scss.bat` respectively and fills in the field automatically. Otherwise, type the path manually or click the Browse button  and choose the file location in the dialog box that opens.

5. Proceed as described on page [Using File Watchers](#).

Compiling the code

When you open a Less, Sass, or SCSS file, PyCharm checks whether an applicable file watcher is available in the current project. If such file watcher is configured but disabled, PyCharm displays a pop-up window that informs you about the configured file watcher and suggests to enable it.

If an applicable file watcher is configured and enabled in the current project, PyCharm starts it automatically upon the event specified in the [New Watcher dialog](#).

- If the Immediate file synchronization check box is selected, the **File Watcher** is invoked as soon as any changes are made to the source code.
- If the Immediate file synchronization check box is cleared, the **File Watcher** is started upon save (File | Save All, ) or when you move focus from PyCharm (upon frame deactivation).

PyCharm creates a separate file with the generated output. The file has the name of the source **Sass**, **Less**, or **SCSS** file and the extension `css`. The location of the generated files is defined in the Output paths to refresh text box of the [New Watcher dialog](#). However, in the Project Tree, they are shown under the source file which is now displayed as a node.

Compiling Stylus to CSS

This feature is supported in the Professional edition only.

On this page:

- [Introduction](#)
- [Before you start](#)
- [Installing the Stylus compiler globally](#)
- [Installing the Stylus compiler in a project](#)
- [Creating a file watcher](#)
- [Compiling the code](#)

Introduction

[Stylus](#) code is not processed by browsers that work with CSS code. Therefore to be executed, Stylus code has to be translated into CSS. This operation is referred to as **compilation** and the tools that perform it are called **compilers**.

PyCharm supports integration with a compiler tool that translates **Stylus** code into CSS.

In PyCharm, compiler configurations are called **File Watchers**. For each supported compiler, PyCharm provides a predefined **File Watcher** template. Predefined **File Watcher** templates are available at the PyCharm level. To run a compiler against your project files, you need to create a project-specific **File Watcher** based on the relevant template, at least, specify the path to the compiler to use on your machine.

The easiest way to install the Stylus compiler is to use the **Node Package Manager (npm)**, which is a part of [Node.js](#). See [Installing and Removing External Software Using Node Package Manager](#) for details.

Depending on the desired location of the Stylus compiler executable file, choose one of the following methods:

- Install the compiler **globally** at the PyCharm level so it can be used in any PyCharm project.
- Install the compiler in a specific project and thus restrict its use to this project.
- Install the compiler in a project as a [development dependency](#).

In either installation mode, make sure that the parent folder of the Stylus compiler is added to the `PATH` variable. This enables you to launch the compiler from any folder.

PyCharm provides user interface both for **global** and **project** installation as well as supports installation through the command line.

Before you start

1. Download and install [Node.js](#). The runtime environment is required for two reasons:
 - The Stylus compiler is started through **Node.js**.
 - **NPM**, which is a part of the runtime environment, is also the easiest way to download the Stylus compiler.

If you are going to use the command line mode, make sure the path to the parent folder of the **Node.js** executable file and the path to the `npm` folder are added to the `PATH` variable. This enables you to launch the Stylus compiler and `npm` from any folder.

2. Install and enable the **NodeJS** repository plugin as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).

Installing the Stylus compiler globally

Global installation makes a compiler available at the PyCharm level so it can be used in any PyCharm project. Moreover, during installation the parent folder of the compiler is automatically added to the `PATH` variable, which enables you to launch the compiler from any folder.

– Run the installation from the command line in the **global** mode:

1. Launch the embedded Terminal (View | Tool Windows | Terminal or by hovering your mouse pointer over  in the lower left corner of PyCharm and choosing Terminal from the menu
2. Switch to the directory where **NPM** is stored or define a `PATH` variable for it so it is available from any folder, see [Installing NodeJS](#).
3. Type the following command at the command line prompt:

```
npm install -g stylus
```

The `-g` key makes the compiler run in the **global** mode. Because the installation is performed through **NPM**, the Stylus compiler is installed in the `npm` folder. Make sure this parent folder is added to the `PATH` variable. This enables you to launch the compiler from any folder.

For more details on the **NPM** operation modes, see [npm documentation](#). For more information about installing the Stylus compiler, see <https://npmjs.org/package/stylus>.

– Run **NPM** from PyCharm using the Node.js and NPM page of the Settings dialog box.

1. Open the [Settings / Preferences Dialog](#) by pressing `(Ctrl+Alt+S)` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Node.js and NPM under Languages & Frameworks.
2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the **global** and at the **project** level. Click `+`.
3. In the Available Packages dialog box that opens, select the required package to install.
4. Select the Options check box and type `-g` in the text box next to it.
5. Optionally specify the product version and click Install Package to start installation.

Installing the Stylus compiler in a project

Local installation in a specific project restricts the use of a compiler to this project.

– Run the installation from the command line:

1. Launch the embedded Terminal (View | Tool Windows | Terminal or by hovering your mouse pointer over  in the lower left corner of PyCharm and choosing Terminal from the menu
2. Switch to the project root folder and type the following command at the command line prompt:

```
npm install stylus
```

– Run **NPM** from PyCharm using the Node.js and NPM page of the Settings dialog box.

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Node.js and NPM under Languages & Frameworks.
2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the **global** and at the **project** level. Click **+**.
3. In the Available Packages dialog box that opens, select the required package.
4. Optionally specify the product version and click Install Package to start installation.

Project level installation is helpful and reliable in [template-based projects](#) of the type **Node Boilerplate** or **Node.js Express**, which already have the `node_modules` folder. The latter is important because **NPM** installs the Stylus compiler in a `node_modules` folder. If your project already contains such folder, the Stylus compiler is installed there.

Projects of other types or **empty** projects may not have a `node_modules` folder. In this case **npm** goes upwards in the folder tree and installs the Stylus compiler in the first detected `node_modules` folder. Keep in mind that this detected `node_modules` folder may be **outside** your current project root.

Finally, if no `node_modules` folder is detected in the folder tree either, the folder is created right under the current project root and the Stylus compiler is installed there.

In either case, make sure that the parent folder of the Stylus compiler is added to the `PATH` variable. This enables you to launch the compiler from any folder.

Creating a file watcher

PyCharm provides a common procedure and user interface for creating **File Watchers** of all types. The only difference is in the predefined templates you choose in each case.

1. Make sure the **File Watchers** plugin is enabled. The plugin is bundled with PyCharm and activated by default. If it is not, enable the plugin. See [Enabling and Disabling Plugins](#) for details.
2. To start creating a **File Watcher**, open the Settings/Preferences dialog box by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS on the main menu, and then click File Watchers under the Tools node. The [File Watchers page](#) that opens, shows the list of **File Watchers** that are already configured in the project.
3. Click the Add button **+** or press `Alt+Insert` and choose the Stylus predefined template from the pop-up list.
4. In the Program text box, specify the path to the executable file:
 - `stylus` for macOS and Unix.
 - `stylus.bat` for Windows.

Type the path manually or click the Browse button  and choose the file location in the dialog box that opens.

5. Proceed as described on page [Using File Watchers](#).

Compiling the code

When you open a Stylus file, PyCharm checks whether an applicable file watcher is available in the current project. If such file watcher is configured but disabled, PyCharm displays a pop-up window that informs you about the configured file watcher and suggests to enable it.

If an applicable file watcher is configured and enabled in the current project, PyCharm starts it automatically upon the event specified in the [New Watcher dialog](#).

- If the Immediate file synchronization check box is selected, the **File Watcher** is invoked as soon as any changes are made to the source code.
- If the Immediate file synchronization check box is cleared, the **File Watcher** is started upon save (File | Save All, `Ctrl+S`) or when you move focus from PyCharm (upon frame deactivation).

PyCharm creates a separate file with the generated output. The file has the name of the source **Stylus** file and the extension `css`. The location of the generated files is defined in the Output paths to refresh text box of the [New Watcher dialog](#). However, in the Project Tree, they are shown under the source file which is now displayed as a node.

Using Stylelint Code Quality Tool

This feature is supported in the Professional edition only.

PyCharm provides facilities to run CSS-specific code quality inspections through integration with the [Stylelint](#) code verification tool. The tool registers itself as a PyCharm code inspection: it checks CSS code for most common mistakes and discrepancies without running the application. When a tool is activated, it launches automatically on the edited CSS file. Discrepancies are highlighted and reported in pop-up information windows, a pop-up window appears when you hover the mouse pointer over a stripe in the Validation sidebar. You can also press `Alt+Enter` to examine errors and apply suggested quick fixes. Learn more about inspections and intention actions at [Code Inspection](#) and [Intention Actions](#).

On this page:

- [Before you start](#)
- [Installing Stylelint](#)
- [Activating and configuring Stylelint](#)

Before you start

1. Integration with PyCharm is supported through the [CSS](#) plugin. The plugin is bundled with PyCharm and activated by default. If the plugin is not activated, enable it on the [Plugins](#) page of the [Settings / Preferences Dialog](#) as described in [Enabling and Disabling Plugins](#).
2. [Stylelint](#) is run through NodeJS, therefore make sure the [NodeJS](#) framework is downloaded and installed on your computer. The framework also contains the [Node Package Manager\(npm\)](#) through which [Stylelint](#) is installed.
3. Integration with [NodeJS](#) and [NPM](#) is supported through the [NodeJs](#) plugin. The plugin is not bundled with PyCharm, but it can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).

Installing Stylelint

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Node.js and NPM under Languages & Frameworks.
2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the [global](#) and at the [project](#) level. Click [+](#).
3. In the Available Packages dialog box that opens, select the `stylelint` package and click Install Package.
Learn more about installing tools through [NPM](#) in [Installing and Removing External Software Using Node Package Manager](#).

Activating and configuring Stylelint

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS. Expand the Languages & Frameworks node, and then click Stylelint under Stylesheets. The [Stylelint](#) page opens.
2. Select the Enable check box to activate [Stylelint](#). After that the controls in the dialog box become available.
3. In the Node Interpreter field, specify the NodeJS installation home. Type the path to the NodeJS executable file manually, or click the `...` button and select the location in the dialog box, that opens.
4. In the Stylelint Package field, specify the location of the `stylelint` package installed globally or in the current project, see [Using Stylelint Code Quality Tool](#).

Using Pug (Jade) Template Engine

PyCharm supports integration with the [Pug \(Jade\)](#) template engine.

On this page:

- [Before you start](#)
- [Changes to the UI](#)
- [Using Pug\(Jade\) templates in a NodeJS application](#)

Before you start

1. Enable **Node.js** support as described in [Node.js](#).
2. Make sure the **NodeJS** and **Pug (ex-Jade)** plugins are installed and enabled. The plugins are not bundled with PyCharm, but they can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#). Once enabled, the plugins are available at the IDE level, that is, you can use them in all your PyCharm projects.

Also, if you need a file watcher, make sure that the plugin **File Watchers** is installed and enabled.

The plugin is not bundled with PyCharm, but it can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).

Changes to the UI

The **Pug (ex-Jade)** plugin introduces the following changes to the PyCharm UI:

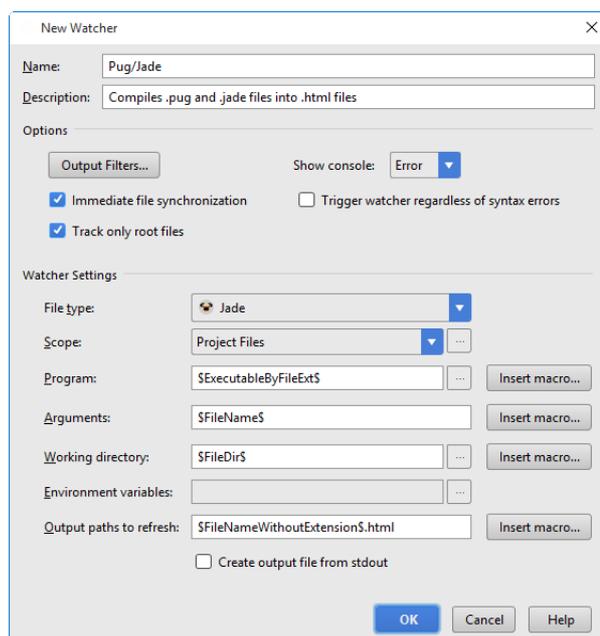
- The Jade file item is added to the New menu.
- The Pug files are marked with the icon ; the Jade files are marked with the icon .
- Coding assistance is provided in the Pug (Jade)-specific and HTML context:
 - [Code formatting](#)
 - Syntax highlighting
 - [Code completion](#)
 - [Color schemes](#)

Using Pug(Jade) templates in a NodeJS application

At runtime, the Pug (Jade) files will be transformed into HTML pages.

1. Create a project from scratch, or around existing sources, or based on a **NodeExpress** template. See [Creating Projects from Scratch in PyCharm](#), [Generating a Project from a Framework Template](#) for details.
2. Create a **Pug (Jade)** file. Follow these steps:
 1. In the [Project tool window](#), select the directory in which you want to create a new file. To do that, for example, choose File | New.
 2. On the context menu, choose Jade file and specify the file name in the dialog box that opens.
3. Create a **File Watcher** to transform files with the extension `.jade` or `.pug` into `.html` pages:
 1. Click the Add Watcher link in the upper right-hand corner of the editor.
 2. In the [New Watcher Dialog](#), accept the default predefined settings.

Note that if the executable is in the PATH, then you should not specify it explicitly. Depending on the file extension (`.jade` or `.pug`), the corresponding executable is invoked.



4. As you edit a `.pug / .jade` file, PyCharm invokes the file watcher which creates an `.html` file with the name of the processed `.pug / .jade` file and stores the generated `html` code in it.

See [Using File Watchers](#) for details.

Validating Web Content Files

On this page:

- [Types of validity checks](#)
- [Choosing the default HTML language level](#)
- [Choosing the default schema to validate XML files](#)
- [Running full validation on an XML file](#)

Types of validity checks

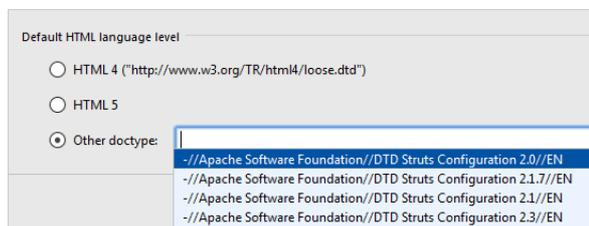
PyCharm performs two different [validity](#) checks:

- **On-the-fly validation** is available for all Web content files and is performed as you edit the file. PyCharm checks well-formedness, that is, detects various violations of syntax requirements, such as unclosed tags, wrong end-tag name, duplicate tags, unresolved links, etc. All encountered errors are highlighted in the editor.
However, this form of code validation is rather *soft*, that is, not all requirements are taken into account.
- **Full validation** involves structure validation in addition to well-formedness check. Full validation is available for files that are associated with an [XSD \(XML Schema Definition\) Schema](#) or contain a [Data Type Definition \(DTD\)](#). PyCharm checks whether the structure of your XML file complies with the structure defined in the corresponding DTD or Schema.
The results of full validation are provided as a [Message View](#).

Choosing the default HTML language level

Normally, an HTML or an XHTML file has the `<!DOCTYPE>` declaration which states the **language level** of the used in the source code from the file. This language level is used as a standard against which the contents of the file are validated. If an HTML or an XHTML file does not have a `<!DOCTYPE>` declaration, the contents of the file will be validated against the default standard (schema).

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS. Expand the Languages & Frameworks node, and then click Default XML Schemas under Schemas and DTDs.
The [Default XML Schemas](#) page opens.
2. In the Default HTML Language Level area, choose the default schema to validate HTML and XHTML files without a `<!DOCTYPE>` declaration. The available options are:
 - HTML 4 or HTML 5: Choose one of these options to have files treated as HTML 4 or HTML 5 and validated against one of these standards.
 - Other doctype: Choose this option to have HTML files by default validated against a custom DTD or schema and specify the URL of the DTD or schema to be used.
Note that code completion is available in this field: press `Ctrl+Space` to see the list of suggested URLs.



choose the [XSD \(XML Schema Definition\) Schema](#) to validate XML files. The available options are:

- XML Schema 1.1 See [W3C XML Schema Definition Language \(XSD\) 1.1 Part 1: Structures](#) and [W3C XML Schema Definition Language \(XSD\) 1.1 Part 2: Datatypes](#) for details.
- XML Schema 1.0 See [XML Schema Part 1: Structures Second Edition](#) and [XML Schema Part 2: Datatypes Second Edition](#) for details.

Choosing the default schema to validate XML files

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS. Expand the Languages & Frameworks node, and then click Default XML Schemas under Schemas and DTDs.
The [Default XML Schemas](#) page opens.
2. In the Default XML Schema Version area, choose the [XSD \(XML Schema Definition\) Schema](#) to validate XML files. The available options are:
 - XML Schema 1.1 See [W3C XML Schema Definition Language \(XSD\) 1.1 Part 1: Structures](#) and [W3C XML Schema Definition Language \(XSD\) 1.1 Part 2: Datatypes](#) for details.
 - XML Schema 1.0 See [XML Schema Part 1: Structures Second Edition](#) and [XML Schema Part 2: Datatypes Second Edition](#) for details.

Running full validation on an XML file

1. Open the desired XML file in the editor, or just select it in the [Project](#) tool window.
2. On the context menu, choose Validate.

Viewing Styles Applied to a Tag

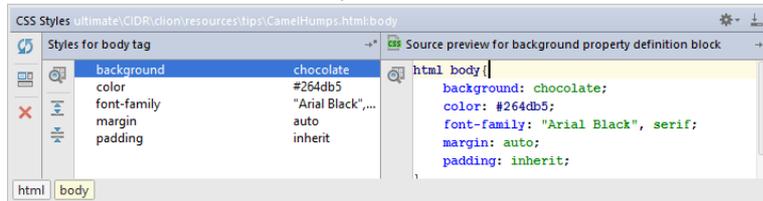
This feature is supported in the Professional edition only.

For HTML, XHTML, files, PyCharm suggests a way to explore all styles applied to an arbitrary tag.

The results for each tag are displayed in the dedicated tabs of the CSS Styles tool window. Using this tool window, you can view the list of styles applied to a tag, and the definition of these styles. Besides that, you can navigate from style to the corresponding tag in the source code.

To show styles that are used for a tag

1. Open the desired file in the editor, and right-click the tag you want to explore for applied styles.
2. On the context menu, choose Show Applied Styles for Tag.
3. View results in a dedicated tab in the CSS Styles tool window:



Viewing Images

PyCharm suggests several ways to view images, embedded in the HTML files. You can use [navigation to source](#), [open an image in an external graphical editor](#), or [preview images on-the-fly](#). The viewing modes are configurable in the [Images dialog](#).

To view an image in PyCharm

1. In the Project tool window, select the desired image file.
2. On the context menu of the file, choose Jump to Source, or press `F4`.

Tip Configure the path to the desired external editor in the External Editor section of the [Images dialog](#).

To view an image in an external editor

1. In the Project tool window, select an image file.
2. On the context menu of the file, choose Open in external editor, or press `Ctrl+Alt+F4`.

To preview an image, place the caret at a reference to an image in the editor, and do one of the following

- On the context menu, choose Jump to Source.
- Invoke the [Navigate to declaration](#) feature by pressing `Ctrl+B`.

Working with Sass and SCSS in Compass Projects

This feature is supported in the Professional edition only.

PyCharm supports development in projects structured in compliance with the [Compass framework](#). This framework uses **Sass** and **SCSS** extensions of **CSS**.

On this page:

- [Preparing for Compass development](#)
- [Setting up a Compass project](#)
- [Integrating Compass with PyCharm](#)
- [Creating a Compass Sass or a Compass SCSS compiler](#)
- [Running a Compass Sass or Compass SCSS compiler](#)

Preparing for Compass development

The **Compass** framework is installed through the **Ruby Gem manager**, therefore you need to install **Ruby** first.

1. [Download and install Ruby](#).
2. Specify a `path` variable for the folder where the Ruby executable file and the `gem.bat` file are stored. This lets you launch **Ruby** and **Gem Manager** from any folder and ensures that **Ruby** is successfully launched during compilation.
3. **Install and enable** the Sass Support plugin. The plugin is not bundled with PyCharm, but it can be installed from the **JetBrains plugin repository** as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#). The plugin is not bundled with PyCharm, but it can be installed from the **JetBrains plugin repository** as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).
4. Install **Compass**.
 1. The installation is performed in the command line mode. To start the built-in **Terminal**, hover your mouse pointer over  in the lower left corner of the IDE, then choose Terminal from the menu (see [Working with Embedded Local Terminal](#) for details).
 2. Type the following command at the command line prompt:

```
gem install compass
```

The tool is installed to the folder where Ruby executable file and the `gem.bat` file are stored.

Setting up a Compass project

You can have a project set up according to the Compass requirements in two ways: create a new Compass project or create an empty project and introduce a Compass-specific structure in it. In either case, a project is set up through command line commands. Of course, you can set up a Compass project externally and then open it in PyCharm.

During project set-up, a `config.rb` configuration file is generated. You will need to specify the location of this file when integrating Compass with PyCharm.

– To set up the Compass-specific structure in an existing project:

1. Open the desired project in PyCharm.
2. Open the built-in Terminal by hovering your mouse pointer over  in the lower left corner of PyCharm and choosing Terminal from the menu.
3. At the command line prompt, type:

```
compass init
```

– To create a Compass project from scratch:

1. Open the desired project in PyCharm.
2. Open the built-in Terminal by hovering your mouse pointer over  in the lower left corner of PyCharm and choosing Terminal from the menu.
3. Switch to the folder that will be the parent for the new project folder. Type the following command:

```
cd <parent folder name>
```

4. At the command line prompt, type:

```
compass create <the name of the project to be created>
```

Integrating Compass with PyCharm

To develop a Compass-specific project in PyCharm, you need to specify the Compass executable file `compass` and the project configuration file `config.rb`. You can do it either through the Compass Support page of the Settings dialog box or on the fly using an intention action that opens the Compass Support dialog box.

1. Open the Compass page or dialog box by doing one of the following:

- Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Compass under Languages & Frameworks.
- 1. In a `.sass` or `.scss` file, type the following **import** statement:

```
@import 'compass'
```

2. Click the red bulb icon or press `Alt+Enter`. Then choose Configure Compass from the suggestion list.

The Compass Support dialog box opens.

2. To activate Compass support, select the Enable Compass support check box.
3. In the Compass executable file text box, specify the location of the `compass` executable file under the Ruby installation. Type the path manually, for example, `C:\Ruby200-x64\bin\compass`, or choose it from the drop-down list, or click the Browse button  and choose the location of the `compass` file in the dialog box that opens.
4. In the Config path field, specify the location of the project Compass configuration file `config.rb`. Type the path manually, for example, `C:\my_projects\compass_project\config.rb`, or choose it from the drop-down list, or click the Browse button  and choose the location of the `compass` file in the dialog box that opens.
The Compass configuration file `config.rb` is generated during project set-up through `compass create` or `compass init` commands.

Creating a Compass Sass or a Compass SCSS compiler

Sass and **SCSS** are not processed by browsers that work with CSS code. Therefore to be executed, Sass or SCSS code has to be translated into CSS. This operation is referred to as **compilation** and the tools that perform it are called **compilers**.

PyCharm supports integration with a compiler tool that translates Sass and SCSS code from a Compass project without changing the Compass-specific project structure.

In PyCharm, compiler configurations are called **File Watchers**. For each supported compiler, PyCharm provides a predefined **File Watcher** template. Predefined **File Watcher** templates are available at the PyCharm level. To run a compiler against your project files, you need to create a project-specific **File Watcher** based on the relevant template, at least, specify the path to the compiler to use on your machine.

PyCharm provides a common procedure and user interface for creating **File Watchers** of all types. The only difference is in the predefined templates you choose in each case.

1. Make sure the **File Watchers** plugin is enabled. The plugin is bundled with PyCharm and activated by default. If it is not, enable the plugin. See [Enabling and Disabling Plugins](#) for details.
2. To start creating a **File Watcher**, open the Settings/Preferences dialog box by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS on the main menu, and then click File Watchers under the Tools node. The [File Watchers page](#) that opens, shows the list of **File Watchers** that are already configured in the project.
3. Click the Add button  or press `Alt+Insert` and choose the compass sass or compass scss predefined template from the pop-up list.
4. In the Program text box, specify the path to the executable file:
 - `compass.bat` for Windows
 - `compass` for Unix and macOS

Type the path manually or click the Browse button  and choose the file location in the dialog box that opens.

5. In the Arguments text box, type one of the following depending on the operating system used:
 - For macOS:
 - `compile $ProjectFileDir$` to process an entire directory
 - `compile $ProjectFileDir$ $FilePath$` to process a single file
 - For Windows:
 - `compile $UnixSeparators($ProjectFileDir$)$` to process an entire directory
 - `compile $UnixSeparators($FilePath$)$` to process a single file
 - For Linux (Ubuntu):
 - `compile $ProjectFileDir$` to process an entire directory
 - `compile $ProjectFileDir$ $FilePath$` to process a single file
6. Proceed as described on page [Using File Watchers](#).

Running a Compass Sass or Compass SCSS compiler

When you open a Sass or SCSS file, PyCharm checks whether an applicable file watcher is available in the current project. If such file watcher is configured but disabled, PyCharm displays a pop-up window that informs you about the configured file watcher and suggests to enable it.

If an applicable file watcher is configured and enabled in the current project, PyCharm starts it automatically upon the event specified in the [New Watcher dialog](#).

- If the Immediate file synchronization check box is selected, the **File Watcher** is invoked as soon as any changes are made to the source code.
- If the Immediate file synchronization check box is cleared, the **File Watcher** is started upon save (File | Save All, `Ctrl+S`) or when you move focus from PyCharm (upon frame deactivation).

PyCharm creates a separate file with the generated output. The file has the name of the source Sass or SCSS file and the extension `css`. The location of the generated files is defined in the Output paths to refresh text box of the [New Watcher dialog](#). However, in the Project Tree, they are shown under the source file which is now displayed as a node.

Live Editing of HTML, CSS, and JavaScript

This feature is supported in the Professional edition only.

On this page:

- [Introduction](#)
- [Before you start](#)
- [Configuring the Live Edit update policy in the debugging mode](#)
- [Live Editing during a debugging session](#)

Introduction

During a JavaScript debugging session, you can monitor how the changes you make in the HTML file are rendered in the actual browser window without refreshing the page manually and even without leaving PyCharm. All the changes you make are immediately reflected in the browser and the affected area is highlighted. Besides pure HTML files, this also works for other file types that contain or generate HTML, CSS, or JavaScript. You just need to specify the URL of the page in the `JavaScript Debug` run configuration. The live contents of the page being edited are shown in the `Elements` tab of the `Debug tool window`.

Currently the `Live Edit` functionality is supported only for [Google Chrome](#) and only during a debugging session.

Before you start

1. **Install** and **enable** the LiveEdit plugin. The plugin is not bundled with PyCharm, but it can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).
2. Make sure the [JetBrains Chrome Extension](#) is installed in your Chrome browser, see [Installing JetBrains Chrome extension](#).

Configuring the Live Edit update policy in the debugging mode

In the debugging mode, PyCharm provides the `Live Edit` functionality which supports both automatic and manual upload of updated files on the server side. For the client side, `Live Edit` does not provide any way to apply the changes.

In `Meteor` projects, the client-side code can be updated through the native [Meteor hot code pushes functionality](#). See [Using Meteor](#) for details.

To configure the Live Edit update policy in the debugging mode

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing `File | Settings for Windows and Linux or PyCharm | Preferences for macOS`. Expand the `Debugger` node under `Build, Execution, Deployment`, and then click `Live Edit`. The `Live Edit` page opens.
2. In the `Update` area, configure the way changes made to the code during a debugging session are applied. Note that an update will now be performed only if none of the modified files have any syntax errors.
 - `Auto in (ms)`: Choose this option to have the changes applied automatically at certain time interval and specify the delay in ms in the text box. This policy is not available for the client-side code of `Meteor` applications.
 - `Manual`: Choose this option to apply the changes manually by clicking the `Update <run configuration name> JavaScript` button  or the `Update <run configuration name>` button  on the toolbar of the `Debug tool window`. It is recommended that you choose this option for debugging `Meteor` applications because applying changes to the `client-side` code is not supported.
 - `Restart if hotswap fails`:
With changes in HTML, CSS, and JavaScript on the client side, the contents of a Web page in the browser are updated without reloading. For `NodeJS` or `Meteor` applications, PyCharm first tries to update the application incorporating the changes without restarting the NodeJS server.
 - Select this check box to have PyCharm try restarting the server if the changes cannot be applied automatically. After the restart, PyCharm brings you to the execution point where the problem took place.
If even with this option chosen automatic upload still fails, you will have to restart the server manually by clicking the `Rerun <run configuration name>` button .
 - When the check box is cleared, PyCharm just displays a pop-up window informing you about the failure and suggesting you to restart the server manually.

Live Editing during a debugging session

`Live Editing` is enabled automatically, no steps are required from your side to activate it.

To perform Live Editing during a debugging session

1. When editing the code and monitoring how the changes are rendered in the browser, it is helpful to have the areas affected by the changes highlighted. To enable highlighting affected areas, [open the PyCharm settings](#) dialog box, click `Live Edit`, and then select the `Highlight current element in browser` check box on the `Live Edit` page that opens.
2. To initiate a debugging session, [create a run configuration](#) of the type `JavaScript Debug` and click the `Debug` button  on the toolbar. PyCharm establishes connection with the `JetBrains Chrome extension` and starts a debugging session.
3. Edit the HTML file that implements the page opened in the browser and view the changes immediately rendered.

PyCharm supports an operation that is somehow opposite to live editing. During a debugging session, the `Elements` tab of the `Debug tool window` shows the HTML source code that implements the actual browser page and its [HTML DOM structure](#). Moreover, any changes made to the page through the browser are immediately reflected in the `Elements` tab. For more details, see [Viewing Actual HTML DOM](#).

Viewing Actual HTML DOM

This feature is supported in the Professional edition only.

On this page:

- [Introduction](#)
- [Before you start](#)
- [Viewing the HTML source and DOM structure](#)

Introduction

Besides **Live Editing**, that is, monitoring how the changes you make in an HTML file are rendered in the actual browser window, during a debugging session you can perform a reverse operation using the Elements tab of the Debug tool window. The tab shows the HTML source code that implements the actual browser page and its [HTML DOM structure](#). Moreover, any changes made to the page through the browser are immediately reflected in the Elements tab.

Before you start

1. Make sure you are using the **Chrome** browser for debugging.
2. Make sure the **JetBrains Chrome Extension** is installed in your Chrome browser, see [Installing JetBrains Chrome extension](#).
3. Make sure the **LiveEdit** repository plugin is installed and enabled. The plugin is not bundled with PyCharm, but it can be installed from the **JetBrains plugin repository** as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).

Viewing the HTML source and DOM structure

To view the HTML source and DOM structure of the actual page

1. To initiate a debugging session, [create a run configuration](#) of the type **JavaScript Debug** and click the Debug button  on the toolbar. PyCharm establishes connection with the JetBrains Chrome extension and starts a debugging session.
2. Switch to the Debug tool window and open the Elements tab. The tab consists of three panes, all the panes are read-only. The Text pane shows the HTML source code of the page that is currently opened in the browser. As soon as any change is made to the page in the browser (e.g. clicking an icon), the code in the pane is updated accordingly.
The Structure pane shows the DOM structure of the HTML code in the Text pane.

The Scripts pane shows a tree of executed scripts.

The Structure and Text panes are mutually synchronized. When you click a node in the DOM structure, PyCharm scrolls through the contents of the Text pane.

The panes are also synchronized with the browser. PyCharm highlights the element in the browser as soon as you click the corresponding node in the DOM structure or in the Text pane.

Using JetBrains Chrome Extension

This feature is supported in the Professional edition only.

On this page:

- [Introduction](#)
- [Installing the JetBrains Chrome extension](#)
- [Working with JetBrains Chrome extension](#)
- [Overriding the default CORS settings](#)
- [Changing port](#)

Introduction

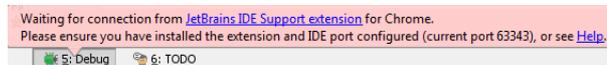
The **JetBrains Chrome extension** is mainly responsible for [debugging JavaScript in Chrome and Dartium](#). During a debugging session, the extension also supports [Live Editing](#) of HTML, CSS, and JavaScript, and shows the [DOM tree](#) and the source code of the actual page.

Installing the JetBrains Chrome extension

You can install the extension during the first debugging session with Chrome or Dartium or at any time directly from the [Chrome Web Store](#).

1. Open the [JetBrains IDE Support](#) page by doing one of the following:

- Visit the [Chrome Web Store](#) at any time.
- Start a JavaScript debugging session: create a **JavaScript** run configuration with Chrome as the debugging browser, and click the Debug button  on the toolbar. For details, see [Debugging JavaScript in Chrome](#). PyCharm informs you that it is waiting for connection with the **JetBrains IDE Support extension** and shows the following message in the Debug tool window:



Click the [JetBrains IDE Support extension](#) link which brings you to the [Chrome Web Store](#).

In either case the [JetBrains IDE Support](#) page opens.

2. Click the Add to Chrome button .
3. In the Confirm New Extension dialog box that opens, click Add. The Add to Chrome button changes to Added to Chrome .

When the extension is installed, the  icon is displayed next to the Chrome address bar.

Working with JetBrains Chrome extension

Control over the JetBrains Chrome extension is provided through the `chrome://extensions` page:

- To open the page, just type `chrome://extensions` in the Chrome address bar.

Alternatively click Customize and control Google Chrome (), choose Settings on the context menu, and then click Extensions on the `chrome://settings` page that opens.

- To deactivate the extension, clear the Enabled check box. The check box name changes to Enable.
- To activate the extension, select the Enable check box.
- To uninstall the extension, click the Remove from Chrome button .

Overriding the default CORS settings

Suppose the page you are debugging requests a resource which is protected against access for security reasons through [CORS](#) settings. You can enable access to the resources protected by default by changing the Chrome extension options.

To override the default CORS settings:

1. Right-click  and choose Options on the context menu. A web page with Chrome extension options opens showing the parameters to connect to PyCharm.
2. In the Force CORS text box, type the pattern that defines the URL addresses you want to make accessible, for example:
`http://youtrack.jetbrains.com/rest/*`.

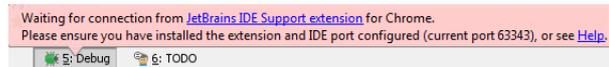
Changing port

During a debugging session, the **Chrome extension** listens to the port of the JetBrains IDE from which the extension was invoked. Each IDE including PyCharm

has its own default port on which it starts by default.

If for some reason the default PyCharm port is already busy, PyCharm finds the closest available port and starts on it. This results in a conflict: PyCharm is running on a "new" port while the **Chrome extension** still listens to the port of a previously started product and fails to establish connection with PyCharm.

The conflict reveals when you initiate a debugging session from PyCharm: the extension fails to connect through the default port, PyCharm waits for connection from the extension and displays the following message with the port number where it is actually running (for example, `current port 63343`):



To fix the problem, specify the actual PyCharm port in the Chrome extension options:

1. Right-click **JD** and choose Options on the context menu. A web page with the Chrome extension options opens showing the parameters to connect to PyCharm.
2. In the IDE Connection area, specify the actual PyCharm port in the Port spin box.

Matplotlib Support

In this section:

- [Prerequisite](#)
- [Matplotlib support](#)
- [Matplotlib in the Debug tool window console](#)

Prerequisite

To start using [Matplotlib](#), make sure that the package is properly installed in your project interpreter.

To learn how to install this package in PyCharm, refer to the section [Installing, Uninstalling and Upgrading Packages](#).

Matplotlib support

[Matplotlib](#) is the graphical package that has the interactive mode. In this mode, a graph opens in its own window, allowing you to resize, zoom in and out, etc.

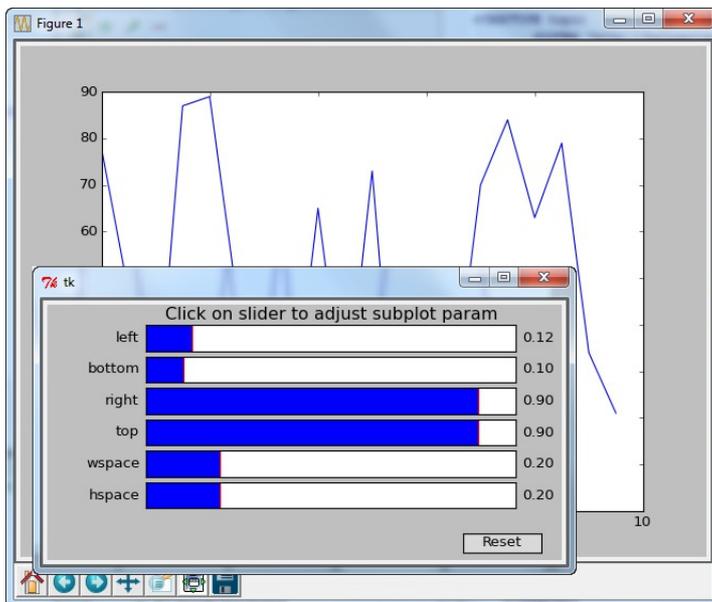
In PyCharm, the interactive mode is enabled by default. When starting an [interactive console](#), one can import Matplotlib, and build graphs as required. So doing, when Matplotlib is imported, PyCharm shows a message that interactive mode is on:

```
Python Console
C:\Python27\python.exe -u C:\Program Files (x86)\JetBrains\PyCharm 4.5\
PyDev console: starting.

>>> from matplotlib import pyplot
Backend TkAgg is interactive backend. Turning interactive mode on.
>>> pyplot.plot([1, 2, 3])
[<matplotlib.lines.Line2D object at 0x06549130>]

>>>
```

In PyCharm, on invoking the function `plot()` for each graph, this graph opens in its own pop-up window interactively:



So doing, the console is accessible for further inputs:

```
Python Console

>>> plt.title("hw")
<matplotlib.text.Text object at 0x0650C570>
>>> ax = plt.gca()

>>> ax.plot([3.1, 2.2])
<matplotlib.lines.Line2D object at 0x061E0C10>
>>> plt.draw()

>>>
```

Matplotlib in the Debug tool window console

[Matplotlib](#) is also available in the [Console](#) of the Debug tool window.

With [Matplotlib](#) imported, when stopping on a [breakpoint](#), an interactive pop-up window with the graph being debugged appears:

Node.js

This feature is supported in the Professional edition only.

In this section:

- Node.js
 - [Introduction](#)
 - [Before you start](#)
 - [Configuring Node.js in PyCharm](#)
- [Configuring Node.js Interpreters](#)
- [Running and Debugging Node.js](#)
- [Running Nodeunit Tests](#)
- [Installing and Removing External Software Using Node Package Manager](#)
- [V8 CPU and Memory Profiling](#)
- [Running NPM Scripts](#)

Introduction

PyCharm supports integration with the [Node.js](#) runtime environment thus enabling running, debugging, and unit testing of **Node.js** applications.

PyCharm recognizes Node.js code and provides basic coding assistance and highlighting for it. To get guidance in Node development, see [HowToNode.org](#).

Before you start

1. Download and install the [Node.js](#) runtime environment.
2. **Install** and **enable** the NodeJS plugin. The **NodeJS** plugin is not bundled with PyCharm, but it can be installed from the **JetBrains plugin repository** as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).

The **Node.js** plugin introduces the following changes to the PyCharm UI:

- [Node.js](#) page is added to the Settings dialog box.
- Run/debug configurations are added.

Configuring Node.js in PyCharm

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Node.js and NPM under Languages & Frameworks. The [Node.js and NPM](#) page opens.
2. In the Node Interpreter field, specify the local Node.js interpreter to use. Choose the interpreter from the drop-down list or click  and choose the interpreter in the dialog box that opens.

The term **local Node.js interpreter** denotes a Node.js installation on your computer. The term **remote Node.js interpreter** denotes a Node.js installation on a remote host or in a virtual environment set up in a **Vagrant** instance. On the [Node.js and NPM](#) page, you can specify only local interpreters. Remote interpreters are configured in the [Configure Node.js Remote Interpreter Dialog](#) dialog which can be accessed only from the [Run/Debug Configuration: Node JS](#) dialog. See [Configuring Node.js Interpreters](#) for details.

3. In the Code Assistance area, configure the Node.js core module sources if they are not configured yet.
When developing a Node.js application it can be convenient to have code completion, reference resolution, validation, and debugging capabilities for Node core modules (`fs` , `path` , `http` , etc.). However, these modules are compiled into the Node.js binary. PyCharm provides the ability to [configure these sources as a JavaScript library](#) and associate it with your project.
 - If the Node.js core module sources are not set up, PyCharm displays a notification Node.js Core Library is not enabled with an Enable button. Click this button to have PyCharm configure **Node.js Core** sources automatically.
When the configuration is completed, PyCharm displays information about the currently configured version, the notification Node.js Core Library is enabled, and adds two buttons: the Disable button and the Usage scope button.
 - If the library is set up, PyCharm displays information about the currently configured version, the notification Node.js Core Library is enabled, and adds two buttons: the Disable button and the Usage scope button.
 - Click the Disable button to discard the configuration of the **Node.js Core** libraries in the current project.
 - Click the Usage scope button to associate the desired directories with libraries.
4. If necessary, configure the scope in which the **Node.js Core** sources are treated as libraries. Click the Usage scope button, and in the [Usage Scope](#) dialog box that opens, click the desired directories, and from the drop-down list select the newly configured Node.js core module sources library.
The use of a library is enabled recursively, that is, if a library is associated with a folder it is automatically enabled in all the nested directories and files.

Configuring Node.js Interpreters

This feature is supported in the Professional edition only.

In this section:

- [Introduction](#)
- [Configuring a local Node.js interpreter](#)
- [Configuring a remote Node.js interpreter on a host accessible through SSH connection](#)
- [Configuring a remote Node.js interpreter in a Vagrant environment instance](#)
- [Configuring a remote Node.js interpreter on a remote host accessible through SFTP](#)
- [Configuring a remote Node.js interpreter in a Docker container](#)
- [Configuring mappings](#)

Introduction

With PyCharm, you can use local and remote Node.js interpreters.

The term **local Node.js interpreter** denotes a Node.js installation on your computer. The term **remote Node.js interpreter** denotes a Node.js installation on a remote host or in a virtual environment set up in a **Vagrant** instance.

You can access a remote interpreter in four ways:

- Using SSH credentials to access the host where the Node.js interpreter is installed.
- Through access to the corresponding **Vagrant** instance.
- According to a [Server Access Configuration](#). This approach is also helpful if you are going to synchronize your project sources with the Web server on the target remote host.
- Through access to a **Docker Container** with Node.js.

Configuring a local Node.js interpreter

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Node.js and NPM under Languages & Frameworks.
2. On the Node.js and NPM page that opens, click the  button next to the Node Interpreter drop-down list.
3. In the [Node.js Interpreters Dialog](#) that opens with a list of all the currently configured interpreters, click  in the toolbar. In the dialog box that opens, choose Add Local on the context menu, then choose the local installation of Node.js and click OK. You return to the [Node.js Interpreters Dialog](#) where the Node interpreter read-only field shows the path to the chosen interpreter.
4. In the Npm package field, specify the Node package manager (npm) associated with the selected interpreter. Choose the relevant npm from the drop-down list or click  next to it and in the dialog box that opens choose the location of the npm to use. Alternatively, you can specify the path to the [Yarn package manager](#) if you want to use it instead of npm.

The field is available only if the selected interpreter is of the type **local**.

When you click OK, you return to the Node.js and NPM page where the Node interpreter field shows the new interpreter.

Configuring a remote Node.js interpreter on a host accessible through SSH connection

Before you start:

1. Configure access to an **ssh** server on the target remote host and make sure this server is running.
2. Make sure the **Node.js Remote Interpreter** repository plugin is installed and enabled. The plugin is not bundled with PyCharm, but it can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).

To configure a Node.js interpreter using SSH credentials:

1. On the main menu, choose Run | Edit Configurations. In the Edit Configuration dialog box, that opens, click the Add New Configuration toolbar button , and choose Node.js on the context menu. In the [Run/Debug Configuration: Node JS](#) dialog that opens, click  next to the Node interpreter field.
2. In the [Node.js Interpreters Dialog](#) that opens with a list of all the currently configured interpreters, click  in the toolbar. In the dialog box that opens, choose Add Remote on the context menu.
3. In the [Configure Node.js Remote Interpreter Dialog](#) that opens, choose the SSH Credentials method.
4. Specify the name of the remote host and the port which the SSH server listens to. The default port number is 22.
5. Specify your credentials to access the remote host in accordance with the credentials received during the registration on the server. Type your user name and choose the authentication method:
 - To access the host through a password, choose Password from the Auth type drop-down list and type the password.
 - To access the host through a pair of SSH keys, choose Key pair, then specify the path to the file where your **private key** is stored or the passphrase if you have configured it during the generation of the key pair.
6. Specify the location of the **Node.js** executable file in accordance with the configuration of the selected remote development environment. By default PyCharm suggests the `/usr/bin/node` folder for remote hosts and Vagrant instances and `node` for Docker containers. To specify another folder, click the Browse button  and choose the relevant folder in the dialog box that opens. Note that the Node.js home directory must be open for edit.
7. When you click OK, PyCharm checks whether the Node.js executable is actually stored in the specified folder.
 - If no Node.js executable is found, PyCharm displays an error message asking you whether to continue searching or save the interpreter configuration anyway.
 - If the Node.js executable is found, you return to the Node.js Interpreters where the installation folder and the detected version of the Node.js interpreter are displayed.

Configuring a remote Node.js interpreter in a Vagrant environment instance

Before you start:

1. Make sure that [Vagrant](#) and [Oracle's VirtualBox](#) are downloaded, installed, and configured on your computer as described in [Vagrant: Working with Reproducible Development Environments](#).
2. Make sure the [Vagrant](#) and [Node.js Remote Interpreter](#) plugins are installed and enabled. The plugins are not bundled with PyCharm, but they can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#). Once enabled, the plugins are available at the IDE level, that is, you can use them in all your PyCharm projects.
3. Make sure the [Node.js Remote Interpreter](#) repository plugin is installed and enabled. The plugin is not bundled with PyCharm, but it can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).
4. Make sure that the parent folders of the following executable files are added to the system `PATH` variable:
 - `vagrant.bat` or `vagrant` from your Vagrant installation. This should be done automatically by the Vagrant installer.
 - `VBoxManage.exe` or `VBoxManage` from your Oracle's VirtualBox installation.
5. Configure the Node.js development environment in the [Vagrant instance](#) to be used. Learn more about using [Vagrant](#) with PyCharm in [Vagrant: Working with Reproducible Development Environments](#).

To configure a Node.js Interpreter in a Vagrant instance

1. On the main menu, choose Run | Edit Configurations. In the Edit Configuration dialog box, that opens, click the Add New Configuration toolbar button **+**, and choose Node.js on the context menu. In the [Run/Debug Configuration: Node JS](#) dialog that opens, click  next to the Node interpreter field.
2. In the [Node.js Interpreters Dialog](#) that opens with a list of all the currently configured interpreters, click **+** in the toolbar. In the dialog box that opens, choose Add Remote on the context menu.
3. In the [Configure Node.js Remote Interpreter Dialog](#) that opens, choose the Vagrant method.
4. Specify the Vagrant instance folder which points at the environment you are going to use. Technically, it is the folder where the [VagrantFile](#) configuration file for the desired environment is located. Based on this setting, PyCharm detects the [Vagrant host](#) and shows it as a link in the Vagrant Host URL read-only field.
5. Specify the location of the [Node.js](#) executable file in accordance with the configuration of the selected remote development environment. By default PyCharm suggests the `/usr/bin/node` folder for remote hosts and Vagrant instances and `node` for Docker containers. To specify another folder, click the Browse button  and choose the relevant folder in the dialog box that opens. Note that the Node.js home directory must be open for edit.
6. When you click OK, PyCharm checks whether the Node.js executable is actually stored in the specified folder.
 - If no Node.js executable is found, PyCharm displays an error message asking you whether to continue searching or save the interpreter configuration anyway.
 - If the Node.js executable is found, you return to the Node.js Interpreters where the installation folder and the detected version of the Node.js interpreter are displayed.

Configuring a remote Node.js interpreter on a remote host accessible through SFTP

Before you start:

1. Make sure a `ssh` server is running on the target remote host and you have configured access to it.
2. Make sure the [Remote Hosts Access](#) plugin is enabled. The plugin is bundled with PyCharm and activated by default. If the plugin is not activated, enable it on the [Plugins](#) page of the [Settings / Preferences Dialog](#) as described in [Enabling and Disabling Plugins](#).
4. Make sure you have at least one PyCharm-wide [server access configuration](#) of the type SFTP to establish access to the target host. To make a configuration available in all PyCharm projects, clear the Visible only for this project check box in the [Connection Tab](#). See [Creating a Remote Server Configuration](#) for details.

To configure a remote Node.js interpreter based on an SFTP server access configuration

1. On the main menu, choose Run | Edit Configurations. In the Edit Configuration dialog box, that opens, click the Add New Configuration toolbar button **+**, and choose Node.js on the context menu. In the [Run/Debug Configuration: Node JS](#) dialog that opens, click  next to the Node interpreter field.
2. In the [Node.js Interpreters Dialog](#) that opens with a list of all the currently configured interpreters, click **+** in the toolbar. In the dialog box that opens, choose Add Remote on the context menu.
3. In the [Configure Node.js Remote Interpreter Dialog](#) that opens, choose the Deployment Configuration method.
4. From the Deployment Configuration drop-down list, choose the [server access configuration](#) of the type SFTP according to which you want PyCharm to connect to the target host. If the settings specified in the chosen configuration ensure successful connection, PyCharm displays the URL address of the target host as a link in the Deployment Host URL field.

To use an interpreter configuration, you need [path mappings](#) that set correspondence between the project folders, the folders on the server to copy project files to, and the URL addresses to access the copied data on the server. By default, PyCharm retrieves path mappings from the chosen server access (deployment) configuration. If the configuration does not contain path mappings, PyCharm displays the corresponding error message.

To fix the problem, open the [Deployment](#) page under the Build, Execution, Deployment node, select the relevant server access configuration, switch to the Mappings tab, and map the local folders to the folders on the server as described in [Creating a Remote Server Configuration](#), section Mapping Local Folders to Folders on the Server and the URL Addresses to Access Them.

5. Specify the location of the [Node.js](#) executable file in accordance with the configuration of the selected remote development environment. By default PyCharm suggests the `/usr/bin/node` folder for remote hosts and Vagrant instances and `node` for Docker containers. To specify another folder, click the Browse button  and choose the relevant folder in the dialog box that opens. Note that the Node.js home directory must be open for edit.
6. When you click OK, PyCharm checks whether the Node.js executable is actually stored in the specified folder.
 - If no Node.js executable is found, PyCharm displays an error message asking you whether to continue searching or save the interpreter configuration anyway.
 - If the Node.js executable is found, you return to the Node.js Interpreters where the installation folder and the detected version of the Node.js interpreter are displayed.

Configuring a remote Node.js interpreter in a Docker container

You can quickly bootstrap your Node.js application with Docker, PyCharm will take care of the initial configuration by automatically creating a new `Dockerfile`, keeping your source code up-to-date and installing `npm` dependencies in the container. Configuring a Node.js environment running in a Docker container as a Node.js remote interpreter lets you run, debug, and profile your Node.js application from PyCharm.

Before you start:

1. Make sure the **Node.js**, **Node.js Remote Interpreter**, and **Docker Integration** plugins are enabled. The plugins are bundled with PyCharm and activated by default. If the plugins are not activated, enable them on the **Plugins** page of the **Settings / Preferences Dialog** as described in [Enabling and Disabling Plugins](#).
2. Download, install, and configure **Docker** as described in [Docker](#).

To configure a remote Node.js interpreter in a Docker container:

1. On the main menu, choose **Run | Edit Configurations**. In the **Edit Configuration dialog box**, that opens, click the **Add New Configuration toolbar button** , and choose **Node.js** on the context menu. In the **Run/Debug Configuration: Node JS** dialog that opens, click  next to the Node interpreter field.
2. In the **Node.js Interpreters Dialog** that opens with a list of all the currently configured interpreters, click  in the toolbar. In the dialog box that opens, choose **Add Remote** on the context menu.
3. In the **Configure Node.js Remote Interpreter Dialog** that opens, choose the **Docker** method.
4. In the **Server** field, specify the **Docker configuration** to use, see [Working with Docker: Process overview](#). Choose a configuration from the drop-down list or click  next to it and create a new configuration in the **Docker** dialog box that opens.
5. In the **Image name** field, specify the base Docker image to use. Choose one of the previously downloaded or your custom images from the drop-down list or type the image name manually, for example, `node:argon` or `mhart/alpine-node`. When you later launch the run configuration, Docker will search for the specified image on your machine. If the search fails, the image will be downloaded from the image repository specified on the [Docker Registry](#) page.
6. The **Node.js interpreter path** field shows the location of the default Node.js interpreter from the specified image.
7. When you click **OK**, PyCharm closes the **Configure Node.js Remote Interpreter Dialog** and brings you to the **Node.js Interpreters Dialog** where the new interpreter configuration is added to the list. Click **OK** to return to the run configuration.

Configuring mappings

When you debug an application with a remote Node.js interpreter, the debugger tells PyCharm the name of the currently processed file and the number of the line to be processed. PyCharm opens the local copy of this file and indicates the line with the provided number. This behaviour is enabled by specifying correspondence between files and folders on the server and their local copies. This correspondence is called mapping, it is set in the debug configuration.

If you use an interpreter accessible through SFTP connection or located on a Vagrant instance, the mappings are automatically retrieved from the corresponding deployment configuration or `Vagrantfile`. To specify additional mappings:

1. On the main menu, choose **Run | Edit Configurations**. In the **Edit Configuration dialog box**, that opens, click the **Add New Configuration toolbar button** , and choose **Node.js** on the context menu.
2. In the **Run/Debug Configuration: Node JS** dialog that opens, choose the required remote interpreter from the Node interpreter drop-down list.
3. Click  next to the **Path Mappings** field.
4. The **Edit Project Path Mappings Dialog** that opens, shows the path mappings retrieved from the deployment configuration or `Vagrantfile`. These mappings are read-only.
 - To add a custom mapping, click  and specify the path in the project and the corresponding path on the remote runtime environment in the **Local Path** and **Remote Path** fields respectively. Type the paths manually or click  and select the relevant files or folders in the dialog box that opens.
 - To remove a custom mapping, select it in the list and click .

Running and Debugging Node.js

This feature is supported in the Professional edition only.

In this section:

- [Before you start](#)
- [Local and remote modes of running or debugging Node.js applications](#)
- [Running a Node.js application](#)
- [Debugging a Node.js application locally](#)
- [Debugging a Node.js application running on a Docker container](#)
- [Debugging a running Node.js application](#)
- [Creating a Node.js run/debug configuration](#)
- [Enabling Live Editing during a Node.js debugging session](#)
- [Creating a Node.js remote debug configuration.](#)
- [Node.js multiprocessing debugging](#)

Before you start

Install and enable the NodeJS plugin. The plugin is not bundled with PyCharm, but it can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).

Local and remote modes of running or debugging Node.js applications

Running a Node.js application in PyCharm is supported only in the **local** mode. This means that PyCharm itself starts the Node.js engine and the target application according to a [run configuration](#) and gets full control over the session.

Debugging can be performed in two modes:

- Locally, with the Node.js engine started from PyCharm.
- Remotely, when PyCharm connects to an already running Node.js application. This approach gives you the possibility to re-start a debugging session without re-starting the Node.js server.

In either case, the debugging session is initiated through a [debug configuration](#)

You can also configure the behaviour of the browser and enable debugging the client-side code of the application. This functionality is provided through a `JavaScript Debug` run configuration, so technically, PyCharm creates separate run configurations for the server-side and the client-side code, but you specify all your settings in one dedicated **NodeJS** run configuration.

Running a Node.js application

1. [Create a Node.js run configuration.](#)
2. To launch the application, select the run configuration from the list on the main tool bar and then choose Run | Run <configuration name> on the main menu or click the Run toolbar button . The Run tool window opens.
3. Open the browser of your choice and open the page with the URL address generated through the [server.listen](#) function based on the `port` and `host` parameters. The page shows the result of executing your Node.js application.

Debugging a Node.js application locally

1. Set the [breakpoints](#) in the Node.js code, where necessary. At least one breakpoint is necessary otherwise the program will be just executed. If you want the debugging tool to stop at the first line of your code, set a breakpoint at the first line.
2. [Create a Node.js run/debug configuration.](#)
3. To start a debugging session, select the required debug configuration from the list on the main tool bar and then choose Run | Debug <configuration name> on the main menu or click the Debug toolbar button .
4. Open the browser of your choice and open the starting page of your application. Control over the debugging session returns to PyCharm.
5. Switch to PyCharm, where the controls of the Debug tool window are now enabled. Proceed with the debugging session [step through the breakpoints](#), switch between frames, change values on-the-fly, [examine a suspended program](#), [evaluate expressions](#), and [set watches](#).

Debugging a Node.js application running on a Docker container

1. Set the [breakpoints](#) in the Node.js code, where necessary. At least one breakpoint is necessary otherwise the program will be just executed. If you want the debugging tool to stop at the first line of your code, set a breakpoint at the first line.
2. On the main menu, choose Run | Edit Configurations. In the Edit Configuration dialog box, that opens, click the Add New Configuration toolbar button , and choose Node.js on the context menu. In the [Run/Debug Configuration: Node JS](#) dialog that opens, choose the relevant Node.js interpreter running in a Docker container. Choose one of the existing interpreters from the Node Interpreter drop-down list or configure a new one as described in [Configuring Node.js Interpreters](#).
3. Specify the Docker container settings. Type the settings manually, or click  next to the field and specify the settings in the Edit Docker Container Settings dialog that opens, or select the Auto configure check box to have PyCharm do it automatically.

To invoke the automatic configuration mode for a Docker container, select the Auto configure check box in the [Run/Debug Configuration: Node JS](#) dialog. In the **Automatic Configuration** mode:

- PyCharm creates a new image and installs the `npm` modules on it.

To avoid reinstalling the modules on every start of the container and messing up with the local and system-specific dependencies, PyCharm copies the `package.json` configuration file to the `/tmp/project_modules` folder on the image, runs the `npm install` command, and then copies the modules to the project folder in the container. Changing `package.json` in the project results in re-building the image.

- PyCharm runs the Docker container with the created image and binds your project folder to `/opt/project` folder in the Docker container to ensure synchronization on update. In addition to that, `/opt/project/node_modules` will be mapped to the OS temporary directory.

With automatic configuration, you still need to bind the port that your application is running on with the port of the container. Those exposed ports are available on the Docker host's IP address (by default 192.168.99.100). Such binding is required when you debug the client side of a **Node.js Express** application so you need to open the browser from your computer and access the application at the host of the container through the port specified in the application.

1. Click  next to the Docker Container Settings field.
 2. In the Edit Docker Container Settings dialog that opens, expand the Port bindings area.
 3. Click . The Port bindings dialog box opens. In this dialog box, map the ports as follows:
 - In the Container port text box, type the port specified in your application.
 - In the Host port text box, type the port through which you want to open the application in the browser from your computer.
 - In the Host IP text box, type the IP address of the Docker's host, the default IP is 192.168.99.100. The host is specified in the API URL field on the [Docker](#) page of the .
 - Click OK to return to the Edit Docker Container Settings dialog where the new port mapping is added to the list.
 4. Click OK to return to the [Run/Debug Configuration: Node JS](#) dialog.
4. Save the run configuration.
5. Proceed as as during a local debugging session, as described above.

Debugging a running Node.js application

With PyCharm, you can connect to an already running Node.js applications. The application can be started either on the same machine or on a **physically remote host**.

When the application to debug is running on a **physically remote host**, you need to run a proxy or any other software that ensures port forwarding on the Node.js server. This is necessary because the debug port can open only on the `localhost` network interface. The `localhost` network interface cannot be accessed from another machine therefore PyCharm cannot connect to it upon initiating a debugging session.

1. Make sure the application to debug has been launched in the target environment with the following parameters: `--debug-brk=<port through which the debugger on the remote host interacts with the network interface which accepts external connections>`
2. [Create a Node.js Remote Debug](#) configuration: in the Debug port text box, type the port number through which you will interact with the remote host according to the [server access configuration](#), see [Creating a Remote Server Configuration](#).
3. With the application still running, launch the Node.js Remote Debug configuration (select the configuration in the list and click the Debug toolbar button .
4. In the Run tool window, copy the URL address of the server and open the corresponding page in the browser. Control over the debugging session returns to PyCharm.
5. Switch to PyCharm. In the Debug tool window, [step through the breakpoints](#), switch between frames, change values on-the-fly, [examine a suspended program](#), [evaluate expressions](#), and [set watches](#).

Creating a Node.js run/debug configuration

1. Choose Run | Edit Configuration on the main menu
2. In the Edit Configuration dialog box that opens, click the Add New Configuration toolbar button , and choose Node.js on the context menu.
3. In the [Run/Debug Configuration: Node.js](#) dialog box, that opens, specify the following:
 - The name of the configuration.
 - In the Node Interpreter field, specify the Node.js installation to use. Choose the local or remote interpreter from the drop-down list, or click  and choose the interpreter in the dialog box that opens, or configure an interpreter as described in [Configuring Node.js Interpreters](#).
 - In the JavaScript File field, specify the location of the file to start running the Node.js application from.
 - If the file to run references any other files, specify their location in the Working directory field.
 - If applicable, in the Application parameters text box, specify the arguments to be passed to the application on start through the [process.argv](#) array.
4. If necessary, configure the behaviour of the browser and enable debugging the client-side code of the application. This functionality is provided through a `JavaScript Debug` run configuration, so technically, PyCharm creates separate run configurations for the server-side and the client-side code, but you specify all your settings in one dedicated **NodeJS** run configuration.
5. Click OK, when ready.

Enabling Live Editing during a Node.js debugging session

You can configure the behaviour of the browser and enable debugging the client-side code of the application. This functionality is provided through a `JavaScript Debug` run configuration, so technically, PyCharm creates separate run configurations for the server-side and the client-side code, but you specify all your settings in one dedicated **NodeJS** run configuration.

1. Choose Run | Edit Configuration on the main menu
2. From the list, choose the **Node.js** run configuration to activate the Live Edit functionality in. In the dialog box that opens, switch to the Browser / Live Edit tab.
3. Select the After launch check box to have a browser started automatically after a debugging session is launched. Specify the browser to use in the drop-down list next to the check box.
 - To use the system default browser, choose Default.
 - To use a custom browser, choose it from the list. Note that **Live Edit** is fully supported only in Chrome.
 - To configure browsers, click the Browse button  and adjust the settings in the Web Browsers dialog box that opens. For more information, see [Configuring Browsers](#).
4. Select the With JavaScript debugger check box to enable the JavaScript debugger in the selected browser and specify the URL address to open the

application at.

Creating a Node.js remote debug configuration.

1. On the main menu, choose Run | Edit Configurations.
2. In the Edit Configuration dialog box, that opens, click the Add New Configuration toolbar button **+**, and choose Node.js Remote Debug on the context menu.
3. In the **Run/Debug Configuration: Node.js Remote Debug** dialog box, that opens, specify the following:
 - The name of the configuration.
 - The host where the target application is running.
 - The port to connect to. Copy the port number from the information message in the **Run** tool window that controls the running application.
4. Click OK, when ready.

Node.js multiprocessing debugging

PyCharm supports debugging additional **Node.js** processes that are launched by the [child_process.fork\(\) method](#) or by the [cluster module](#). Such processes are shown as **threads** in the **Frame** pane on the **Debugger** tab of the **Debug Tool Window**.

1. Set the breakpoints in the processes to debug.
2. Create a **Node.js** run/debug configuration.
3. Choose the newly created configuration in the Select run/debug configuration drop-down list on the tool bar and click the Debug toolbar button .

The **Debug tool window** opens and the **Frames** drop-down list shows the additional processes as threads as soon as they are launched:



To examine the data (variables, watches, etc.) for a process, select its thread in the list and view its data in the **Variables** and **Debug Tool Window**. **Watches** panes. When you select another process, the contents of the panes are updated accordingly.

Running Nodeunit Tests

This feature is supported in the Professional edition only.

In this section:

- [Introduction](#)
- [Creating and running unit tests for Node.js applications](#)
- [Enabling unit testing for Node.js applications](#)
- [Creating Nodeunit tests](#)
- [Creating a Nodeunit run configuration](#)

Introduction

PyCharm supports integration with the `nodeunit` framework thus enabling running unit test for Node.js applications.

This topic provides guidelines in Node.js-specific unit testing procedures. For general information on testing in PyCharm, see the section [Testing](#).

Creating and running unit tests for Node.js applications

To create and run unit tests for a Node.js application, perform these general steps

1. [Enable nodeunit support](#).
2. [Write the unit tests](#).
3. To have PyCharm recognize the unit tests and detect the corresponding production source code, mark the folder where the unit tests are stored as `test` folder.
4. [Create a run configuration](#) of the type `Nodeunit`.
5. [Launch unit tests](#) and [monitor test results](#) in the `Run` tool window.

Enabling unit testing for Node.js applications

To enable unit testing for Node.js applications

1. Download, install, and enable the `Node.js` plugin. The plugin is not bundled with PyCharm, but it can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).
2. Download and install the [Node.js](#) runtime environment.
3. Download and install the [nodeunit](#) testing framework

Creating Nodeunit tests

To create Nodeunit tests

- [Create a folder](#) `test` at the same level as the `src` folder
- Populate the `test` folder. For each production file, create a separate test file.
- Mark the folder where the tests are stored as `test` folder.

Creating a Nodeunit run configuration

To create a Nodeunit run configuration

1. Open the [Run/Debug Configuration](#) dialog box by choosing `Run | Edit Configurations` on the main menu.
2. Click the Add button `+` on the toolbar and select the `Nodeunit` configuration type.
3. In the dialog box that opens, specify the following:
 1. The name to identify the configuration.
 2. The path to the NodeJS installation to use.
If you have [appointed one of the installations as default](#), the field displays the path to its executable file.
 3. The working directory. This can be the project root folder or the parent directory for the `test` folder.
 4. The scope of tests to run.
 - To have PyCharm run all the test files in a folder, choose `All JavaScript test files in the directory` from the `Run` drop-down list. In the `Directory` text box, type the path to the test folder relative to the [working directory](#).
 - To have a specific test executed, choose `JavaScript test file` from the `Run` drop-down list. In the `JavaScript test file` text box, type the path to the file relative to the [working directory](#).
4. Apply the changes and close the dialog box.

Installing and Removing External Software Using Node Package Manager

This feature is supported in the Professional edition only.

In this section:

- [Introduction](#)
- [Installing Node.js and Node Package Manager \(npm\)](#)
- [Installing an external tool globally](#)
- [Installing an external tool in a project](#)
- [Installing an external tool as a development dependency](#)

Introduction

A number of tools are started through **Node.js**, for example, the **CoffeeScript**, **TypeScript**, and **Less** compilers, **YUI**, **UglifyJS**, and **Closure** compressors, **Karma** test runner, **Grunt** task runner, etc. The **Node Package Manager (npm)** is the easiest way to install these tools, the more so that you have to install **Node.js** anyway.

Depending on the desired location of the tool executable file, choose one of the following methods:

- Install the tool **globally** at the PyCharm level so it can be used in any PyCharm project.
- Install the tool in a specific project and thus restrict its use to this project.
- Install the tool in a project as a [development dependency](#).

In either installation mode, make sure that the parent folder of the tool is added to the `PATH` variable. This enables you to launch the tool from any folder.

Installing Node.js and Node Package Manager (npm)

1. Download and install [Node.js](#). The runtime environment is required for two reasons:

- The tool is started through **Node.js**.
- **NPM**, which is a part of the runtime environment, is also the easiest way to download the tool.

If you are going to use the command line mode, make sure the path to the parent folder of the **Node.js** executable file and the path to the `npm` folder are added to the `PATH` variable. This enables you to launch the tool and **npm** from any folder.

2. Install and enable the **NodeJS** repository plugin as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).

Installing an external tool globally

Global installation makes a tool available at the PyCharm level so it can be used in any PyCharm project. Moreover, during installation the parent folder of the tool is automatically added to the `PATH` variable, which enables you to launch the tool from any folder.

– Run the installation from the command line in the **global** mode:

1. Launch the embedded Terminal (View | Tool Windows | Terminal or by hovering your mouse pointer over  in the lower left corner of PyCharm and choosing Terminal from the menu
2. Switch to the directory where **NPM** is stored or define a `PATH` variable for it so it is available from any folder, see [Installing NodeJs](#).
3. Type the following command at the command line prompt:

```
npm install -g <tool name>
```

The `-g` key makes the tool run in the **global** mode. Because the installation is performed through **NPM**, the tool is installed in the `npm` folder. Make sure this parent folder is added to the `PATH` variable. This enables you to launch the tool from any folder.

For more details on the **NPM** operation modes, see [npm documentation](#). For more information about installing the tool, see <https://npmjs.org/package/>.

– Run **NPM** from PyCharm using the Node.js and NPM page of the Settings dialog box.

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Node.js and NPM under Languages & Frameworks.
2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the **global** and at the **project** level. Click **+**.
3. In the Available Packages dialog box that opens, select the required package to install.
4. Select the Options check box and type `-g` in the text box next to it.
5. Optionally specify the product version and click Install Package to start installation.

Installing an external tool in a project

Local installation in a specific project restricts the use of a tool to this project.

– Run the installation from the command line:

1. Launch the embedded Terminal (View | Tool Windows | Terminal or by hovering your mouse pointer over  in the lower left corner of PyCharm and choosing Terminal from the menu
2. Switch to the project root folder and type the following command at the command line prompt:

```
npm install <tool name>
```

– Run **NPM** from PyCharm using the Node.js and NPM page of the Settings dialog box.

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Node.js and NPM under Languages & Frameworks.
2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the **global** and at the **project** level. Click **+**.
3. In the Available Packages dialog box that opens, select the required package.
4. Optionally specify the product version and click Install Package to start installation.

Project level installation is helpful and reliable in [template-based projects](#) of the type **Node Boilerplate** or **Node.js Express**, which already have the `node_modules` folder. The latter is important because **NPM** installs the tool in a `node_modules` folder. If your project already contains such folder, the tool is installed there.

Projects of other types or **empty** projects may not have a `node_modules` folder. In this case **npm** goes upwards in the folder tree and installs the tool in the first detected `node_modules` folder. Keep in mind that this detected `node_modules` folder may be **outside** your current project root.

Finally, if no `node_modules` folder is detected in the folder tree either, the folder is created right under the current project root and the tool is installed there.

In either case, make sure that the parent folder of the tool is added to the `PATH` variable. This enables you to launch the tool from any folder.

Installing an external tool as a development dependency

If a tool is a documentation or a test framework, which are of no need for those who are going to re-use your application, it is helpful to have it excluded from download for the future. This is done by marking the tool as a [development dependency](#), which actually means adding the tool in the `devDependencies` section of the `package.json` file.

With PyCharm, you can have a tool marked as a **development dependency** right during installation. Do one of the following:

– Run the installation from the command line:

1. Launch the embedded Terminal (View | Tool Windows | Terminal or by hovering your mouse pointer over  in the lower left corner of PyCharm and choosing Terminal from the menu
2. Switch to the project root folder and type the following command at the command line prompt:

```
npm install --dev <tool name>
```

– Run **NPM** from PyCharm using the Node.js and NPM page of the Settings dialog box.

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Node.js and NPM under Languages & Frameworks.
2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the **global** and at the **project** level. Click **+**.
3. In the Available Packages dialog box that opens, select the package.
4. Select the Options check box and type `--dev` in the text box next to it.
5. Optionally specify the product version and click Install Package to start installation.

After installation, a tool is added to the `devDependencies` section of the `package.json` file.

V8 CPU and Memory Profiling

This feature is supported in the Professional edition only.

In this section:

- [Introduction](#)
- [Why is profiling important](#)
- [Before you start](#)
- [Preparing for V8 CPU and memory heap profiling](#)
 - [Installing the v8-profiler Package](#)
- [CPU profiling](#)
 - [Configuring CPU profiling](#)
 - [Collecting CPU profiling information](#)
 - [Analyzing CPU profiling information](#)
 - [Exploring call Trees](#)
 - [Analyzing the Flame chart](#)
 - [Selecting a Fragment in the Timeline](#)
 - [Synchronization in the Flame Chart](#)
- [Memory profiling](#)
 - [Configuring memory profiling](#)
 - [Collecting memory profiling information](#)
 - [Analyzing memory profiling information](#)
 - [Navigating through a snapshot](#)

Introduction

As you may know, [V8](#) is an open-source JavaScript engine developed by Google. It is used in Google [Chrome](#), [Chromium](#), [Node.js](#), and [io.js](#).

V8 has a sampling CPU profiler [V8 profiler](#) which is intended for capturing and analyzing CPU profiles and heap snapshots for your [Node.js](#) applications. With V8 CPU profiling you can get a better understanding of which parts of your code take up the most CPU time, and how your code is executed and optimized by the V8 JavaScript engine.

You can also open and explore profiles and snapshots captured in Google Chrome DevTools for your client-side code.

Why is profiling important

- A carefully designed algorithm can make your code faster and manage your memory consumption better, even more efficiently than the virtual machine can. Profiling is the way to look inside the execution of your code and prove your assumptions about your design decisions.
- Profiling is especially relevant for Javascript, which is a powerful language with advanced features like dynamic typing, closures, and even the ability to create code at runtime. Therefore the behavior of the JavaScript engine is quite sophisticated, and there are cases when you really need to go deep into the details of how the engine works. Some of the code patterns you use need being tweaked to allow the JavaScript optimizer to do its work.
- JavaScript is by no means a simple language to judge if your code manages memory well. Closure memory leaks are a good illustration of the fact that code that looks good and simple may still cause leaks.

Before you start

1. Install the [Node.js](#) runtime environment version 0.11.0 or higher.
2. **Install** and **enable** the NodeJS plugin. The plugin is not bundled with PyCharm, but it can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#). The plugin is not bundled with PyCharm, but it can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).
3. Restart PyCharm for the changes to take effect.

Preparing for V8 CPU and memory heap profiling

V8 CPU profiling is provided through the PyCharm built-in functionality, you do not need to install any additional software.

V8 memory heap profiling is provided through the [v8-profiler](#) package.

The [v8-profiler](#) package can be installed from PyCharm using the [Node Package Manager \(npm\)](#).

Installing the v8-profiler Package

You can install the [v8-profiler](#) package using the [Node Package Manager \(npm\)](#) either in the command line mode or on the [Node.js and NPM](#) page of the Settings dialog box.

To install the [v8-profiler](#) package globally, do one of the following:

- In the command line mode, type `npm install -g v8-profiler`.
- Use the [Node.js and NPM](#) page of the Settings dialog box:
 1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Node.js and NPM under Languages & Frameworks.
 2. On the Node.js and NPM page that opens, click Install  in the Packages area.
 3. In the Available Packages dialog box that opens, select the [v8-profiler](#) package from the list, possibly using the search field.

4. Select the Options check box and type `-g` in the text box next to it to have the package installed **globally** and thus make it available from any PyCharm project.
5. Click Install.
6. When the package is installed successfully, close the dialog box. PyCharm brings you back to the Node.js and NPM page where the package is added to the list.

Learn more about installing packages through NPM at [Installing and Removing External Software Using Node Package Manager](#).

CPU profiling

With V8 CPU profiling you can get a better understanding of which parts of your code take up the most CPU time, and how your code is executed and optimized by the V8 JavaScript engine.

To identify the processes that consume most of your CPU, you can use two methods: **sampling** and **tracing**.

- When the **sampling** method is applied, you periodically record stack traces of your application. The periods between records are measured in conventional units referred to as **ticks**.
This method does not guarantee very good accuracy or precision for the following reason: snapshots are taken at random moments therefore any function can happen to be recorded in a snapshot. However, sampling can give us a rough picture of where the most of time is spent.
- When the **tracing** method is used, we actively record tracing information by ourselves, directly in the code. It is obviously better to get exact measurements of how much time each method took, and also allows you to count how many times the traced method was called. The disadvantage of this method is that it comes with bigger **result distortion** compared to **sampling**.

Result Distortion. Both sampling and tracing introduce delays into execution and therefore influence the profiling results. With sampling, delays can be estimated as some fixed amount of time for each sampling event and do not introduce greater distortion than the sampling method itself (i.e. the delay is much shorter than the sampling interval). With tracing, the profiling delay depends on the code and the places where we made tracing measurements. For instance, if a traced method is called inside other traced methods numerous, all inner delays will accumulate for the outer method. If so, it may be difficult to separate the execution time from tracing distortion.

Usually we use sampling and tracing methods together. We start with sampling to get an idea of which parts of our code take the most time, and then instrument the code with tracing calls to zero in on the issues.

Measurements are made not only for the work of your code, but also activities performed by the engine itself, such as compilation, calls of system libraries, optimization, and garbage collection. The following time metrics are made for execution of functions themselves and for performing activities:

- **Total:** the number of ticks (the time) during which a function was executed or an activity was performed.
- **Total%:** the ratio of a function/activity execution time to the entire time when measurements were made.
- **Self:** the pure execution time of a function/activity itself, without the time spent on executing functions called by it.
- **Self%:** the ratio of the pure execution time of a function/activity to the entire time when the measurements were made.
- **Of Parent:** the ratio of the pure execution time of a function to the execution time of the function that called it (**Parent**).

Configuring CPU profiling

To invoke V8 CPU profiling on application start, you need to specify additional settings in the Node.js run configuration according to which the application will be launched.

1. Choose Run | Edit Configuration on the main menu
2. From the list, choose the **Node.js** run configuration to activate CPU Profiling in or create a new configuration as described in [Running and Debugging Node.js](#).
3. Switch to the V8 Profiling pane and specify the following:
 1. Select the Record CPU profiling info check box.
 2. In the Log folder field, specify the folder to store recorded logs in. Profiling data are stored in V8 log files `isolate-<session number>`.

Collecting CPU profiling information

1. Select the run configuration from the list on the main tool bar and then choose Run | Run <configuration name> on the main menu or click the Run toolbar button .
2. When the scenario that you need to profile is executed, stop the process by clicking  on the tool bar of the Run tool window.

V8 log file will be processed by V8 scripts to calculate averaged call traces. PyCharm opens the [V8 Profiling Tool Window](#).

Analyzing CPU profiling information

Analyzing the profiling logs is available after the process stops because currently stopping and restarting profiling during execution of an application is not supported.

The collected profiling data is displayed in the [V8 Profiling Tool Window](#) which PyCharm opens automatically when you stop your application. If the window is already opened and shows the profiling data for another session, a new tab is added. Tabs that were opened automatically are named after the run configurations that control execution of the applications and collecting the profiling data.

If you want to open and analyze some previously saved profiling data, choose V8 Profiling - Analyze V8 Profiling Log on the main menu and select the relevant V8 log file `isolate-<session number>`. PyCharm creates a separate tab with the name of the log file.

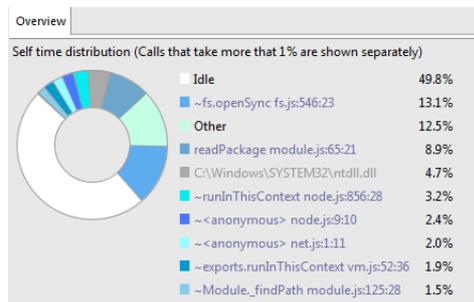
Exploring call Trees

Based on the collected profiling data, PyCharm builds three call trees and displays each of them in a separate pane. Having several call trees provides the possibility to analyze the application execution from two different points of view: on the one hand, which calls were time consuming ("heavy"), and on the other hand, "who called whom".

- The Top Calls pane shows a list of performed activities sorted in the descending order by the **Self** metrics. For each activity PyCharm displays its **Total**, **Total%**, and **Self%** metrics. For each function call, PyCharm displays the name of the file, the line, and the column where the function is defined.

Calls	Total	Total %	Self %
Unknown <total>	1153	98%	0%
JavaScript			
Function: ~<anonymous> js:1:11	730	62%	62%
Function: ~mg uuid.js:20:29	83	7%	7%
Function: ~ScriptBreakPoint.set native debug.js:266:40	59	5%	5%
Function: ~exports.runInThisContext vm.js:68:36	45	3%	3%
Function: ~runInThisContext node.js:733:28	40	3%	3%
Function: ~FrameMirror.evaluate native mirror.js:895:40	16	1%	1%
Function: ~fs.statSync fs.js:707:23	14	1%	1%
LazyCompile: ~test native regexp.js:129:20	6	0%	0%
LazyCompile: ~captureStackTrace native messages.js:808:27	6	0%	0%
Function: ~fs.lstatSync fs.js:702:24	6	0%	0%
Function: ~fs.openSync fs.js:440:23	5	0%	0%
LazyCompile: Join native array.js:68:14	4	0%	0%

The diagram in the Overview pane shows distribution of self time for calls with the **Self%** metrics above 1%.



- The Bottom-up pane also shows the performed activities sorted in the descending order by the **Self** metrics. Unlike the Top Calls pane, the Bottom-up pane shows only the activities with the **Total%** metrics above 2 and the functions that called them. This is helpful if you encounter a **heavy** function and want to find out where it was called from.

For each activity PyCharm displays its execution time in **ticks** and the **Of Parent** metrics. For each function call, PyCharm displays the name of the file, the line, and the column where the function is defined.

- The Top-down pane shows the entire call hierarchy with the functions that are execution entry points at the top. For each activity PyCharm displays its **Total**, **Total%**, **Self**, and **Self%** metrics. For each function call, PyCharm displays the name of the file, the line, and the column where the function is defined. Some of the functions may have been optimized by V8, see [Optimizing for V8](#) for details.
 - The functions that have been optimized are marked with an asterisk (*) before the function name.
 - The functions that possibly require optimization but still have not been optimized are marked with a tilde (~) character before the function name. Though optimization may be delayed by the engine or skipped if the code is short-running, a tilde (~) points at a place where the code can be rewritten to achieve better performance.

Calls	Total	Total %	Self	Self %
Function: listOnTimeout timers.js:94:23	1084	92%	0	0%
Function: ~Module.runMain module.js:488:26	1084	92%	0	0%
Function: ~Module.load module.js:268:24	1084	92%	0	0%
Function: ~Module.load module.js:339:33	1070	91%	0	0%
Function: ~Module_extensions.js module.js:4	1069	90%	0	0%
Function: ~Module_compile module.js:36	1069	90%	0	0%
Function: ~<anonymous> js:1:11	1065	90%	730	62%
Function: ~require module.js:372:1	296	25%	0	0%
Function: ~Module.require mo	296	25%	0	0%
Function: ~Module_load mo	296	25%	0	0%
Function: ~Module.load	292	24%	0	0%
Function: ~Module_	291	24%	0	0%
Function: ~Modi	291	24%	0	0%
Function: ~<	273	23%	0	0%
Function: ~<	273	23%	0	0%

- To navigate to the source code of a function, select the function in question in the tree and click on the toolbar or choose Jump to source on the context menu of the selection. The file with the source code of the selected function is opened in the editor with the cursor positioned at the function.

- When a tab for a profiling session is opened, by default the nodes with heaviest calls are expanded. While exploring the trees, you may like to fold some nodes or expand other ones. To restore the original tree presentation, click the Expand Heavy Traces button on the toolbar.

- To have PyCharm display only the calls that indeed cause performance problems, filter out light calls:
 - Click the Filter button on the toolbar.
 - Using the slider, specify the minimum **Total%** or **Parent%** value for a call to be displayed and click Done.

- To expand or collapse all the nodes in the active pane, click or on the toolbar respectively.

- To expand or collapse a node, select it and choose Expand Node or Collapse Node on the context menu of the selection.

- Save and compare calls and lines:

- To save a line with a function and its metrics, select the function and choose Copy on the context menu of the selection. This may be helpful if

you want to compare the measurements for a function from two sessions, for example, after you make some improvements to the code.

- To save only the function name and the name of the file where the function is defined, select the function and choose Copy Call on the context menu of the selection.
- To compare an item with the contents of the Clipboard, select the item in question and choose Compare With Clipboard on the context menu of the selection. Compare the items in the **Difference Viewer** that opens.
- To save the call tree in the current pane to a text file, click  on the toolbar and specify the target file in the dialog box that opens.

Analyzing the Flame chart

Use the multicolor chart in the Flame Chart tab to find where the application paused and explore the calls that provoked these pauses. The chart consists of four areas:

- The upper area shows a timeline with two sliders to limit the beginning and the end of a fragment to investigate.
- The bottom area shows a stack of calls in the form of a multicolor chart. When called for the first time, each function is assigned a random color, whereupon every call of this function within the current session is shown in this color.
- The middle area shows a summary of calls from the **Garbage Collector**, the engine, the external calls, and the execution itself. The colors reserved for the **Garbage Collector**, the engine, the external calls, and the execution are listed on top of the area:

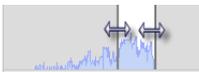


- The right-hand pane lists the calls within a selected fragment, for each call the list shows its duration, the name of the called function, and file where the function is defined.

Selecting a Fragment in the Timeline

To explore the processes within a certain period of time, you need to select the fragment in question. You can do it in two ways:

- Use the sliders:



- Click the **window** between two sliders and drag it to the required fragment:



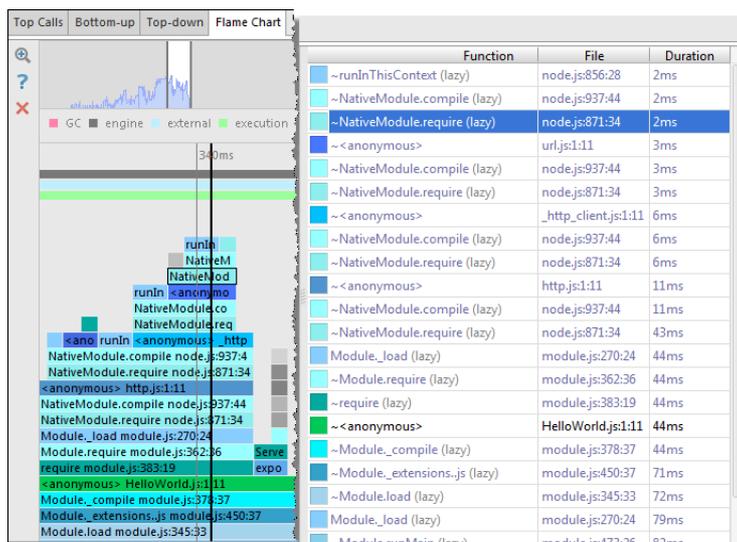
In either case, the multicolor chart below shows the stack of calls within the selected fragment.

To enlarge the chart, click the selected fragment and then click the Zoom button  on the toolbar. PyCharm opens a new tab and shows the selected fragment enlarged to fit the tab width so you can examine the fragment with more details.

Synchronization in the Flame Chart

The bottom and the right-hand areas are synchronized: as you drag the slider in the bottom area through the timeline the focus in the right-hand pane moves to the call that was performed at each moment.

Moreover, if you click a call in the bottom area, the slider moves to it automatically and the focus in the right-hand pane switches to the corresponding function, if necessary the list scrolls automatically. And vice versa, if you click an item in the list, PyCharm selects the corresponding call in the bottom area and drags the slider to it automatically:



Function	File	Duration
~runInThisContext (lazy)	node.js:856:28	2ms
~NativeModule.compile (lazy)	node.js:937:44	2ms
~NativeModule.require (lazy)	node.js:871:34	2ms
~<anonymous>	url.js:1:11	3ms
~NativeModule.compile (lazy)	node.js:937:44	3ms
~NativeModule.require (lazy)	node.js:871:34	3ms
~<anonymous>	_http_client.js:1:11	6ms
~NativeModule.compile (lazy)	node.js:937:44	6ms
~NativeModule.require (lazy)	node.js:871:34	6ms
~<anonymous>	http.js:1:11	11ms
~NativeModule.compile (lazy)	node.js:937:44	11ms
~NativeModule.require (lazy)	node.js:871:34	43ms
Module._load (lazy)	module.js:270:24	44ms
~Module.require (lazy)	module.js:362:36	44ms
~require (lazy)	module.js:383:19	44ms
~<anonymous>	HelloWorld.js:1:11	44ms
~Module._compile (lazy)	module.js:378:37	44ms
~Module._extensions.js (lazy)	module.js:450:37	71ms
~Module.load (lazy)	module.js:345:33	72ms
Module._load (lazy)	module.js:270:24	79ms
~Module.runMain (lazy)	module.js:473:26	82ms

PyCharm supports navigation from the right-hand area to the source code of called functions, to the other panes of the tool window, and to areas in the flame chart with specific metrics.

- To jump to the source code of a called function, select the call in question and choose Jump to Source on the context menu of the selection.
- To switch to another pane, select the call in question, choose Navigate To on the context menu of the selection, and then choose the destination:
 - Navigate in Top Calls
 - Navigate in Bottom-up

- Navigate in Top-down

PyCharm switches to the selected pane and moves the focus to the call in question.

- To have the flame chart zoomed at the fragments with specific metrics of a call, select the call in question, choose **Navigate To** on the context menu of the selection, and then choose the metrics:
 - Navigate to Longest Time
 - Navigate to Typical Time
 - Navigate to Longest Self Time
 - Navigate to Typical Self Time

You can also navigate to the stacktrace of a call to view and analyze exceptions. To do that, select the call in question and choose **Show As Stacktrace**.

PyCharm opens the stacktrace in a separate tab, to return to the Flame Chart pane, click **V8 CPU Profiling tool window** button in the bottom tool window.

Memory profiling

Though the **V8** JavaScript engine does memory management for you, memory leaks or dynamic memory problems are still possible. Here are some examples of memory leaks or shortcomings reasons:

- Using global objects to store collections of data, with complicated free policies.
- Errors in usages of closures: closures keep references onto outside objects.
- Keeping detached DOM nodes in javascript variables.
- Too frequent memory allocation.

Memory management control is especially important for **Node.js** applications, because the server-side code tend to run long while memory inaccuracy is accumulated.

V8 heap snapshots are complicated by their nature; they include many “engine” objects; the inner structure of objects differs from what you expect while reading your code. Learn more at [Javascript Memory Profiling](#).

Configuring memory profiling

To allow taking memory snapshots, you need to specify additional settings in the Node.js run configuration according to which the application will be launched.

1. Choose **Run | Edit Configuration** on the main menu
2. From the list, choose the **Node.js** run configuration to activate CPU Profiling in or create a new configuration as described in [Running and Debugging Node.js](#).
3. Switch to the **V8 Profiling** pane and specify the following:
 1. Select the **Allow taking heap snapshots** check box.
 2. In the **Log folder** field, specify the folder to store recorded logs in. Profiling data are stored in V8 log files `isolate-<session number>`.
 3. Specify the **v8-profiler** package to use. Choose the relevant package from the **v8-profiler** package drop-down list or click the  button next to it and choose the package in the dialog box that opens.
 4. Specify the port through which PyCharm communicates with the profiler, namely, sends a command to take a snapshot when you click the **Take Heap Snapshot** button  on the toolbar of the **Run** tool window.

To take memory snapshots of an application running on a Docker container, select the **Auto configure** check box and add **v8-profiler** to your `package.json` file, then switch to the **V8 profiling** tab and specify the path as `./node_modules/v8-profiler`.

Collecting memory profiling information

1. Select the run configuration from the list on the main tool bar and then choose **Run | Run <configuration name>** on the main menu or click the **Run** toolbar button .
2. At any time during the application execution, click the **Take Heap Snapshot** button  on the toolbar of the **Run** tool window.
3. In the dialog box that opens, choose the folder to store the taken snapshot in and specify the name to save the snapshot file with. To start analyzing the snapshot immediately, select the **Open snapshot** check box.
4. Click **OK** to save the snapshot.

Analyzing memory profiling information

The collected profiling data is displayed in the **V8 Heap Tool Window** which opens when you take a snapshot at choose to open it. If the window is already opened and shows the profiling data for another session, a new tab is added. Tabs that were opened automatically are named after the run configurations that control execution of the applications and collecting the profiling data.

If you want to open and analyze some previously saved memory profiling data, choose **V8 Profiling - Analyze V8 Heap Snapshot** on the main menu and select the relevant `.snapshot` file. PyCharm creates a separate tab with the name of the selected file.

The tool window has three tabs that present the collected information from difference point of views.

- The **Containment** tab shows the objects in your application grouped under several top-level entries: **DOMWindow objects**, **Native browser objects**, and **GC Roots**, which are roots the **Garbage Collector** actually uses. See [Containment View](#) for details.

For each object, the tab shows its **distance from the GC root**, that is the shortest simple path of nodes between the object and the GC root, the **shallow size** of the object, and the **retained size** of the object. Besides the absolute values of the object's size, PyCharm shows the percentage of memory the object

occupies.

- The Biggest Objects tab shows the most memory-consuming objects sorted by their [retained sizes](#). In this tab, you can spot memory leaks provoked by accumulating data in some global object.
- The Summary tab shows the objects in your application grouped by their types. The tab shows the number of objects of each type, their size, and the percentage of memory that they occupy. This information may be a clue to the memory state.

Each tab has a Details pane, which shows the path to the currently selected object from GC roots and the list of object's **retainers**, that is, the objects that keep links to the selected object. Every heap snapshot has many "back" references and loops, so there are always many retainers for each object.

Navigating through a snapshot

- To help differentiate objects and move from one to another without losing the context, mark objects with text labels. To set a label to an object, select the object of interest and click  on the toolbar or choose Mark on the context menu of the selection. Then type the label to mark the object with in the dialog box that opens.
- To navigate to the function or variable that corresponds to an object, select the object of interest and click  on the toolbar or choose Edit Source on the context menu of the selection. If the button and the menu option are disabled, this means that PyCharm has not found a function or a variable that corresponds to the selected object. If several functions or variables are found, they are shown in a pop-up suggestion list.
- To jump from an object in the Biggest Objects or Summary tab or Occurrences view to the same object in the Containment tab, select the object in question in the Biggest Objects or Summary tab and click  on the toolbar or choose Navigate in Main Tree on the context menu of the selection. This helps you investigate the object from the containment point of view and concentrate on the links between objects.
- To search through a snapshot:
 1. In the Containment tab, click  on the toolbar.
 2. In the [V8 Heap Search Dialog](#) that opens, specify the search pattern and the scope to search in. The available scopes are:
 - Everywhere: select this check box to search in all the scopes. When this check box is selected, all the other search types are disabled.
 - Link Names: select this check box to search among the object names that **V8** creates when calling the **C++ runtime**, see <http://stackoverflow.com/questions/11202824/what-is-in-javascript>. In the [V8 Heap Tool Window](#), link names are marked with the `%` character (`%<link name>`).
 - Class Names: select this check box to search among functions-constructors.
 - Text Strings: select this check box to perform a textual search in the contents of the objects.
 - Snapshot Object IDs: select this check box to search among the unique identifiers of objects. **V8** assigns such a unique identifier in the format to each object when the object is created and preserves it until the object is destroyed. This means that you can find and compare the same objects in several snapshots taken within the same session. In the [V8 Heap Tool Window](#), object IDs are marked with the `@` character (`@<object id>`).
 - Marks: select this check box to search among the labels you set to objects manually by clicking  on the toolbar of the Containment tab.

The search results are displayed in the Details pane, in a separate Occurrences of '<search pattern>' view. To have the search results shown grouped by the search scopes you specified, press the Group by Type toggle button on the toolbar.

When you open the dialog box next time, it will show the settings from the previous search.

Running NPM Scripts

This feature is supported in the Professional edition only.

In this section:

- [Introduction](#)
- [Before you start](#)
- [Building a tree of scripts](#)
- [Running npm scripts from the tree of scripts](#)
- [Running tasks according to a run configuration](#)
- [Running npm scripts automatically](#)
- [Running a script as a as a before-launch task](#)

Introduction

PyCharm provides interface for [running npm scripts](#). npm scripts support involves:

- Parsing `package.json` files, recognizing definitions of scripts.
- Building trees of scripts.
- Navigation between a script in the tree and its definition in the `package.json` file.
- Running and debugging scripts.
- Configuring the script execution mode and output.

Before you start

1. Download and install [Node.js](#) which contains [npm](#).
2. Install and enable the **NodeJS** plugin. The plugin is not bundled with PyCharm, but it can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).
3. Create a `package.json` file with the `scripts` property with the definitions of the scripts to run.
4. To enable debugging a script, add the `$NODE_DEBUG_OPTION` to its definition in the `package.json` file, for example:

```
{
  "name": "application-name",
  "version": "0.0.1",
  "scripts": {
    "main": "node $NODE_DEBUG_OPTION ./app-compiled.js"
  }
}
```

Scripts are launched in four ways:

- From a tree of scripts in the dedicated **NPM Tool Window**. The tool window opens when you invoke `npm` by choosing Show npm Scripts on the context menu of a `package.json` in the Project tool window or of a `package.json` opened in the editor.
- According to a dedicated run configuration, see [Run/Debug Configuration: NPM](#).
- Automatically, as a start-up task.
- As a before-launch task, from another run configuration.

The result of executing a script is displayed in the **Run tool window**. The tool window shows the npm script output, reports the errors occurred, lists the packages or plugins that have not been found, etc. The name of the last executed script is displayed on the title bar of the tool window.

Building a tree of scripts

- If the npm tool window is not opened yet, do one of the following:
 - Select the required `package.json` file in the Project tool window and choose Show npm Scripts on the context menu of the selection.
 - Open the required `package.json` file in the editor and choose Show npm Scripts on the context menu of the editor.

In either case, the npm tool window opens showing the scripts tree built according to the selected or opened `package.json` file.

- If the npm tool window is already opened, click  on the toolbar and choose the required `package.json` file from the list. PyCharm adds a new node and builds a scripts tree under it. The title of the node shows the path to the `package.json` file according to which the tree is built.
- To re-build a tree, switch to the required node and click  on the toolbar.

By default, the scripts in a tree are listed in the order in which they are defined in `package.json` (option Definition order). To have them listed in the alphabetic order, click the  toolbar button, then choose Sort by on the menu, and then choose Name.

Running npm scripts from the tree of scripts

- To run a script, do one of the following:
 - Double click the required script in the tree.
 - Select the required script and choose Run <script name> on the context menu of the selection.
 - Select the required script and press `Enter`.
- To run several scripts, use the multiselect mode: hold `Shift` (for adjacent items) or `Ctrl` (for non-adjacent items) keys and select the required scripts, then choose Run on the context menu of the selection.

Running tasks according to a run configuration

Besides using **temporary** run configurations that PyCharm creates automatically, you can create and launch your own **npm** run configurations. For details about run configurations, see [Run/Debug Configuration](#) and [Creating and Editing Run/Debug Configurations](#).

To create an **npm** run configuration:

1. Choose Run | Edit Configuration on the main menu
2. Click the Add New Configuration button **+** on the toolbar, and select npm from the pop-up list.
3. In the [Run/Debug Configuration: NPM](#) dialog box that opens, specify the name of the run configuration, the [npm command line command](#) to execute, the scripts to run (use blank spaces as separators), the location of the `package.json` file to retrieve the definitions of the scripts from, and the command line arguments to execute the script with.
Specify the location of the Node executable file and the NodeJS-specific options to be passed to this executable file, see [Node parameters](#) for details.

If applicable, specify the [environment variables](#) for the NodeJS executable file.

To run a script according to a run configuration, select the run configuration from the list on the main tool bar and then choose Run | Run <configuration name> on the main menu or click the Run toolbar button **▶**. The output is displayed in the [Run tool window](#).

Running npm scripts automatically

If you have some scripts that you run on a regular basis, you can add the corresponding run configurations to a list of **startup tasks**. The tasks will be executed automatically on the project start-up.

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Startup Tasks under Tools.
2. On the [Startup Tasks](#) page that opens, click **+** on the toolbar.
3. From the drop-down list, choose the required **npm** run configuration. The configuration is added to the list.
If no applicable configuration is available in the project, click **+** and choose Edit Configurations. Then define a configuration with the required settings in the [Run/Debug Configuration: NPM](#) page that opens. When you save the new configuration it is automatically added to the list of startup tasks.

Running a script as a before-launch task

1. Open the [Run/Debug Configurations](#) dialog by choosing Run | Edit Configurations on the main menu, and select the required configuration from the list or create it anew by clicking **+** and choosing the relevant run configuration type.
2. In the dialog box that opens, click **+** in the Before launch area and choose Run npm script from the drop-down list.
3. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with.
Specify the location of the Node.js interpreter and the parameters to pass to it.

Pyramid

This feature is supported in the Professional edition only.

PyCharm supports [Pyramid](#) development.

Note Pyramid is supported for the Python interpreters versions 2.6 and higher.

In this section:

- Pyramid
 - [Pyramid Support](#)
- [Creating Pyramid Project](#)

Pyramid Support

Pyramid support in PyCharm includes:

- Dedicated [project type](#) .
- When a new Pyramid project is created, PyCharm creates a dedicated [run/debug configuration](#) for launching a Pyramid server.
- [Chameleon](#) template language.
- [Navigation between views and templates](#) using the gutter icons.
- [Code completion](#) and resolve.

Creating Pyramid Project

This feature is supported in the Professional edition only.

Pyramid project scaffolds are intended for productive development of Pyramid applications. PyCharm takes care of creating the specific directory structure and settings.

To create a Pyramid project, follow these steps

1. Do one of the following:

- On the main menu, choose File | New | Project.
- Click New Project in the [Welcome screen](#).

Create New Project dialog box opens.

2. In the Create New Project dialog box, select Pyramid as the project type, and then specify the following:

- Project location. Note the Pyramid project will be placed within the location and the Project name will inherit the name of the parent directory. Therefore do not use leading underscores for the last segment of the location.
- Python interpreter to be used for the project.
If the desired interpreter is not found in the list, click the browse button to review the available interpreters and virtual environments, or [configure a new one](#).

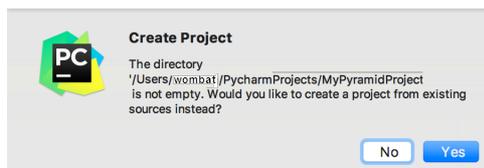
If Pyramid is missing from the selected interpreter, PyCharm informs you that Pyramid will be installed.

3. Expand the More Settings section, and do the following:

- Select the desired scaffold from the drop-down list.
- Select template language from the drop-down list.
- Specify template folder.

4. Click Create.

If an alert pops up,



click **No**. We are creating a new project.

PyCharm creates a project, installs Pyramid and its dependencies, and produces specific directory structure, which you can explore in the Project tool window. Open any file in the project directory. If there are [unsatisfied package requirements](#), PyCharm suggests to resolve or ignore them:

```
Package requirements 'pyramid_chameleon', 'pyramid_debugtoolbar', 'pyramid_tm', 'pyramid_zodbconn', 'tr... Install requirements Ignore requirements ✖
```

You should install the dependencies to be able to run the development server.

When you create a Pyramid project, you must run `setup.py develop` to install the project for development. PyCharm might inform you to do so as shown in the screenshot below:

```
Project is not installed for development Run `setup.py develop` ✖
```

Alternatively, you can select Tools | Run setup.py Task... and enter `develop`. Another pop-up dialog appears Run Setup Task develop. Click OK.

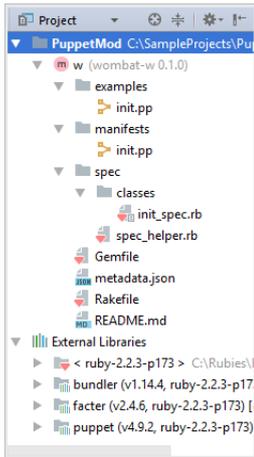
If you created an alchemy-based project, you need to initialise the database, open the terminal (make sure virtual environment is active) and run the command:

```
initialize_PROJECTNAME_db development.ini
```

(Replace `PROJECTNAME` with your exact project name.)

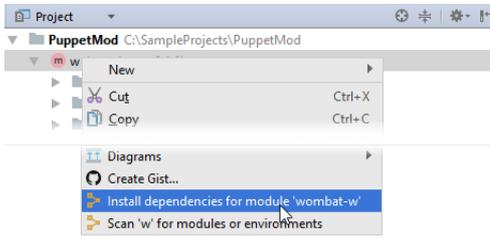
Creating a Puppet module

The project is created in the specified location. So doing, the created project features the structure of a Puppet module. Refer the [Puppet documentation](#) for details.



Installing dependencies

There is an action in the Project tool window that enables installing dependencies into a Puppet module:



Puppet modules recognize dependencies from 3 different sources:

1. If the file `.fixtures.yml` exists in a Puppet module, then the dependencies are installed into the directory `spec/fixtures/modules`, no other sources being checked.
2. If a `Puppetfile` exists in a Puppet module, then the dependencies are installed using `librarian-puppet` into `.dependencies` directory. If a `Puppetfile` exists, `librarian-puppet` ignores dependencies specified in `metadata.json`.
3. If a `metadata.json` file exists in a Puppet module, then the dependencies are installed using `librarian-puppet` into `.dependencies` directory.

Typical workflow

Here's how it works...

To work with a Puppet project, follow these general steps:

1. Open or [create a Puppet module](#).
2. If installing dependencies from the files `Puppetfile` or `metadata.json`, make sure that the gem `librarian-puppet` is installed. If the gem is not yet installed, PyCharm notifies you about the missing gem and suggests you to install it:

```
Gem 'librarian-puppet' is not available in SDK 'ruby-2.2.3-p173'  
Try to install gem 'librarian-puppet'
```

(If the dependencies are installed from `.fixtures.yml` file, this gem is not required, and no notification will be shown.)

PyCharm can find all modules/environments in a project automatically, based on dependencies files, and updates the project structure accordingly, if anything has changed. Even if PyCharm fails to update your project structure after installing additional modules into the project using the terminal, you can manually rescan the directory for modules or environments by using `Scan for modules and environments` action on the context menu.

3. Having placed the dependencies in the file `.fixtures.yml`, `Puppetfile` or `metadata.json`, right-click the `Project tool window`, and then choose `Install dependencies for module <module name>` on the context menu.

So doing, the dependencies are taken from the files `.fixtures.yml`, `Puppetfile` or `metadata.json`, located in the project root. The folder `.dependencies` (in case of creating dependencies from `Puppetfile` or `metadata.json`) or `spec/fixtures/modules/` (in case of creating dependencies from `.fixtures.yml`) is created under the project root, if it didn't exist before.



If you want to add more dependencies, invoke this command again.

Templates

This feature is supported in the Professional edition only.

PyCharm makes it possible to create and render templates written in one of the supported template languages:

- [Django](#)
- [Mako](#)
- [Jinja2](#)

It is important to note that one can edit templates without actually installing the template languages. However, in order to create or render templates, and navigate between views and templates, the corresponding template language should be properly installed.

In this section:

- [Configuring Template Languages](#)
- [Defining Template Directories](#)

Configuring Template Languages

This feature is supported in the Professional edition only.

Before making use of a certain template language, configure it in the [Python Template Languages](#) page of the Settings dialog.

To configure a template language for a project

1. [Open the Settings/Preferences dialog box](#), and click the node Python Template Languages.
2. In the [Python Template Languages](#) page, do the following:
 - From the Template language drop-down list, select the specific template language to be used in project.
 - In the Template file types area, specify the types of files, where template tags will be recognized.
Note that in HTML, XHTML, and XML files templates are always recognized.

Use Add and Remove buttons to make up the desired list of file types.

- Specify the directories where the templates in project will be stored.

To do that, open the [Project Structure](#) page, select a directory to be declared as a template directory, and click .

Note This command is duplicated on the context menu of a content root and its subfolders as Mark Directory As | Template Folder.

By default, the directory that has been defined as the template folder on [project creation](#), is marked as .

Note that if a directory is specified in the `TEMPLATES_DIR` of the `settings.py` file, then, on the first project opening, it is automatically marked as the template root.

Defining Template Directories

This feature is supported in the Professional edition only.

PyCharm suggests the following ways to mark folders as template directories:

- [Project creation](#)
- [Using the Project Structure page](#)
- [Using the Project tool window](#)

Project creation

When [creating a Django application](#), you can immediately specify a folder where templates will be stored.

Using the Project Structure page

To define template directories

1. [Open the Settings dialog](#), and click the [Project Structure](#) page.
2. Choose the directory to be marked as a template root.
3. Do one of the following:
 - Click  on the toolbar of the Content roots pane.
 - Choose Templates on the directory's context menu.

Using the Project tool window

To mark a folder as a template directory

1. In the [Project tool window](#), right-click the desired directory.
2. On the context menu, choose Mark Directory As, and then click the checked command Template Directory.
This results in adding the marked directory to the list of template directories in the [Project Structure](#) page.

TextMate

In this section:

- TextMate
 - [Prerequisites](#)
 - [TextMate support](#)
- [Importing TextMate Bundles](#)
- [Editing Files Using TextMate Bundles](#)

Prerequisites

Before you start working with TextMate, make sure that the TextMate bundle support plugin is enabled. The plugin is bundled with PyCharm and is activated by default. If the plugin is not activated, enable it on the [Plugins](#) page of the [Settings / Preferences Dialog](#) as described in [Enabling and Disabling Plugins](#).

Also, make sure that the TextMate bundles are **downloaded** to your computer.

TextMate support

TextMate files are marked with .

TextMate support in PyCharm includes:

- Possibility to [import bundles](#).
- Possibility to establish mappings between the color schemes of PyCharm and TextMate.
- [Syntax and error highlighting](#).
- Code completion in `*.markdown` files.

Importing TextMate Bundles

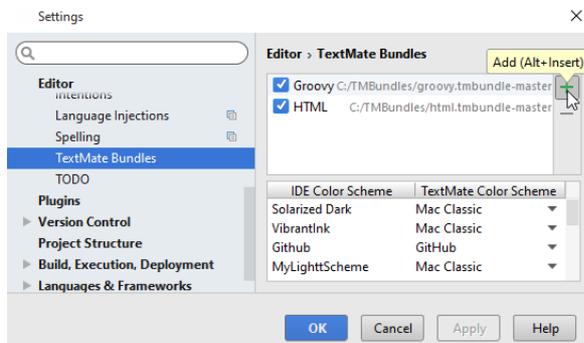
TextMate bundles become available to PyCharm when they are **downloaded** to your computer.

To import a TextMate bundle, follow these general steps

1. Make sure that TextMate bundles you want to import to PyCharm, are already downloaded. For example, you can find the desired TextMate bundles on [GitHub](#).

Note It is important to note that PyCharm cannot read the binary bundles. Use only bundle sources instead. You can find them on [GitHub](#).

2. Open **Settings/Preferences dialog**, and click **TextMate Bundles**.
3. In the **TextMate Bundles** page, click **+**:



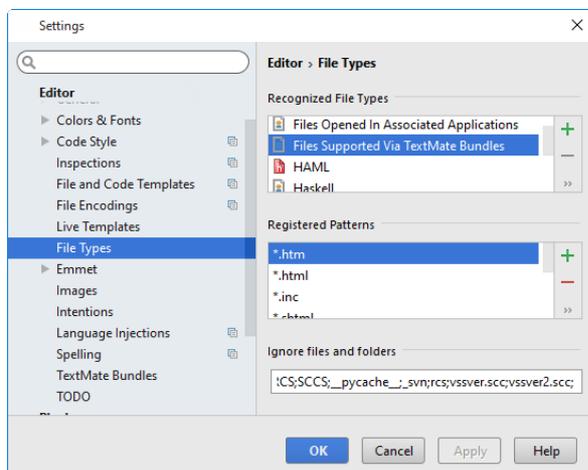
4. In the **Select Path** dialog box that opens, locate the desired bundle in the file system, and click **OK**. Repeat this step as required.
5. Apply changes.

6. If necessary, add the desired file type to the to the imported TM bundle.

For example, if you want Ruby files with `.rb` extension to be supported by the PyCharm's TextMate integration, you have to open for editing the file `Ruby.tmbundle\Syntaxes\Ruby.plist`, locate the section `fileTypes`, and under `array` add `rb`.

Then restart PyCharm.

7. Just to make sure that the desired file type will be opened via the TextMate bundles, open **File Types** page of the Settings/Preferences dialog, among the recognized file types find **Files supported via TextMate bundles**, and see the list of extensions:



Editing Files Using TextMate Bundles

1. [Create a file](#) with the extension `*.markdown`.

Note A file with the extension `*.markdown` should be associated with the type Files supported via TextMate Bundles, as described in the section [Creating and Registering File Types](#).

2. [Open](#) file for editing, and enter the desired text. The lines are highlighted according to the imported bundles.

Testing Frameworks

PyCharm enables usage of the following testing frameworks:

- [Python unittests](#)
- [py.test](#)
- [Python nosetests](#)
- [Python doctests](#)
- [Tox Support](#).

This feature is supported in the Professional edition only.

- BDD frameworks:

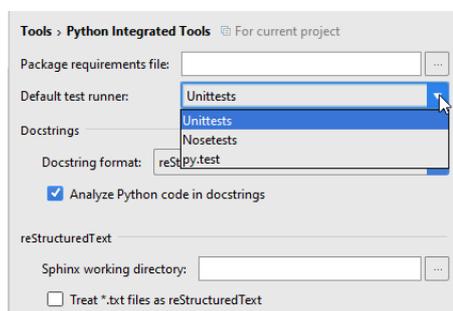
- [Behave](#).
- [Lettuce](#).
- [Cucumber](#).

Refer to the section [BDD Testing Framework](#) for details.

Prerequisites

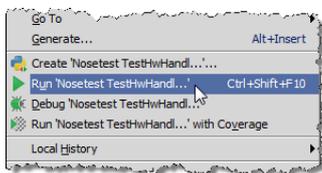
Before you start working with the testing framework of your choice, make sure that:

- The desired framework is installed on your machine. Refer to the framework documentation for the installation details.
- The default test runner is selected ([Settings/Preferences | Tools | Python Integrated Tools - Default test runner](#))



If the selected test runner is missing in the specified interpreter, the appropriate notification appears. Click the Fix button to download and install the corresponding framework.

With the test runner selected, PyCharm suggests the appropriate default run/debug configuration:



Testing frameworks support

For each of the supported testing frameworks, PyCharm provides:

- [Code completion](#), aware of the specific testing framework.
- [Run/debug configurations](#).
- Ability to [create tests](#).
- Ability to [navigate between tests and test subjects](#).
- Ability to run tests from within the IDE, and view test results in the test runner UI. The test results are shown on the [Test Runner](#) tab of the [Run tool window](#).
- Ability to run all tests or features in a directory, specific test classes, test cases or features, individual test methods or examples.
- Code inspections.
- In [docstrings](#) :
 - PyCharm recognizes Python code, provides syntax highlighting, code completion and resolve, and Python inspections.
 - Ability to recognize Python code can be turned [on or off](#).
 - If `doctests` are presented as separate files, PyCharm allows opening such files as reStructuredText files with `*.rst` extension. Such files are marked with  icon, and feature syntax highlighting.

Refer to the section [Testing](#) for the detailed description of the common testing procedures.

Tox Support

PyCharm integrates with [Tox](#) and allows running tests in multiple environments.

In this section:

- [Using Tox integration](#)

Make sure the following prerequisites are met:

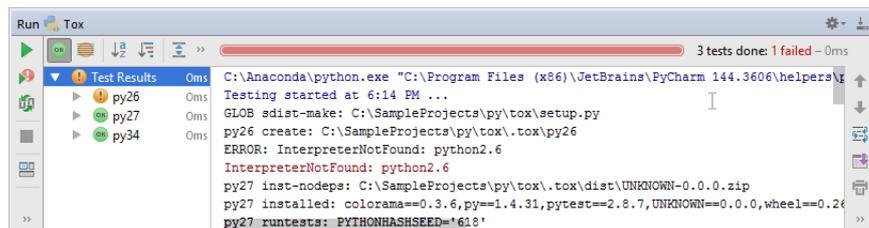
- Tox is [installed](#) on at least one of the Python interpreters available on your computer.
- Your project contains the following files:
 - `tox.ini`
 - `setup.py`
 - test files

Using Tox integration

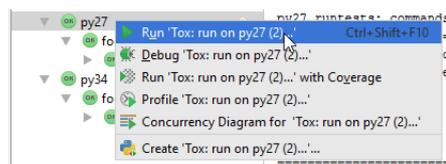
To make use of the Tox integration, follow these steps:

1. [Create a project](#) and the required [files](#).
2. Right-click the file `tox.ini` and choose Run. So doing, the dedicated [Tox run/debug configuration](#) is launched.

The results show up in the [test runner tab of the Run tool window](#):



Right-click any test result in the Test Runner to execute Tox in a particular environment:



Tip The test tree view shows only for those runners that PyCharm is aware of. If PyCharm doesn't understand the test runner, then the interpreter name only is written.

BDD Testing Framework

This feature is supported in the Professional edition only.

In this section:

- BDD Testing Framework
 - [Overview](#)
 - [Running a feature file](#)
 - [Renaming steps](#)
- [Creating .feature Files](#)
- [Creating Step Definition](#)
- [Supporting Regular Expressions in Step Definitions](#)
- [Navigating from .feature File to Step Definition](#)
- [Creating Examples Table in Scenario Outline](#)

Overview

[Behavior-driven development](#), or BDD, makes it possible to write tests in a human-readable language.

PyCharm supports [Gherkin](#) and the Cucumber-based frameworks:

- [Behave](#)
- [Lettuce](#)

Find information on testing in the section [Testing](#).

Running a feature file

PyCharm provides the ability to run a specific feature file, or all feature files in a folder, which is specified in the corresponding run/debug configurations for [Behave](#), or [Lettuce](#).

The procedure of running tests is the [same](#) as for the other testing frameworks:

1. Open the desired feature file in the editor, or select it in the Project tool window.
2. Do one of the following:
 - Right-click the selected file or folder, and choose Run <feature file name> on the context menu of the selection.
 - [Create run/debug configuration](#) for one of the BDD frameworks, and specify the desired file or folder there.

Renaming steps

When renaming Gherkin steps, mind the following limitations:

- Step definitions should not contain regular expressions
- Step names should contain alphanumeric characters only.
- A step definition should be only one in various frameworks.
- There should be a "one-to-one" mapping between a step and a step definition.

Refer to the section [Renaming](#) for details.

Creating .feature Files

This feature is supported in the Professional edition only.

Prerequisite

Behave or Lettuce should be downloaded and installed on your computer!

To install Behave or Lettuce, follow these steps:

1. [Open the project structure.](#)
2. Add the behave or lettuce package, as described in the section [Installing, Uninstalling and Upgrading Packages.](#)

Creating feature files

To create a feature file

1. In the Project tool window, right-click a directory, where feature files should be created.
2. On the context menu of the target directory, choose New | Gherkin feature file, and specify the file name.
3. In the feature file, type your scenario. Since there are no step definitions, the steps will be highlighted as unresolved.
4. [Create step definitions.](#)

Creating Step Definition

This feature is supported in the Professional edition only.

On this page:

- [Overview](#)
- [Creating step definition](#)
- [Tips and tricks](#)

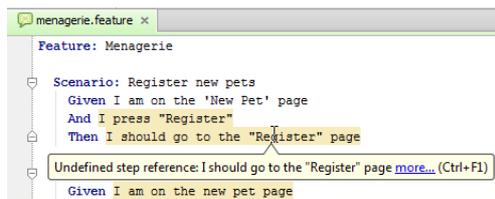
Overview

If a `.feature` file refers to a non-existent step, PyCharm's code inspection recognizes and highlights such step, and provides an intention action that helps create missing step definition.

Creating step definition

To create a missing step definition

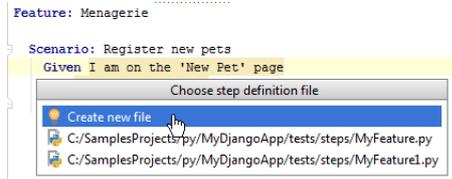
1. While editing the `.feature` file, type a reference to a step definition. PyCharm highlights step as undefined, and gives detailed information at the tooltip:



2. Press `Alt+Enter` to show the Create Step Definition intention action:



3. Select the target step definition file from the pop-up list:



You can either select one of the existing step definition files from the suggestion list, or create a new one.

PyCharm creates a step definition stub in the specified location.

4. In the selected step definition file that opens in the editor, enter the desired code.

Tips and tricks

- While typing, use code completion (`Ctrl+Space`) after keywords. Note that you can narrow down the suggestion list by typing characters contained anywhere within a description. So doing, on top of the suggestion list there will be the variants that contain specified characters as prefixes, followed by the variants with the arbitrary occurrences of characters:

```
Scenario: Scenario2
When #
Then I have [0-9]+ dollars left
Dot space and some other keys will also close
```

- You can find usages of a step definition. To do that, place the caret at the desired definition, and press `Alt+F7`. Refer to the section [Finding Usages in Project](#) for details.
- PyCharm keeps an eye on the uniqueness of the step definitions. Step definitions with the same names are highlighted.

```
def step_impl(context, amount):
    """ """
    context.amount = int(amount)

def step_impl(context, amount):
    pass
```

Supporting Regular Expressions in Step Definitions

This feature is supported in the Professional edition only.

PyCharm supports regular expressions of Python flavor. As such, it uses the Python's `re` module.

When editing source code in a step definition file, note that it is possible to specify data in a step definition as constants, or as regular expressions.

To make PyCharm perceive the entered code as a regular expression, ensure the following line is added to the source code of a step definition:

```
use_step_matcher("re")
```

So doing, the expressions used in step definitions are perceived as regular expressions:

```
use_step_matcher("re")
@when("I have (?P<number>[0-9]+) pets")
def step_impl(context, number):
    """..."""
    pass
```

If the matcher is not used, then the expressions in quotes are perceived as strings:

```
# use_step_matcher("re")
@given("I have (?P<amount>[0-9]+) dollars")
```

This feature is supported in the Professional edition only.

To navigate from a .feature file to step definition

1. Open the desired `.feature` file in the editor.
2. Do one of the following:
 - Keeping the `Ctrl` button pressed, hover your mouse pointer over a step. The step turns to a hyperlink, and its reference information is displayed at the tooltip:

```
1 Feature: Manage books
2   In order to [goal]
3     [stakeholder]
4     wants [behaviour]
5
6     Scenario: R click_button(button)
7       Given I a end
8       And I press "Create"
9
```

Click the hyperlink. The step definition file opens in the editor, with the caret resting at the desired step definition.

- On the main menu, choose `Navigate | Declaration`.
- Press `Ctrl+B`.

This feature is supported in the Professional edition only.

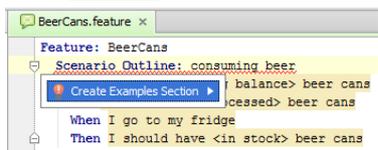
PyCharm provides support for scenario outlines, enabling you to describe multiple scenarios by means of templates with placeholders. This support includes:

- Code completion (`Ctrl+Space`) for keywords.
- Syntax highlighting for keywords, placeholders, and attributes.
- Code inspection to detect missing examples, and a quick fix for generating Examples table stub.

To create Examples table for a scenario outline, follow these general steps

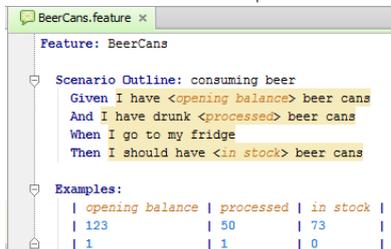
1. Having created a feature file, type the desired scenario outline. Use angle brackets to enclose placeholders. Note that initially the placeholders are not syntactically highlighted. If the steps are not defined, [create step definitions](#). Since the Examples section is missing, PyCharm marks Scenario Outline name as an error.

2. Press `Alt+Enter` to show the suggested intention action, and press `Enter`:



The header row of the Examples table is created; so doing, the placeholders are highlighted both in the table header, and in the scenario outline steps.

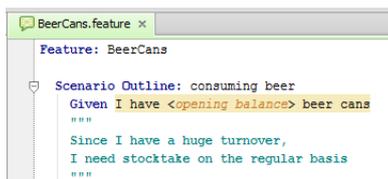
3. Add the desired rows to the Examples table:



As you add rows, the columns in the Examples section are aligned automatically.

Note highlighting of the placeholders, and the values in the Examples table, which will be substituted on running examples.

- If a colon is missing after the keyword Examples, it is recognized and marked as a syntax error, and a quick fix suggests to create colon.
- You can add textual notes to the steps of a scenario outline. Such notes should be enclosed in triple quotes; in this case, PyCharm perceives the text inside as a string, and displays it when running the scenario step:



Testing RESTful Web Services

This feature is supported in the Professional edition only.

On this page:

- [Introduction](#)
- [Composing and submitting a test request to a Web service method](#)
- [Viewing and analyzing responses from Web services](#)
- [Working with cookies](#)
- [Configuring Proxy settings](#)
- [Reusing requests](#)

Introduction

With PyCharm, you can test [RESTful](#) Web services. PyCharm provides facilities to emulate interaction with a WebService by composing and running requests, as if you were the application that actually calls the service. In addition to this, you can create, save, edit, and remove cookies, both received through responses and created manually. The **name** and **value** of a cookie is automatically included in each request to the URL address that matches the **domain** and **path** specified for the cookie, provided that the **expiry date** has not been reached. .

Testing RESTful Web Services is supported via the REST Client bundled plugin, which is by default enabled. If not, activate it in the [Plugins](#) page of the [Settings](#) dialog box.

There are two main use cases when you need to compose and run requests to a RESTful Web service:

- When you have developed and deployed a RESTful Web service and want to make sure it works as expected: that it is accessible in compliance with the specification and that it responds correctly.
- When you are developing an application that addresses a RESTful Web service. In this case it is helpful to investigate the access to the service and the required input data before you start the development. During the development, you may also call the Web service from outside your application. This may help locate errors when your application results in unexpected output while no logical errors are detected in your code and you suspect that the bottleneck is the interaction with the Web service.

Testing a RESTful Web service includes the following checks:

- That URL addresses are constituted correctly based on the service deployment end-point and the method annotations.
- That the generated server requests call the corresponding methods.
- That the methods return acceptable data.

PyCharm enables you to run these checks from the REST Client tool window by [composing and submitting requests](#) to the local server, viewing and analyzing [server responses](#).

If necessary, configure the Proxy settings on the [HTTP Proxy](#) page of the [Settings](#) dialog box.

Composing and submitting a test request to a Web service method

To compose and submit a test request to a Web service method

1. If you are going to test your own Web service, make sure it is deployed and running.
2. Choose Tools | Test RESTful Web Service. The [REST Client dedicated tool window](#) opens.
3. To have PyCharm generate an **authentication header** which will be used in [basic authentication](#), click the Generate Authorization Header button and in the dialog box that opens, specify your user name and password for accessing the target RESTful Web service through. Based on these credentials PyCharm will generate an **authentication header** which will be used in [basic authentication](#). Learn more at http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.
4. Select the test [request method](#) from the HTTP method drop-down list. The available options are:
 - [GET](#)
 - [POST](#)
 - [PUT](#)
 - [PATCH](#)
 - [DELETE](#)
 - [HEAD](#)
 - [OPTIONS](#)
5. Provide the data to calculate the URL address of the target method:
 1. In the Host/port text box, type the URL address of the host where the Web service is deployed.
 2. In the Path field, specify the relative path to the method to invoke.

You can enter the entire URL address of a method to test in the Host/port text box. Regardless of the chosen **HTTP method**, upon pressing  PyCharm will split the URL address into the **host/port** and the path to the method. The extracted relative path will be shown in the Path text box and the extracted parameters will be added to the list in the Request Parameters pane of the Request tab.

6. In the Header data pane, specify the technical data included in the [request header](#). These data are passed through header fields and define the format of the input parameters ([accept](#) field), the response format ([content-type](#) field), the caching mechanism ([cache-control](#) field), etc. To add a field to the list, click Add , then specify the field name in the Name text box and the field value in the Value drop-down list.

The set of fields and their values should comply with the Web service API. In other words, the specified input format should be exactly the one expected by the Web service as well as the expected response format should be exactly the one that the service returns.

For `accept`, `content-type`, and some other fields PyCharm provides a list of suggested values. Choose the relevant format type from the Value drop-down list.

7. Create a set of parameters to be passed to the target method and specify their values. Depending on the chosen request method, you can create a list of parameters in two ways:
 - For `GET` requests, specify the parameters to be passed as a query string inside the URL. Use the Request Parameters pane. By default, the pane shows an empty list with one line.
 - To add a parameter, click Add **+**, then specify the name of the parameter in the Name text box and the value of the parameter in the Value drop-down list.
 - To delete a parameter from the list, select it and click Remove **–**.
 - To suppress sending the specified query string parameters and disable the controls in the Request Parameters pane, press the Don't send anything toggle button .
 - To have the parameters passed to the target method inside a [request message body](#), use the Request Body pane or have them inserted in the request from a local file. The Request Body pane is disabled when the `GET`, `DELETE`, `HEAD`, or `OPTIONS` request method is selected.
 - To specify the parameters explicitly, choose the Text option and type the parameters and values in the text box.
 - To have the parameters inserted from a text file, choose the File contents option and specify the file location in the File to send field.
 - To have a binary file converted and sent in the request, choose the File upload(multipart/form-data) option and specify the file location in the File to send field.

In either case, the set of parameters and their types should comply with the Web service API, in particular, they should be exactly the same as the input parameters of the target method.

8. To submit a request to the server, click the Submit request button .
- Note that the server may lack certificate, or be untrusted.

Note If a server is not trusted, PyCharm shows a dialog box suggesting you to accept the server, or reject it. If you accept the server as trusted, PyCharm writes its certificate to the trust store. On the next connect to the server, this dialog box will not be shown.

Viewing and analyzing responses from Web services

To view and analyze responses from a Web service

- To view the response to the server request, switch to the Response tab. The tab is opened automatically when a response is received. By default, the server response is shown in the format, specified in the request header through the `content-type` field.
- To have the response converted into another format and opened in a separate tab in the editor, use the View as HTML , View as XHTML , or View as JSON  buttons.
- To view the technical data provided in the [header of a Web service response](#), switch to the Response Headers tab.

Working with cookies

Using the dedicated Cookies tab, you can create, save, edit, and remove cookies, both received through responses and created manually. The **name** and **value** of a cookie is automatically included in each request to the URL address that matches the **domain** and **path** specified for the cookie, provided that the **expiry date** has not been reached.

The tab shows a list of all currently available cookies that you received through responses or created manually. The cookies are shown in the order they were added to the list. When you click a cookie, its details become editable and are displayed in text boxes. See [REST Client Tool Window](#) for details.

- No specific steps are required to save a cookie received through a response, all the cookies received from servers are saved automatically. To edit a received cookie, click the row with the cookie in the list and update the details that are now shown in editable text boxes.
- To create a cookie manually, click **+** and specify the following:
 - The **name** and **value** of the cookie to be included in requests.
 - The **domain** and **path** the requests to which must be supplied with the **name**, **value**, and **expiry date** of the cookie.
- To remove a cookie from the list, select the row with the cookie and click **–**.

Configuring Proxy settings

To configure Proxy settings, follow these steps

1. Click the Configure HTTP Proxy icon .
2. In the [Proxy dialog](#) that opens, specify the following:
 - Enter the Proxy host name and Proxy port number in the Proxy host and Proxy port text boxes respectively.
 - To enable authorization, check the Use authorization check box and type the User name and password in the relevant fields.

Reusing requests

During a PyCharm session, PyCharm keeps track of requests and you can run any previously executed request. You can also save the settings of a request in an XML file so they are available in another PyCharm session. When necessary, you can retrieve the saved settings and run the request again.

To rerun a request within a PyCharm session

1. Click the Replay Recent Requests button .
2. From the Recent Requests pop-up list, select the relevant request. The fields are filled in with the settings of the selected request.
3. Click the Submit Request button .

To save the settings of a request so they can be retrieved in another PyCharm session

- Click the Export Request button . In the dialog box that opens, specify the name of the file to save the settings in and its parent folder.

To run a request saved during a previous PyCharm session

1. Click the Import Request button .
2. In the dialog box that opens, select the relevant XML file. The fields are filled in with the settings of the selected request.
3. Click the Submit Request button .

TypeScript Support

This feature is supported in the Professional edition only.

In this section:

- TypeScript Support
 - [Introduction](#)
 - [Preparing for TypeScript development](#)
 - [Coding assistance](#)
- [Compiling TypeScript to JavaScript](#)
- [Running TypeScript](#)
- [Debugging TypeScript](#)
- [Using TSLint Code Quality Tool](#)

Introduction

PyCharm supports developing and running [TypeScript](#) source code. PyCharm recognizes `*.ts` files, and allows you to edit them providing full range of coding assistance without any additional steps from your side. TypeScript files are marked with the  icon.

To run, debug, and test your code you will need it translated into JavaScript which requires a [compiler](#).

Preparing for TypeScript development

1. Make sure the [JavaScript Support](#) plugin is enabled. The plugin is bundled with PyCharm and activated by default. If the plugin is not activated, enable it on the [Plugins](#) page of the [Settings / Preferences Dialog](#) as described in [Enabling and Disabling Plugins](#).
2. Make sure the [Node.js](#) plugin is installed and enabled. The plugin is not bundled with PyCharm, but it can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).

Coding assistance

TypeScript support includes:

- Coding assistance:
 - [Code completion](#) for keywords, labels, variables, parameters, and functions.
 - Error and syntax highlighting.
 - Code [formatting](#) and [folding](#).
 - Numerous code [inspections](#) and [quick-fixes](#).

This assistance can be provided either by PyCharm itself, or based on the data from the [TypeScript Language Service](#). Provision of coding assistance is configured on the [TypeScript](#) page of the [Settings / Preferences Dialog](#):

- Select the Use TypeScript Service check box to get native support from the [TypeScript Language Service](#) according to the up-to-date specifications. In this case, syntax and error highlighting is performed based on the annotations retrieved from the [TypeScript Language Service](#) while code completion lists contain both suggestions from the [TypeScript Language Service](#) and suggestions calculated by PyCharm itself. To get only suggestions from PyCharm, click Configure and clear the Code completion check box in the [Service Options](#) dialog box that opens.

By default, the Use TypeScript Service check box is selected.

In the Default options field, specify the command line options to be passed to the compiler when the `tsconfig.json` file is not found. See the list of acceptable options at [TSC arguments](#). Note that, the `-w` or `--watch` option (Watch input files) is irrelevant.

- Refactoring:
 - Common refactoring procedures, such as [extract method](#), [inline](#), [rename/move](#), etc.
 - TypeScript-specific refactoring procedures, such as [change signature](#), [extract parameter](#), [extract variable](#).

See [JavaScript-Specific Refactorings](#) for details.

- [Code generation](#)
 - Generating code stubs based on [file templates](#) during file creation.
 - Ability to create [line and block comments](#) (`Ctrl+Slash` / `Ctrl+Shift+Slash`).
 - Generating `import` statements for modules, classes, and any other symbol that can be exported and called as a type, see [Creating Imports](#), section [Importing TypeScript Symbols](#).
 - Configuring automatic insertion or skipping the `public` access modifier in generated code.
- [Navigation](#) and [search](#) through the source code:
 - [Navigating with Structure View](#).
 - [Navigate | Declaration](#) (`Ctrl+B`).
 - [Navigate | Implementation](#) (`Ctrl+Alt+B`) from overridden method / subclassed class.
- [Compiling to JavaScript](#) for further running, debugging, and testing, see [Running TypeScript](#) and [Debugging TypeScript](#).

Compiling TypeScript to JavaScript

This feature is supported in the Professional edition only.

On this page:

- [Introduction](#)
- [Configuring and activating the built-in TypeScript compiler](#)
- [Compiling TypeScript using the built-in compiler](#)
- [Running the built-in TypeScript compiler as a before-launch task](#)

Introduction

TypeScript code is not processed by browsers that work with JavaScript code. Therefore to be executed, TypeScript code has to be translated into JavaScript. This operation is referred to as **compilation** and the tools that perform it are called **compilers**.

In addition to this, **compilation** also can involve generation of [source maps](#) that set correspondence between lines in your TypeScript code and in the generated JavaScript code, otherwise your breakpoints will not be recognised and processed correctly.

In PyCharm, TypeScript code is compiled into JavaScript using the [built-in TypeScript compiler](#).

Configuring and activating the built-in TypeScript compiler

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Typescript under Languages & Frameworks.
2. On the [TypeScript](#) page that opens, select the Enable TypeScript Compiler check box to activate the compiler. When the check box is selected, syntax highlighting and code completion are provided based only on the data from the built-in Typescript compiler. After you select this check box, the fields below in the area become active and you can configure the behaviour of the compiler. By default, the check box is cleared.
3. To have the compiler "wake up" upon any change to a TypeScript file, select the Track changes check box. When this check box is cleared, the built-in compiler ignores changes to TypeScript files. To re-activate the compiler, open the [TypeScript Compiler Tool Window](#) (View | Tool Windows | TypeScript Compiler), and click the Compile All button  on the toolbar.

If you have not opened the TypeScript Compiler Tool Window yet and it is not available from the View menu, choose Help | Find Action, then find and launch the TypeScript Compile All action from the list.

4. In the other fields of the TypeScript Compiler page, specify the following:
 - In the Node interpreter field, specify the location of the **Node.js** executable file. In most cases, PyCharm detects the **Node.js** executable and fills in the field automatically.
 - In the TypeScript version area, specify the version of the compiler to use (PyCharm displays the currently chosen version):
 1. Click Edit.
 2. In the Configure TypeScript Compiler dialog box that opens, choose one of the following options:
 - Detect: if you choose this option, PyCharm searches for a `typescript` package in the current project. If a `typescript` package is found, PyCharm uses it. Otherwise the default bundled package is used. This option is chosen by default.
 - Bundled: if you choose this option, PyCharm uses it without attempting to find another `typescript` package.
 - Custom directory: choose this option to use a custom version of the compiler. In the text box, specify the location of the `typescriptServices.js`, `lib.d.ts`, and `lib.es6.d.ts` files downloaded from <https://github.com/Microsoft/TypeScript/>.
 - From the Scope drop-down list, choose the scope to apply the compiler in. The available options are:
 - Project Files: all the files within the project content roots (see [Content Root](#) and [Configuring Content Roots](#)).
 - Project Production Files: all the files within the project content roots excluding test sources.
 - Project Test Files: all the files within the project test source roots.
 - Open Files: all the files that are currently opened in the editor.

VCS Scopes: these scopes are only available if your [project is under version control](#).

- Changed Files: all changed files, that is, all files associated with all existing changelists.
- Default: all the files associated with the changelist `Default`.

Alternatively, click the Browse button and configure a **custom scope** in the **Scopes** dialog box that opens. For more details on scopes, see the pages [Scopes](#) and [Scopes dialog](#).

- Choose the Set options manually to configure compilation manually:
 - Select the Generate source maps check box to generate [source maps](#) that set correspondence between lines in your TypeScript code and in the generated JavaScript code, otherwise your breakpoints will not be recognised and processed correctly.
 - Select the Compile main file only check box to have PyCharm compile only a specific file and the files that are referenced from it and ignore all the other files in the project. This may be helpful if you have a dedicated `main.ts` file which only references other files.
 - Select the Use output path check box to have the built-in compiler store the generated JavaScript files and source map in a custom folder. Specify the path to this folder explicitly or use one of the listed available macros in the format: `${<macro_name>}` . The available macros are:
 - `FileDir`: the path to the folder where the file is stored.
 - `FileRelativeDir`: the path to the **file directory** relative to the project root.
 - `FileDirRelativeToProjectRoot`: the path to the **file directory** relative to the project root.
 - `ModuleFileDir`: the path to the project root folder.
 - `SourcePath`: the path to the **source folder** under the **content root** to which the file belongs, see [Configuring Content Roots](#) and [Configuring Folders Within a Content Root](#).

- `FileDirRelativeToSourcePath` : the path to the file relative to the **source folder** under the **content root** to the file belongs.
- Choose the Use tsconfig.json option, have the built-in compiler analyze the code according to the settings specified in the `tsconfig.json` file. When you open a project, PyCharm starts searching for a `tsconfig.json` file in it. If a `tsconfig.json` file is found, the compiler uses the options specified in it. Otherwise an error is reported.

Compiling TypeScript using the built-in compiler

1. To activate a previously configured built-in compiler, select the Enable TypeScript Compiler check box on the [TypeScript Compiler page](#). Alternatively, use the buttons on the pane which PyCharm displays when you open a TypeScript file.
 - To enable an already configured compiler, click Enable.
 - To configure a compiler, click Configure and specify the settings on the TypeScript Compiler page that opens.
 - To leave the compiler disabled and suppress showing the pane in the future, click Dismiss, whereupon the compiler can be later activated only on the TypeScript Compiler page.
2. View the errors detected during compilation, compile separate files and the entire project in the [TypeScript Compiler Tool Window](#). If you have configured the compiler to track changes, which is the default behaviour, the tool window opens automatically as soon as you start editing a TypeScript file. If tracking changes is disabled and you have closed the tool window, to re-open it choose View | Tool Windows | TypeScript Compiler.
 - Click the Compile Current File toolbar button  to run the compiler against the entire file opened in the active editor tab. This approach is helpful if you do not want the compiler to wake up on any change as you edit your code and for this purpose the Track Changes check box on the [TypeScript](#) page of the Settings dialog box is cleared.
If you click this button from the Project Errors pane, after compilation PyCharm switches to the Current Errors pane.
 - Click the Compile All toolbar button  to run the compiler against all the TypeScript files in the current project. If you click this button from the Current Errors pane, after compilation PyCharm switches to the Project Errors pane.
 - To clear the contents of the current pane, click the Clear All toolbar button .
 - To navigate to the fragment of code where an error occurred during compilation, select the error message in question and choose Jump to Source on the context menu of the selection.

The compiler stores the generated output in a separate file. The JavaScript file has the name of the source TypeScript file and the extension `.js`, the source map file has the name of the source TypeScript file and the `.js.map` extension. In the Project Tree, the files are shown under the source TypeScript file which is now displayed as a node. This presentation is preserved even if you turn off the compiler during a PyCharm session.

Running the built-in TypeScript compiler as a before-launch task

You can run the built-in compiler before launching a run configuration to make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files.

1. Check whether the compiler is activated on the [TypeScript Compiler page](#), see [Compiling TypeScript to JavaScript](#) above.
2. Open the [Run/Debug Configurations](#) dialog by choosing Run | Edit Configurations on the main menu, and select the required configuration from the list or create it anew by clicking **+** and choosing the relevant run configuration type.
3. In the dialog box that opens, click **+** in the Before launch area and choose Compile TypeScript from the drop-down list.
4. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:
 - If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.
 - If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.

Running TypeScript

This feature is supported in the Professional edition only.

On this page:

- [Introduction](#)
- [Running a file with injected TypeScript from PyCharm](#)

Introduction

TypeScript code is not processed by browsers that work with JavaScript code. Therefore to be executed, TypeScript code has to be translated into JavaScript. This operation is referred to as **compilation** and the tools that perform it are called **compilers**.

Running a file with injected TypeScript from PyCharm

Note that no [run configuration](#) is required for launching applications with injected TypeScript from PyCharm.

1. [Compile the TypeScript code into Javascript](#).
2. In the editor, open the HTML file with the TypeScript reference. This HTML file does not necessarily have to be the one that implements the starting page of the application.
3. Do one of the following:
 - Choose View | Preview file in on the main menu or press `Alt+F2`. Then select the desired browser from the pop-up menu.
 - Hover your mouse pointer over the code to show the browser icons bar:  Click the icon that indicates the desired browser.

Debugging TypeScript

This feature is supported in the Professional edition only.

Before debugging, you need to compile your code into JavaScript. You can do that using the [built-in compiler](#) or another tool of your choice.

PyCharm needs [source maps](#) to recognize breakpoints you set in the TypeScript code. To have source maps generated during compilation, open the `tsconfig.json` file and make sure the `sourceMap` property is set to `true`.

Before you start, install the [JetBrains IDE Support](#) Chrome extension. Find more about that and about additional configuration of the debugger in [Configuring JavaScript Debugger and JetBrains Chrome Extension](#).

Debugging client-side TypeScript

Most often, you may want to debug a client-side application running on an external development web server, e.g. powered by Node.js.

1. [Configure and set breakpoints](#) in the TypeScript code.
2. Compile the TypeScript code into JavaScript [using the built-in compiler](#) or as a part of your build process.
3. Run the application in the **development mode**. Often you need to run `npm start` for that. When the development server is ready, copy the URL address at which the application is running in the browser - you will need to specify this URL address in the run/debug configuration.
4. Create a debug configuration of the type **JavaScript Debug**:

Choose Run | Edit Configuration on the main menu, click the Add New Configuration button **+** on the toolbar, and select JavaScript Debug from the pop-up list.

5. In the dialog box that opens, specify the URL address at which the application is running. This URL can be copied from the address bar of your browser as described in Step 3 above. Click OK to save the configuration settings.
6. Choose the newly created configuration in the Select run/debug configuration drop-down list on the toolbar and click the Debug toolbar button **🐛**. The URL address specified in the run configuration opens in the chosen browser and the [Debug tool window](#) appears.
7. In the Debug tool window, proceed as usual: [step through the program](#), [stop and resume](#) program execution, [examine it when suspended](#), [view actual HTML DOM](#), etc.

If your TypeScript code is running on the **built-in PyCharm server**, you can also debug it same ways as when [debugging JavaScript running on the built-in server](#).

Testing TypeScript

This feature is supported in the Professional edition only.

Using TSLint Code Quality Tool

This feature is supported in the Professional edition only.

On this page:

- [Introduction](#)
- [Before you start](#)
- [Installing TSLint](#)
- [Activating and configuring the TSLint tool](#)
- [TSLint quick-fixes](#)

Introduction

PyCharm provides facilities to run TypeScript-specific code quality inspections through integration with the **TSLint** code verification tool. This tool registers itself as a PyCharm code inspection: it checks TypeScript code for most common mistakes and discrepancies without running the application. When the tool is activated, it launches automatically on the edited **TypeScript** file. Discrepancies are highlighted and reported in pop-up information windows, a pop-up window appears when you hover the mouse pointer over a stripe in the Validation sidebar. You can also press `Alt+Enter` to examine errors and apply suggested quick fixes. Learn more about inspections and intention actions at [Code Inspection](#) and [Intention Actions](#).

Before you start

1. The **TSLint** tool is run through **NodeJS**, therefore make sure the [NodeJS](#) runtime environment is downloaded and installed on your computer. The runtime environment also contains the [Node Package Manager\(npm\)](#) through which **tslint** is installed.
2. Integration with **NodeJS** and **NPM** is supported through the **NodeJS** plugin. The plugin is not bundled with PyCharm, but it can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).

Installing TSLint

Open the PyCharm built-in Terminal (View | Tool Windows | Terminal or `Alt+F12`) and type `npm install tslint typescript --save-dev`. See [TSLint](#) for more information.

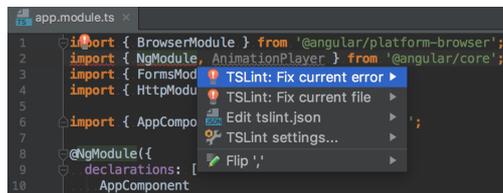
Activating and configuring the TSLint tool

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS. Expand the Languages & Frameworks node, and then click TSLint under TypeScript.
2. On the [TSLint page](#) that opens, select the Enable check box. After that all the controls in the page become available.
3. Specify the location of the **Node.js** executable file and the path to the **TSLint** package.
4. In the Configuration File area, appoint the configuration to use. By default, PyCharm first looks for a `tslint.json` configuration file. PyCharm starts the search from the folder where the file to be checked is stored, then searches in the parent folder, and so on until reaches the project root. If no `tslint.json` file is found, **TSLint** uses its default embedded configuration file. Accordingly, you have to define the configuration to apply either in a `tslint.json` configuration file, or in a custom JSON configuration file, or rely on the default embedded configuration.
 - To have PyCharm look for a `tslint.json` file, choose the Search for tslint.json option. If no `tslint.json` file is found, the default embedded configuration file will be used.
 - To use a custom file, choose the Configuration File option and specify the location fo the file in the Path field. Choose the path from the drop-down list, or type it manually, or click the `...` button and select the relevant file from the dialog box that opens.
5. If necessary, in the Additional Rules Directory field, specify the location of the files with additional code verification rules. These rules will be applied after the rules from `tslint.json` or the above specified custom configuration file and accordingly will override them.

TSLint quick-fixes

With PyCharm, you can fix some of the issues reported by **TSLint** automatically.

- To fix a specific error, place the cursor at the highlighted code, press `Alt+Enter`, and then choose TSLint: fix current error from the pop-up menu.
- To fix all the issues detected in the file, choose TSLint: fix current file.



Using Bower Package Manager

This feature is supported in the Professional edition only.

On this page:

- [Introduction](#)
- [Preparing to install Bower](#)
- [Installing Bower globally](#)
- [Installing Bower in a project](#)
- [Creating a Bower configuration file bower.json](#)
- [Configuring Bower in PyCharm](#)

Introduction

PyCharm provides interface for installing, uninstalling, and upgrading client-side libraries and frameworks for your project using the [Bower Package Manager](#). Alternatively, you can use the tool in the command line mode from the [embedded local terminal](#).

The easiest way to install the Bower package manager is to use the **Node Package Manager (npm)**, which is a part of [Node.js](#). See [Installing and Removing External Software Using Node Package Manager](#) for details.

Depending on the desired location of the Bower package manager executable file, choose one of the following methods:

- Install the package manager **globally** at the PyCharm level so it can be used in any PyCharm project.
- Install the package manager in a specific project and thus restrict its use to this project.
- Install the package manager in a project as a [development dependency](#).

In either installation mode, make sure that the parent folder of the Bower package manager is added to the `PATH` variable. This enables you to launch the package manager from any folder.

PyCharm provides user interface both for **global** and **project** installation as well as supports installation through the command line.

Preparing to install Bower

1. Download and install [Node.js](#). The runtime environment is required for two reasons:

- The Bower package manager is started through **Node.js**.
- **NPM**, which is a part of the runtime environment, is also the easiest way to download the Bower package manager.

If you are going to use the command line mode, make sure the path to the parent folder of the **Node.js** executable file and the path to the `npm` folder are added to the `PATH` variable. This enables you to launch the Bower package manager and **npm** from any folder.

2. Install and enable the **NodeJS** repository plugin as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).

Installing Bower globally

Global installation makes a package manager available at the PyCharm level so it can be used in any PyCharm project. Moreover, during installation the parent folder of the package manager is automatically added to the `PATH` variable, which enables you to launch the package manager from any folder.

– Run the installation from the command line in the **global** mode:

1. Launch the embedded Terminal (View | Tool Windows | Terminal or by hovering your mouse pointer over  in the lower left corner of PyCharm and choosing Terminal from the menu
2. Switch to the directory where **NPM** is stored or define a `PATH` variable for it so it is available from any folder, see [Installing NodeJS](#).
3. Type the following command at the command line prompt:

```
npm install -g bower
```

The `-g` key makes the package manager run in the **global** mode. Because the installation is performed through **NPM**, the Bower package manager is installed in the `npm` folder. Make sure this parent folder is added to the `PATH` variable. This enables you to launch the package manager from any folder.

For more details on the **NPM** operation modes, see [npm documentation](#). For more information about installing the Bower package manager, see <https://npmjs.org/package/bower>.

– Run **NPM** from PyCharm using the Node.js and NPM page of the Settings dialog box.

1. Open the [Settings / Preferences Dialog](#) by pressing `(Ctrl+Alt+S)` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Node.js and NPM under Languages & Frameworks.
2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the **global** and at the **project** level. Click `+`.
3. In the Available Packages dialog box that opens, select the required package to install.
4. Select the Options check box and type `-g` in the text box next to it.
5. Optionally specify the product version and click Install Package to start installation.

Installing Bower in a project

Local installation in a specific project restricts the use of a package manager to this project.

– Run the installation from the command line:

1. Launch the embedded Terminal (View | Tool Windows | Terminal or by hovering your mouse pointer over  in the lower left corner of PyCharm and

choosing Terminal from the menu

2. Switch to the project root folder and type the following command at the command line prompt:

```
npm install bower
```

– Run **NPM** from PyCharm using the Node.js and NPM page of the Settings dialog box.

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Node.js and NPM under Languages & Frameworks.
2. On the Node.js and NPM page that opens, the Packages area shows all the Node.js-dependent packages that are currently installed on your computer, both at the **global** and at the **project** level. Click **+**.
3. In the Available Packages dialog box that opens, select the required package.
4. Optionally specify the product version and click Install Package to start installation.

Creating a Bower configuration file bower.json

1. In the command line mode, switch to your project directory.
2. Type the following command at the command line prompt:

```
bower init
```

If **Bower** does not start, check the installation: the parent folder or the **Bower** executable file should be specified in the **PATH** variable.

3. Answer the questions to specify the following basic settings:
 - The testing framework to use.
 - The browsers to be captured automatically.
 - The patterns that define the location of test files to be involved in testing or excluded from it. For more details, see

For more details, see [Bower: Configuration](#).

Configuring Bower in PyCharm

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS. Expand the Languages & Frameworks node, and then click Bower under JavaScript.
2. On the **Bower** page that opens, specify the location of the Node.js and Bower executable files and the `bower.json` configuration file.

Installing and Removing Bower Packages

This feature is supported in the Professional edition only.

Bower packages can be installed and used only in a specific project. PyCharm supports two installation modes: from the command line and through the dedicated user interface.

On this page:

- [Installing a Bower package in the command-line mode](#)
- [Installing a Bower package through the PyCharm interface](#)
- [Removing Bower packages](#)
- [Installing a Bower package as a development dependency](#)

Installing a Bower package in the command-line mode

1. Launch the embedded Terminal (View | Tool Windows | Terminal or by hovering your mouse pointer over  in the lower left corner of PyCharm and choosing Terminal from the menu .
2. Switch to the directory where **Bower** is stored or define a `PATH` variable for it so it is available from any folder, see [Installing Bower](#).
3. Type the following command at the command line prompt:

```
bower install <tool name>
```

Installing a Bower package through the PyCharm interface

1. Run **Bower** from PyCharm using the Bower page of the Settings dialog box.
 1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS. Expand the Languages & Frameworks node, and then click Bower under JavaScript.
 2. On the Bower page that opens, the Packages area shows all the **Bower**-dependent packages that are currently installed on your computer, both at the **global** and at the **project** level. Click **+**.
 3. In the Available Packages dialog box that opens, select the required package.
 4. Optionally specify the product version and click Install Package to start installation.

Removing Bower packages

Open the [Bower](#) page, select the package to remove, and click **-**.

Installing a Bower package as a development dependency

If a Bower package is a documentation or a test framework, which are of no need for those who are going to re-use your application, it is helpful to have it excluded from download for the future. This is done by marking the tool as a [development dependency](#), which actually means adding the Bower package in the `devDependencies` section of the `package.json` file.

With PyCharm, you can have a Bower package marked as a **development dependency** right during installation. Do one of the following:

- Run the installation from the command line in the **global** mode: launch the embedded Terminal (View | Tool Windows | Terminal or by hovering your mouse pointer over  in the lower left corner of PyCharm and choosing Terminal from the menu and type `bower install -dev <tool name>` at the command line prompt.
- Install the package using the PyCharm user interface:
 1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS. Expand the Languages & Frameworks node, and then click Bower under JavaScript.
 2. On the Bower page that opens, the Packages area shows all the Bower-dependent packages that are currently installed on your computer, both at the **global** and at the **project** level. Click **+**.
 3. In the Available Packages dialog box that opens, select the required package.
 4. Select the Options check box and type `--dev` in the text box next to it.
 5. Optionally specify the product version and click Install Package to start installation.

After installation, a Bower package is added to the `devDependencies` section of the `package.json` file.

Using File Watchers

This feature is supported in the Professional edition only.

On this page:

- [Basics](#)
- [Specifying the File Watcher name, type, and description](#)
- [Configuring the behaviour of the File Watcher](#)
- [Configuring the expected type and location of input files](#)
- [Configuring interaction with the compiler](#)
- [Configuring advanced options](#)
- [Enabling and disabling File Watchers](#)
- [Examples of customizing the behaviour of a compiler](#)

Basics

PyCharm supports integration with various third-party [compilers](#) that run in the background and perform the following:

- Translate **Less**, **Sass**, **SCSS**, and **Stylus** source code into **CSS** code.
- Translate **CoffeeScript** source code into **JavaScript** code, possibly also creating **source maps** to enable debugging.
- Compress **JavaScript** and **CSS** code.

Note that PyCharm does not contain built-in **compilers** but only supports integration with the tools that you have to download and install outside PyCharm.

In PyCharm, these compiler configurations are called **File Watchers**. For each supported compiler, PyCharm provides a predefined **File Watcher** template. Predefined **File Watcher** templates are available at the PyCharm level. To run a compiler against your project files you need to create a project-specific **File Watcher** based on the relevant template, at least, specify the path to the compiler to use on your machine.

You can download a **compiler** of your choice and set it up as a **File Watcher**. However, in this case no predefined template is available so you will have to specify all the settings manually.

To be applicable, a **File Watcher** must be enabled by selecting the check box next to it on the File Watchers page of the Settings dialog box, see [Enabling and Disabling File Watchers](#). After that the **File Watcher** will be invoked automatically upon any changes made to the source code or upon save, depending on whether the Immediate File Synchronization check box is selected or cleared, see [New Watcher Dialog](#).

The output of a **File Watcher** is stored in a separate file. The predefined templates suggest the type of the file depending on the compiler type. By default the output file is created in the same folder as the input file when the **File Watcher** is invoked for the first time, whereupon this file is only updated. You can customize all these settings during **File Watcher** creation.

JavaScript files generated by **File Watchers** are excluded from code completion and refactoring.

In the Project tree view, the output file is shown under the original file which is shown as a node. This is done to improve visibility so you can easier locate necessary files.

File watchers have two dedicated **code inspections**:

- The **File watcher available** inspection is invoked in every file that is recognized as subject for applying a predefined file watcher (Sass, Less, SCSS, or CoffeeScript). If none of the applicable predefined **File Watchers** is associated with the current project, PyCharm suggests to add one.
- The **File watcher problems** inspection is invoked by a running **File Watcher** and highlights errors specific for it.

To create a project **File Watcher** based on a predefined template, perform these general steps:

1. Make sure the **compiler** to use is downloaded and installed on your machine.
2. Make sure the **File Watchers** plugin is enabled. The plugin is bundled with PyCharm and activated by default. If the plugin is not activated, enable it on the [Plugins](#) page of the [Settings / Preferences Dialog](#) as described in [Enabling and Disabling Plugins](#).
3. Specify the File Watcher name, type, and description.
4. In the Options area, configure the behaviour of the File Watcher: the events that will invoke it, from which files it can be invoked, when you want the console shown, etc.
5. In the Watcher Settings area, configure interaction with the compiler and its behaviour: the path to the executable file, the arguments to pass to it, the acceptable input and expected output file types, etc.

Specifying the File Watcher name, type, and description

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click File Watchers under Tools.
2. The [File Watchers page](#) that opens, shows the list of **File Watchers** that are already configured in the project. Click the Add button `+` or press `Alt+Insert` and do one of the following:
 - Choose the predefined template to create the **File Watcher** from. The choice depends on the **compiler** you are going to use.
 - To set up a compiler of your choice, choose Custom.

After you choose the relevant template or the Custom option, the [New Watcher](#) dialog box opens.

3. In the Name text box, type the name of the **File Watcher**. By default, PyCharm suggests the name of the selected predefined template.

Configuring the behaviour of the File Watcher

In the Options area of the New Watcher dialog box, configure the behaviour of the **File Watcher**: the events that will invoke it, from which files it can be invoked, when you want the console shown, etc.

1. Specify whether you want the **File Watcher** to interact with the PyCharm syntax parser:
 - To have the **File Watcher** ignore update, save, and change focus events in files that are syntactically invalid and start only in error-free files, select the Trigger watcher regardless of syntax errors check box.
 - To have the **File Watcher** start regardless of the syntactical correctness of a file, clear the Trigger watcher regardless of syntax errors check box. The **File Watcher** will start upon update, save, or frame deactivation, depending on the status of the Immediate file synchronization check box.
2. Specify the events which will invoke the **File Watcher**:
 - To have the **File Watcher** invoked as soon as any changes are made to the source code, select the Immediate file synchronization check box.
 - Clear the check box to have the **File Watcher** started upon save (File | Save All) or when you move focus from PyCharm (upon frame deactivation).
3. Specify how you want the **File Watcher** to deal with dependencies. When the **File Watcher** is invoked on a file, PyCharm detects all the files in which this file is included. For each of these files, in its turn, PyCharm again detects the files into which it is included. This operation is repeated recursively until PyCharm reaches the files that are not included anywhere **within the specified scope**. These files are referred to as **root files** (do not confuse with content roots).
 - To run the **File Watcher** only against the **root files**, select the Track only root files check box.
 - Clear the check box to run the **File Watcher** against the file from which it is invoked and against all the files in which this file is recursively included within the specified scope.

This option is available only for **Babel**, **Closure Compiler**, **Compass**, **Jade**, **Less**, **Sass/SCSS**, **Stylus**, **UglifyJS**, and **YUI Compressor JS**.

4. In the Show console drop-down list, specify when you want the console shown. The available options are:
 - Always: when this option is chosen, the **File Watcher** opens the console when it starts.
 - Error: when this option is chosen, the **File Watcher** opens the console only if any errors occur in compilation.
 - Never: choose this option to suppress opening the console under any circumstances.
5. Configure the output filters to distinguish the output of the **File Watcher** from other output. These filters make the basis for:
 1. Displaying paths to the **File Watcher** output files as links in error and other messages and logs. When you click such link, the corresponding file is opened in the editor. For example, to get useful error messages displayed, specify the following expression in the Regular expression to match output field of the [Add/Edit Filter Dialog](#):

```
$FILE_PATH$: $LINE$ $MESSAGE$
```

2. Error highlighting in the output files.

Click the Output Filters button to open the [Output Filters dialog](#) and manage the list of filters.

Configuring the expected type and location of input files

In the Watched Files area, specify the type and location of files to be processed by the **File Watcher**.

1. From the File type drop-down, choose the expected type of input files. The **File Watcher** will consider only files of this type as subject for analyzing and processing. File types are recognised based on [associations between file types and file extensions](#).
By default, the field shows the file type in accordance with the chosen predefined **File Watcher** template.
2. In the Scope drop-down list, define the range of files the **File Watcher** can be applied to. Changes in these files will invoke the **File Watcher** either immediately or upon save or frame deactivation, depending on the status of the Immediate file synchronization check box.
You can choose one of the predefined scopes from the drop-down list:
 - Project Files: all the files within the project content roots (see [Content Root](#) and [Configuring Content Roots](#)).
 - Project Production Files: all the files within the project content roots excluding test sources.
 - Project Test Files: all the files within the project test source roots.
 - Open Files: all the files that are currently opened in the editor.

VCS Scopes: these scopes are only available if your [project is under version control](#).

- Changed Files: all changed files, that is, all files associated with all existing changelists.
- Default: all the files associated with the changelist `Default`.

Alternatively, click the Browse button and configure a **custom scope** in the **Scopes** dialog box that opens.

For more details on scopes, see the pages [Scopes](#) and [Scopes dialog](#).

The Scope setting overrides the Track only root files check box setting: if a dependency is outside the specified scope, the **File Watcher** is not applied to it.

Configuring interaction with the compiler

In the Watcher Settings area, configure interaction with the **compiler**: specify the path to the executable file, the arguments to pass to it, the acceptable input and expected output file types, etc.

In the Program text box, specify the location of the **compiler's** executable file (`java`, `cmd`, `bat`, or other depending on the specific tool).

- In the Program text box, specify the location of the **compiler's** executable file (`.exe`, `.cmd`, `.bat` or other depending on the specific tool). `.jar` archives are also acceptable but defining `PATH` variables for them is not supported. Do one of the following:
 - Type the path explicitly.
 - Click the Browse button  to open the Select Path dialog box and navigate to the desired location.
 - Click the Insert Macro button to open the [Macros](#) dialog box where you can select the relevant macro from the list.
- In the Arguments text box, define the arguments to pass to the **compiler**. Among other cases, use this text box to change the default output location, that is, specify a custom location where you want the **compiler** to store the files generated during compilation. Note that if you re-define the default output location here you need to clear the Create output file from stdout check box in the Other Options area because otherwise the content of your generated file will be overwritten by the compiler's output stream.

Do one of the following:

 - Type the list of arguments in the text box.
 - Click the Insert Macro button to open the [Macros](#) dialog box where you can select the desired macro from the list.

When specifying the arguments, follow these rules:

 - Use spaces to separate individual arguments.
 - If an argument includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some "arg"` or `"some arg"`.
 - If an argument includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `Dmy.prop="\quoted_value\"`.
- In the Output paths to refresh text box, specify the files where the **compiler** stores its output: the resulting source code, source maps, and dependencies. Based on this settings, PyCharm detects files generated through compilation.

Please note, that changing the value in this text box does not make the **compiler** store its output in another location. To do that, specify the desired output location in the Arguments text box.

Do one of the following:

 - Type the output paths manually. Use colons as separators.
 - Click the Insert Macro button to open the [Macros](#) dialog box, where you can select the desired pattern from the list.

Configuring advanced options

In the Other Options area, specify advanced compiler configuration settings.

- In the Working Directory text box, specify the directory to which the **compiler** will be applied. Because the tool is always invoked in the context of a file, the default working directory is the directory of the current file. This setting is specified in all predefined templates through a macro `$FileDir$`. To update this default settings, do one of the following:
 - Type the path explicitly in the text box.
 - Click the Browse button  to open the Select Path dialog box and navigate to the desired location.
 - Click the Insert Macro button to open the [Macros](#) dialog box, where you can select the desired macro from the list.

Tip If the field is left blank, PyCharm uses the directory of the file where the **File Watcher** is invoked.

- Use the Create output file from stdout check box to specify how you want the output file generated.
 - To have PyCharm read the native **compiler's** output (`standard output stream (stdout)`) and generate the resulting files from it, select the Create output file from stdout check box.
 - To have the **compiler** write its output directly to the files specified in the Output paths to refresh field, clear the check box.

Tip Some compilers generate a `standard output stream (stdout)` file, others do not, which may lead to errors. Therefore it is strongly recommended to preserve the default setting.

- Optionally, define environment variables. For example, specify the `PATH` variable for the tool that is required for starting the **compiler** but is not referenced in the path to it. In most cases it is **Node.js** or **ruby.exe**. Such situation may happen if you installed the **compiler** manually but not through the **Node Package Manager** or **gem manager** and its installation folder is not under **Node.js** or **ruby**.

Enabling and disabling File Watchers

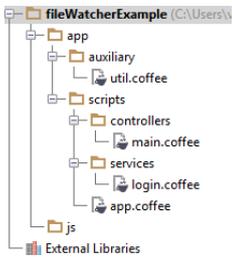
To toggle the enable/disable status of a **File Watcher**, select/clear the check box next to it on the File Watchers page of the Settings dialog box. If an error occurs while a **File Watcher** is running, the **File Watcher** is automatically disabled. To restore the status, enable the **File Watcher** manually.

When a **File Watcher** is enabled, it starts automatically as soon as a file to which compilation is applicable is changed or saved, see [Configuring the behaviour of the File Watcher](#).

Examples of customizing the behaviour of a compiler

Any **compiler** is an external, third-party tool. Therefore the only way to influence a **compiler** is pass arguments to it just as if you were working in the command line mode. These arguments are specific for each tool. Below are two examples of customizing the default output location for the **CoffeeScript compiler**.

Suppose, you have a project with the following folder structure:



By default, the generated files will be stored in the folder where the original file is. You can change this default location and have the generated files stored in the `js` folder. Moreover, you can have them stored in a flat list or arranged in the folder structure that repeats the original structure under the `app` node.

- To have all the generated files stored in the output `js` folder without retaining the original folder structure under the `app` folder:

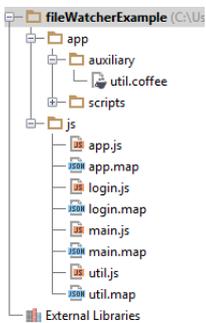
1. In the Arguments text box, type:

```
--output $ProjectFileDir\js\ --compile --map $FileName$
```

2. In the Output paths to refresh text box, type:

```
$ProjectFileDir\js\${FileNameWithoutExtension}.js:$ProjectFileDir\js\${FileNameWithoutExtension}.map
```

As a result, the project tree looks as follows:



- To have the original folder structure under the `app` node retained in the output `js` folder:

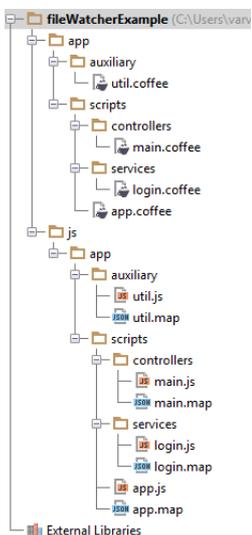
1. In the Arguments text box, type:

```
--output $ProjectFileDir\js\${FileDirRelativeToProjectRoot}\ --compile --map $FileName$
```

2. In the Output paths to refresh text box, type:

```
$ProjectFileDir\js\${FileDirRelativeToProjectRoot}\${FileNameWithoutExtension}.js:$ProjectFileDir\js\${FileDirRelativeToProjectRoot}
```

As a result, the project tree looks as follows:



Using ReactJS in JavaScript and TypeScript

This feature is supported in the Professional edition only.

[ReactJS](#) introduces [JSX](#) and [TSX](#), an XML-like syntax that you can use inside your [JavaScript](#) and [TypeScript](#) code respectively.

On this page:

- [Installing ReactJS](#)
- [Enabling ReactJS support in JavaScript code](#)
- [Enhancing code completion with typed parameter information](#)
- [Using ReactJS with TypeScript](#)

Installing ReactJS

Download the [ReactJS Starter Kit](#) and extract the files from the archive.

Enabling ReactJS support in JavaScript code

PyCharm recognizes [JSX](#) code and provides syntax highlighting, code completion, navigation, and code analysis for it. Code completion and navigation is also provided for [ReactJS](#) methods. See [Auto-Completing Code](#) and [Navigating to Declaration or Type Declaration of a Symbol](#).

PyCharm can also provide code completion for [HTML](#) tags and component names that you have defined inside methods in [JavaScript](#) or inside other components.

Completion also works for imported components with [ES6](#) style syntax. You can navigate from a component name to its definition with `Ctrl+B` or see a definition in a popup with `Ctrl+Shift+I`.

In [JSX](#) tags, PyCharm provides coding assistance for [ReactJS](#)-specific attributes such as `className` or `classID`. Moreover, for class names you can autocomplete classes defined in the project's CSS files.

Code completion is also provided for [JavaScript](#) expressions inside curly braces. Code completion applies to all methods and functions that you have defined.

PyCharm supports [Emmet](#) in [JSX](#) code thus enabling some [ReactJS](#)-specific twists. For example, the abbreviation `div.my-class` would expand in [JSX](#) to `<div className="my-class"></div>` and not to `<div class="my-class"></div>` as it would in [HTML](#).

To get coding assistance with [JSX](#) or [JavaScript](#):

1. Make sure you have the `react.js` file anywhere under the project root folder, possibly copy it from the downloaded [ReactJS Starter Kit](#) after extraction. Alternatively, configure `react.js` as an external JavaScript library, see [Configuring JavaScript Libraries](#).
2. Do one of the following:
 - Create a `.jsx` file to write your code in. PyCharm recognizes this file type and provides coding assistance in it.
 - To use the same coding assistance in `.js` files, change the language level to JSX Harmony: Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click JavaScript under Languages & Frameworks. On the [JavaScript](#) page that opens, choose JSX Harmony from the JavaScript Language Version drop-down list.

Enhancing code completion with typed parameter information

To see information on parameters that you can use in a method, configure a TypeScript definition file `react.d.ts` as a JavaScript library and add it to your project:

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS. Expand the Languages & Frameworks node, and then click Libraries under JavaScript.
2. On the [JavaScript. Libraries](#) page that opens, click Download on the toolbar.
3. In the Download Library dialog box that opens, select `react` from the list of stubs, and click Download and Install. You return to the [JavaScript. Libraries](#) page where the library is added to the list as `Global` and is already activated.
4. Copy the `react.d.ts` file to your project.

Using ReactJS with TypeScript

PyCharm recognizes and supports [ReactJS](#) syntax in [TypeScript](#) context ([TSX](#)) only in `.tsx` files. To use [ReactJS](#) with [TypeScript](#) code:

1. Add a TypeScript definition file `react.d.ts` to your project as described above in the section [Enhancing Code Completion with Typed Parameter Information](#) or add the `react.js` to your project.
2. Write your code in `.tsx` files.

Vagrant: Working with Reproducible Development Environments

Capability to configure remote interpreters, detecting path mappings from Vagrant configuration file, UI for starting Vagrant

This feature is supported in the Professional edition only.

In this section:

- Vagrant: Working with Reproducible Development Environments
 - [Prerequisites](#)
 - [Basics](#)
 - [Vagrant support](#)
 - [Preparing to work with Vagrant](#)
 - [Initializing the Vagrantfile](#)
 - [Creating and launching an instance \(virtual machine\)](#)
 - [Stopping, suspending, resuming, reloading, and destroying an instance \(virtual machine\)](#)
- [Creating and Removing Vagrant Boxes](#)
- [Initializing Vagrant Boxes](#)

Prerequisites

Before you start working with Vagrant, make sure that the Vagrant plugin is enabled. The plugin is bundled with PyCharm and is activated by default. If the plugin is not activated, enable it on the [Plugins](#) page of the [Settings / Preferences Dialog](#) as described in [Enabling and Disabling Plugins](#).

Besides that, make sure that the following prerequisites are met (outside of PyCharm):

- [Oracle's VirtualBox](#) is installed on your computer.
- [Vagrant](#) is installed on your computer, and all the necessary infrastructure is created.
- The parent folders of the following executable files are added to the system **PATH** variable:
 - `vagrant.bat` or `vagrant` from your Vagrant installation. This should be done automatically by the installer.
 - `VBoxManage.exe` or `VBoxManage` from your Oracle's VirtualBox installation.
- The required virtual boxes are created.

Basics

Integration with [Vagrant](#) helps you create reproducible development environments defined by **VagrantFile** configuration files. To have the same environment set up on the machines of all the team members you only need to put the **VagrantFile** under your team version control. Any team member can set up the required environment by running the `vagrant up` command with the relevant **VagrantFile**. Vagrant support in PyCharm is provided through the **Vagrant** plugin that comes bundled with PyCharm .

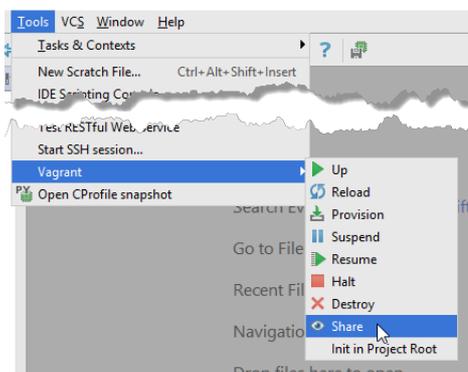
In the context of PyCharm, the following terms are used to denote Vagrant-specific notions:

- A **Vagrant box** in the PyCharm user interface or **Vagrant base box** in the current documentation is a [box](#) in the native Vagrant terminology. It denotes a pure image, a skeleton, on the base of which a specific environment is customized, provisioned, and deployed on your machine.
- An **instance** is a **virtual machine**. In other words, it is a specific environment customized, provisioned, and deployed on your machine on the base of a **Vagrant base box** and in accordance with a **VagrantFile**. In other similar products and documentation, an **instance** can be referred to as **virtual machine**.
- An **instance folder** is the folder where the relevant **VagrantFile** is stored after initialization and where PyCharm will look for it. By default, it is the project root folder.

Vagrant support

Once the Vagrant plugin is enabled, the following changes are made to the PyCharm UI:

- A [Vagrant page](#) is added to the Settings dialog.
- A Vagrant node is added to the Tools menu. The node contains the following commands that correspond to the standard Vagrant actions:



Vagrant support in PyCharm features:

- capability to [create new virtual boxes](#), and [delete](#) the unnecessary ones.
- capability to [configure remote interpreters](#) by reading settings from the Vagrant configuration file.
- capability to [initialize Vagrant boxes](#), and execute other Vagrant commands without leaving the IDE.
- Multiple Vagrant configuration: when performing the `vagrant up` command, you are suggested to select the desired virtual box.

Preparing to work with Vagrant

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS, and click Vagrant under Tools. The Vagrant page opens.
2. Specify the **Vagrant** executable file. Because the parent folder of the executable file has been already added to the system **PATH** variable, type just the name of the executable.
3. In the Instance Folder field, specify the fully qualified path to the directory where the `VagrantFile` is initialized and stored. A `VagrantFile` is a configuration file that defines the **instance (virtual machine)** you need. The file contains the virtual IP address, port mappings, and the memory size to assign. The file can specify which folders are shared and which third-party software should be installed. According to the `VagrantFile` your **instance (virtual machine)** is configured, provisioned against the relevant **Vagrant base box**, and deployed on your computer. A `VagrantFile` is created through the `vagrant init` command.
When creation of an **instance (virtual machine)** is invoked either through the `vagrant up` command or through the Tools | Vagrant | Up menu option, PyCharm looks for the `VagrantFile` in the directory specified in the Instance folder field. For more information, see <http://docs.vagrantup.com/v2/vagrantfile/>.
You can create a `VagrantFile` in any directory and appoint it as **instance folder**. If the field is empty, PyCharm will treat the **project root** as the **instance folder** and look for a `VagrantFile` in it.
4. In the Vagrant Boxes area, configure a list of the predefined [Vagrant base boxes](#) available in PyCharm. Each item presents a **Vagrant base box** on which Vagrant configures and launches its **instances (virtual machines)**. The entries of this list correspond to the output of the command `vagrant box list`.
 - To download a new **base box**, click the Add button `+`. In the dialog box that opens, specify the URL address to access the **base box** and the name to refer to it in PyCharm. By default, PyCharm suggests the URL to the **lucid32** box.
This command corresponds to `vagrant box add <name> <URL>`. As a result, the specified **base box** is downloaded to your machine.
 - To remove a **base box**, select it in the list and click the Remove button `-`. The **base box** and the nested files are physically deleted from the disk. This command corresponds to `vagrant box remove <name>`.

Initializing the Vagrantfile

A `VagrantFile` is a configuration file that defines the **instance (virtual machine)** you need. The file contains the virtual IP address, port mappings, and the memory size to assign. The file can specify which folders are shared and which third-party software should be installed. According to the `VagrantFile` your **instance (virtual machine)** is configured, provisioned against the relevant **Vagrant base box**, and deployed on your computer. A `VagrantFile` is created through the `vagrant init` command.

You can initialize the **Vagrantfile** in any folder, just keep in mind that this folder should be specified as the **instance folder** on the Vagrant page of the Settings dialog box. Otherwise PyCharm will be unable to find the relevant **VagrantFile** during the **instance (virtual machine)** creation.

To initialize the **VagrantFile**, do one of the following:

To initialize the Vagrantfile

1. To have the **Vagrantfile** created in the project root, choose Tools | Vagrant | Init on Project Root on the main menu and select the target project root from the pop-up list that opens. The output of the `init` command is displayed in the Run tool window.
2. To have the **Vagrantfile** created in a specific folder, launch the embedded Terminal (View | Tool Windows | Terminal or by hovering your mouse pointer over  in the lower left corner of PyCharm and choosing Terminal from the menu and then type the following commands at the prompt:

```
cd <directory to initialize the VagrantFile in>
vagrant init <base box name> <base box url>
```

Once the **VagrantFile** initialization is successfully completed, you are ready to import the **base box**, provision and deploy it according to the **VagrantFile**, thus creating your own **instance (virtual machine)**.

Creating and launching an instance (virtual machine)

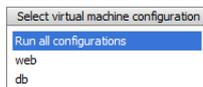
Creating an **instance (virtual machine)** means importing and provisioning a **base box** according to the **VagrantFile** located in the **instance folder**.

To create an instance, follow these steps:

1. Do one of the following:
 - To have the instance created in the project root, choose Tools | Vagrant | Up.
 - Launch the embedded Terminal (View | Tool Windows | Terminal or by hovering your mouse pointer over  in the lower left corner of PyCharm and choosing Terminal from the menu and then type the following commands at the command line prompt:

```
cd <instance folder>
vagrant up
```

2. Select the desired virtual machine configuration from the suggestion list:



Stopping, suspending, resuming, reloading, and destroying an instance (virtual machine)

To reload an instance

- If you have made some changes to the **VagrantFile** and want a running virtual machine updated in accordance to them, choose Tools | Vagrant | Reload on the main menu or run the following command in the embedded Terminal:

```
vagrant reload
```

For details, see <http://docs.vagrantup.com/v2/cli/reload.html>.

To suspend an instance

- To temporary stop the operating system on a running virtual machine (**guest machine**) and save the exact state of the environment as it is at this moment so it can be resumed exactly from this point, choose Tools | Vagrant | Suspend on the main menu or run the following command in the embedded Terminal:

```
vagrant suspend
```

For details, see <http://docs.vagrantup.com/v2/cli/suspend.html>.

To resume an instance

- To resume a previously suspended operating system on a virtual machine (**guest machine**) and have it run from the state saved at the suspension moment, choose Tools | Vagrant | Resume on the main menu or run the following command in the embedded Terminal:

```
vagrant resume
```

For details, see <http://docs.vagrantup.com/v2/cli/resume.html>.

To shut an instance down

- To shut down the operating system on a virtual machine (**guest machine**), choose Tools | Vagrant | Halt on the main menu or run the following command in the embedded Terminal:

```
vagrant halt
```

For details, see <http://docs.vagrantup.com/v2/cli/halt.html>.

- To shut down the operating system on a virtual machine (**guest machine**), stop the virtual machine itself, and remove the resources provisioned on it during the creation (`vagrant up`), choose Tools | Vagrant | Destroy on the main menu or run the following command in the embedded Terminal:

```
vagrant destroy
```

For details, see <http://docs.vagrantup.com/v2/cli/destroy.html>.

Creating and Removing Vagrant Boxes

This feature is supported in the Professional edition only.

On this page:

- [Introduction](#)
- [Creating a Vagrant box](#)
- [Deleting a Vagrant box](#)

Introduction

It is possible to work with multiple virtual boxes, created in Vagrant. With PyCharm, you can [add new boxes](#) without leaving the IDE.

The unnecessary boxes can be easily [deleted](#).

Creating a Vagrant box

1. [Open the Settings dialog box](#).
2. Under the Project Settings, click [Vagrant](#).
3. Click **+** button, located next to the Vagrant Boxes table.
4. In the Add Vagrant Box dialog box, specify the name of the new box, and its URL. Then click OK.
The new box is added to the list of available boxes.

Deleting a Vagrant box

1. In the [Vagrant](#) page of the project settings, under the Vagrant Boxes table, select the box to be deleted.
2. Click **-**, and confirm deletion.

The selected box is removed from the list.

Initializing Vagrant Boxes

This feature is supported in the Professional edition only.

On this page:

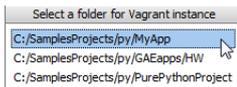
- [Basics](#)
- [Initializing a virtual box](#)
- [Activating a virtual box](#)

Basics

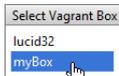
PyCharm makes it possible to execute the `init` procedure on a project root. After performing `init` on a project root, the Vagrant configuration file named `Vagrantfile` is created there. This file can be used for [creating a remote interpreter](#).

Initializing a virtual box

1. On the main menu, choose Tools | Vagrant | Init in Project Root.
2. In the Select Vagrant Folder pop-up menu, select the project root to be used:



3. Since it is possible to have multiple virtual boxes, specify the particular box you want to work with:



The `vagrant init` command is launched, and shows its output messages in the Run tool window.

If the Vagrant executable is not specified in the [Vagrant](#) page, you can still initialize a Vagrant box. However, in this situation, PyCharm pops up a file browser, enabling you to select Vagrant executable.

Activating a virtual box

Once `init` is successfully completed, you are ready to perform the `vagrant up` command and import to the root the box used at initialization.

To activate a virtual box by executing the `vagrant up` command, choose Tools | Vagrant | Up on the main menu. The `vagrant up` command is launched, and shows its output messages in the Run tool window.

Web2Py

This feature is supported in the Professional edition only.

PyCharm supports [Web2Py](#) development.

In this section:

- Web2Py
 - [Web2Py Support](#)
- [Creating Web2Py Project](#)

Web2Py Support

Web2Py support in PyCharm includes:

- Dedicated [project type](#) .
- Possibility to open an existing Web2Py project. So doing, PyCharm will recognize project structure, choose the proper template language, and create run/debug configurations.
- When a new Web2Py project is created, PyCharm creates a dedicated run/debug configuration for launching a web2Py server.
- [Web2Py template language](#).
- [Navigation between views and templates](#) using the gutter icons.
- [Code completion](#) and resolve.

Creating Web2Py Project

This feature is supported in the Professional edition only.

[Web2Py](#) project is intended for productive development of Web2Py applications. PyCharm takes care of creating the specific directory structure, and settings.

To create a Web2Py project, follow these steps

1. Do one of the following:

- On the main menu, choose File | New | Project.
- Click New Project in the [Welcome screen](#).

Create New Project dialog box opens.

2. In the [Create New Project](#) dialog box, specify the following:

- Project name and location.
- Python interpreter to be used for the project.
If the desired interpreter is not found in the list, click the browse button to review the available interpreters and virtual environments, and [configure the new ones](#).

So doing, if Web2Py is missing, PyCharm informs you that Web2Py will be installed on the current project interpreter.

3. In the More Settings section, do the following:

1. Specify the desired application name.
2. If necessary, select the check box Local Web2Py.
If this check box is selected, then the local Web2Py sources will be used; you can download Web2Py sources from [here](#).

If this check box is not selected, then the entire Web2Py will be [downloaded](#).

4. Click Create.

PyCharm creates an application and produces specific directory structure, which you can explore in the Project tool window. If there are [unsatisfied requirements](#), PyCharm suggests to resolve or ignore them.

Migration Guides and Tutorials

- [Code Quality Assistance Tips and Tricks, or How to Make Your Code Look Pretty?](#)
- [Code Running Assistance](#)
- [Configuring Generic Task Server](#)
- [Configuring PyCharm to Work on the VM](#)
- [Creating and Applying Live Templates \(Code Snippets\)](#)
- [Debugging JavaScript with PyCharm](#)
- [Debugging Python Code](#)
- [Deployment in PyCharm](#)
- [Exploring Navigation and Search](#)
- [File Watchers in PyCharm](#)
- [Finding and Replacing Text in File Using Regular Expressions](#)
- [Migrating from Text Editors](#)
- [PyCharm Refactoring Tutorial](#)
- [TODO Example](#)
- [Using the Advanced Vagrant Features in PyCharm](#)
- [Using Live Templates in TODO Comments](#)
- [Using IPython/Jupyter Notebook with PyCharm](#)
- [Using the PyCharm Built-in SSH Terminal and Remote SSH External Tools](#)
- [Using Emacs as an External Editor in PyCharm](#)
- [Using TextMate Bundles](#)
- [Using Vim Editor Emulation in PyCharm](#)

Code Quality Assistance Tips and Tricks, or How to Make Your Code Look Pretty?

In this section:

- [What this tutorial is about](#)
- [Before you start](#)
- [Highlighting code style violations](#)
- [Tuning the PEP8 inspections](#)
- [Tracking PEP8 rules](#)
- [Code inspections and their settings](#)
 - [Creating scopes](#)
 - [Creating inspection profile with these scopes](#)
- [Highlighting errors](#)
- [Generating source code](#)
- [Reformatting code](#)
- [Adding documentation comments](#)
- [Type hinting](#)

What this tutorial is about

This tutorial aims to walk you step by step through creating source code in a Python project, with the use of PyCharm's code intelligence features. You will see how PyCharm helps keep your source code in a perfect shape, with proper indentations, spaces, imports etc. - actually, you'll see that PyCharm itself is a code quality tool.

Python programming is out of scope of this tutorial. To learn more about the Python language, please refer to the [official website](#).

Before you start

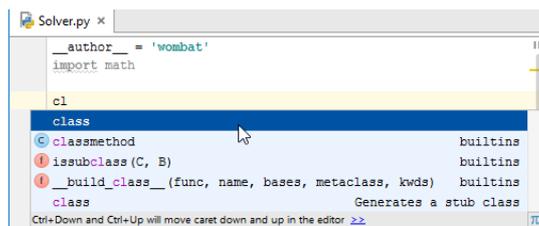
Make sure that:

- You are working with PyCharm version 5.0 or later. If you still do not have PyCharm, download it from [this page](#). To install PyCharm, follow the instructions, depending on your platform. Refer to the [product documentation](#) for details.
- You have created a Python project (File|New Project...). Refer to the [product documentation](#) for details
- You have created two directories `src` and `test_dir` (File|New or `Alt+Insert`). To learn about creating directories, refer to the [product documentation](#).
- You have added Python files to the `src` and `test_dir` directories of your project (File|New or `Alt+Insert`). To learn about creating files, refer to the section [Populating Projects](#).

Highlighting code style violations

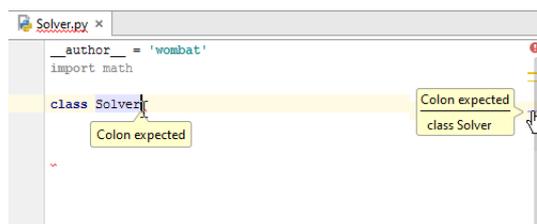
Open a new Python file for editing (`F4`). The file by default has two lines: the author and the project names. These lines appear because the file `Solver.py` is created by a [file template](#) that (in the case of Python files) contains definitions of these variables.

Next, start typing the keyword `class`. When you just start typing, PyCharm immediately shows the suggestion list to complete your code:



(Refer to [Code Completion](#) page of the product documentation for details.)

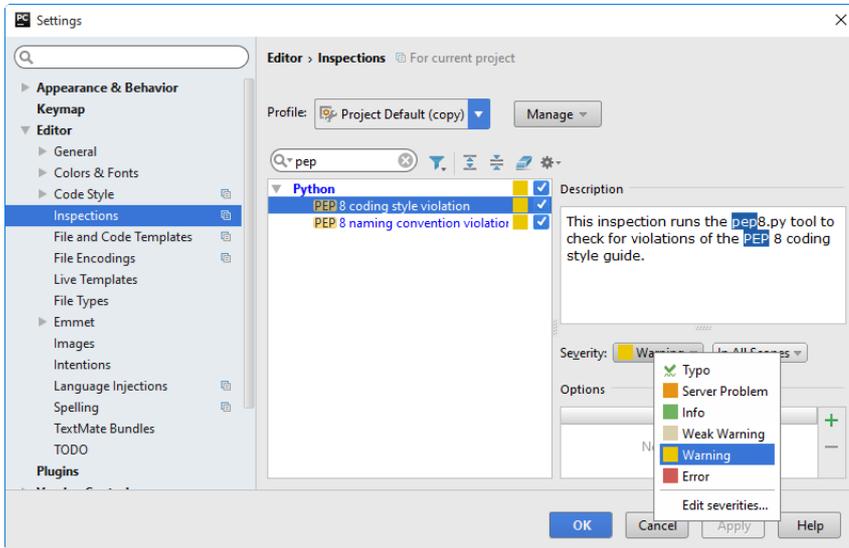
The red curve marks the next expected entry - in this case, this is the expected identifier. Enter the class name `Solver`. The red curve moves after the class name. If you hover your mouse pointer over this curve, you see the error description ("Colon expected"). Also, mind the red error stripe in the right gutter - it also marks the same error:



OK, type the colon, and press `Enter`. According to the [Python code style](#), the next statement is indented. If by chance you press space after `Enter`, you will thus violate the code style settings.

Tuning the PEP8 inspections

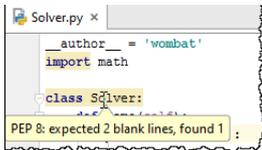
However, by default these violation are but week warnings, and as such, are not visible. So, at first, let's raise their importance. Click  on the main toolbar, on the [Inspections page](#) of the Settings/Preferences dialog, type PEP8 to find all PEP8-related inspections, and from the Severity drop-down list, choose Warning:



Apply changes and close the dialog. Now let's return to our source code.

Tracking PEP8 rules

Now PyCharm shows its best! It stands on guard to protect your code style integrity. You immediately note that indented space is highlighted, and, when you type the next statement, for example, `def demo(self, a, b, c):`, PyCharm will show the message from the PEP8 inspection:



So, as you can see, PyCharm supports [PEP8](#) as the official Python style guide. If you explore the list of inspections (`Ctrl+Alt+S` - [Inspections](#)), you will see that PyCharm launches the `pep8.py` tool on your code, and pinpoints the code style violations.

Code inspections and their settings

Btw, look at the [Inspections](#) more attentively. If you have just opened this page, you see the [default inspection profile](#) with the default settings: it means that the inspections apply to all the sources of the current project.

Let's try to customize this profile for two different [scopes](#):

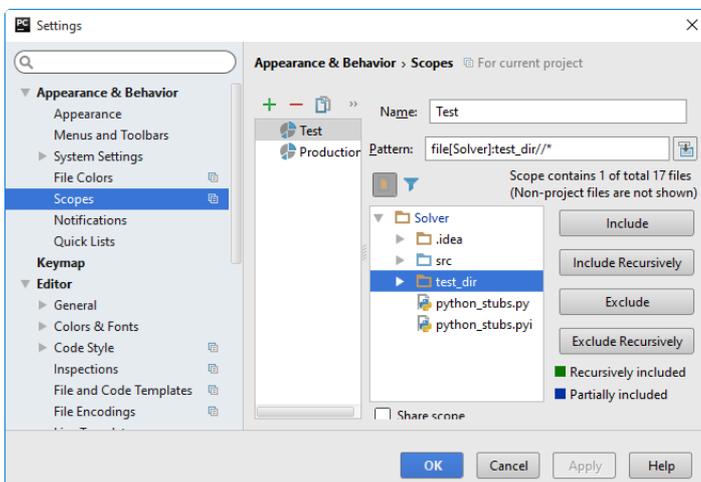
- In the **Test** scope, the spelling errors should be marked as typos (green)
- In the **Production** scope, the spelling errors should be marked as errors (red) - can we actually produce code with typos?

This is how it's done...

Creating scopes

First, let's define the two scopes. To do that, click on the main toolbar, in the Settings/Preferences dialog box expand the node Appearance and Behavior, open the page [Scopes](#). Then click and choose scope type Local.

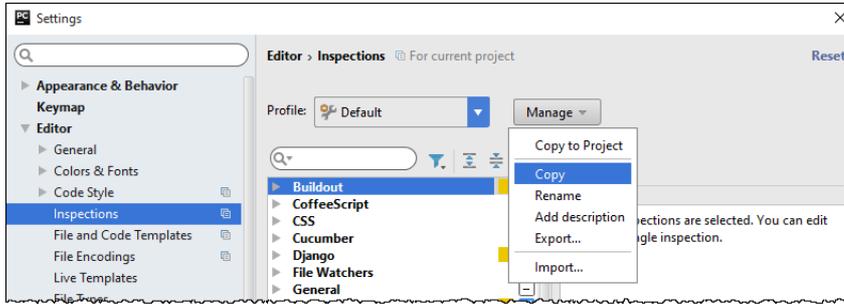
In the Add New Scope dialog box, type the scope name (Test), and then, in the project tree, choose the directory to be included in the Test scope, `test_dir`. Note that the Pattern field is filled in automatically, as you include the directory:



Repeat this process to create the Production scope.

Creating inspection profile with these scopes

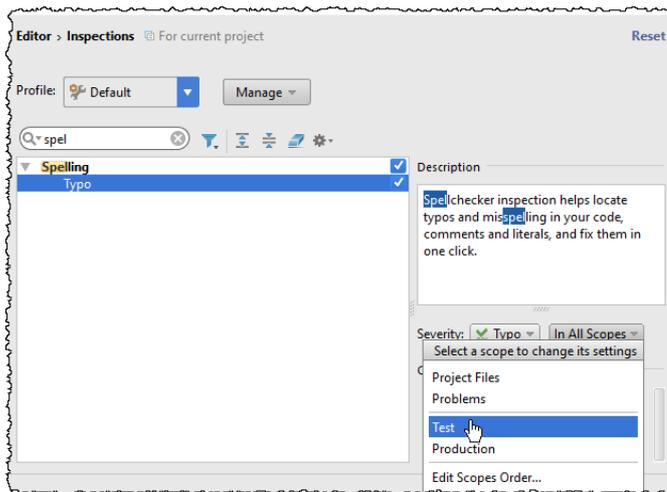
Next, let's create a copy of the default profile (though this profile is editable... just to be on the safe side):



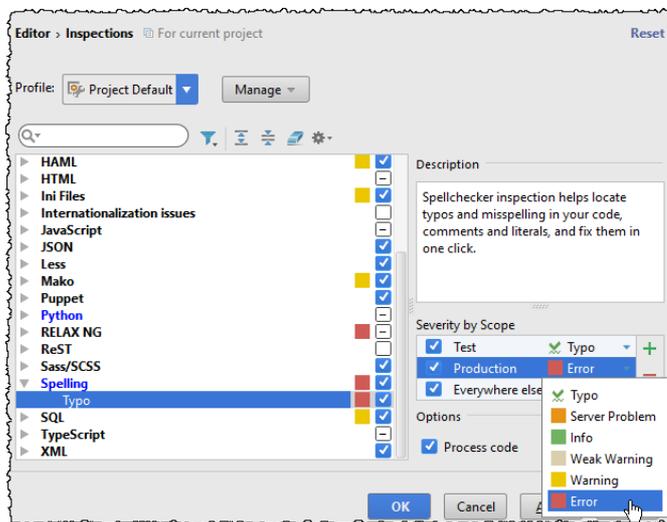
and give it a new name, for example, MyProjectProfile. This new profile is a copy of the default one, and has the same set of inspections.

With this new profile selected, let's locate the Spelling inspection and change it. To find the Spelling inspection (we've already done it before), just type `spel` in the search area.

What's next? Click In All Scopes button and select the Test scope from the list; repeat same for the Production scope



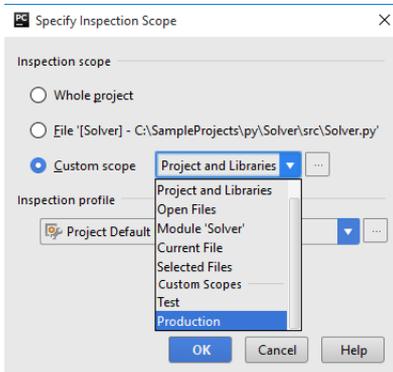
In the scope "Test", the inspection severity is left as is (a typo); however, the scope "Production" we'll choose "Error" from the list of severities:



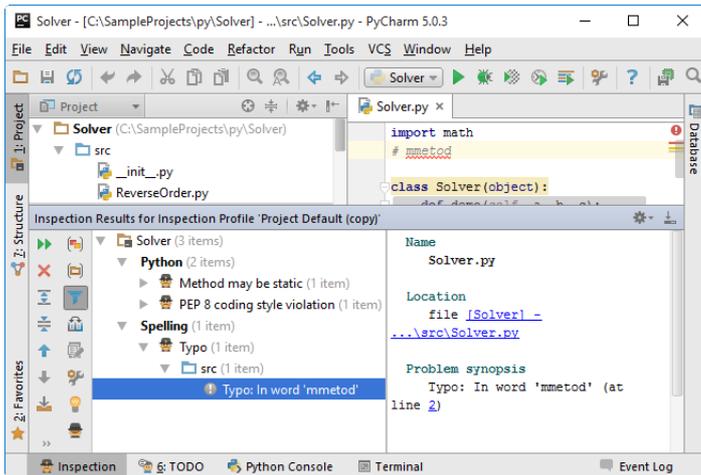
Mind the color code of inspections. They are shown black if unchanged. If they are blue, then it means that they have been changed.

Apply changes and close the dialog...

So, the modified inspection profile is ready. Its name is Project Default (copy), and it has different settings for the Spelling inspection in the Test and Production scopes. Next, let's inspect code against this profile. To do that, choose Code>Inspect Code on the main menu, and in the dialog box, choose the desired profile and scope:



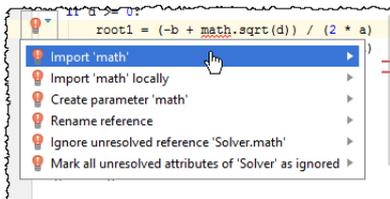
Do it twice - for Test and Production scopes (if you want to preserve inspection results for further examination and sharing, you can [export them](#)). Explore results:



Highlighting errors

Besides coding style violations, PyCharm highlights the other errors too, depending on the selected profile.

For example, if your inspection profile includes Python inspection Unresolved references, and you use a symbol that not yet has been imported, PyCharm underlines the unresolved reference and suggests to add import statement:



Refer to the product [documentation](#).

Anyway, you can change the error highlighting with some aid from [Hector](#).

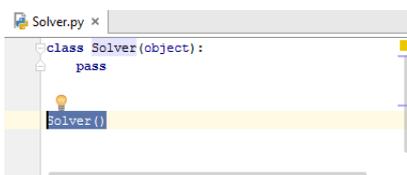
Generating source code

PyCharm provides lots of possibilities to automatically generate code. You can explore the auto-generation features in the [product documentation](#). Let's explore the main code generation procedures. To do that, just delete all contents of the file `Solver.py`, and start from scratch.

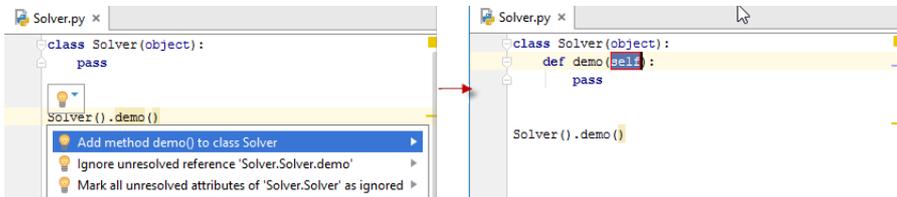
First, create an instance of a class:



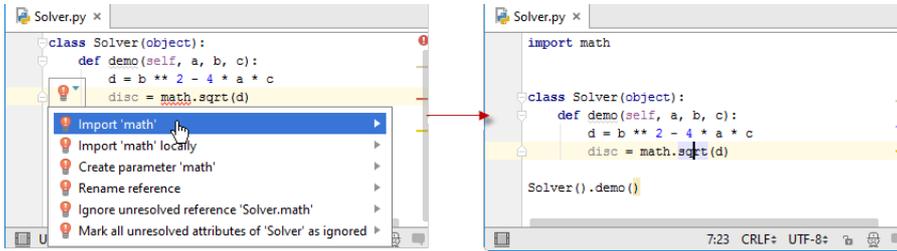
Great! PyCharm has stubbed out a class:



Next, let's add a method to the class instance. To do that, type a dot after class instance, and then type the method name. This method does not yet exist, and PyCharm suggests to create one:



Let's do some manual work - type the source code. When it comes to calculate the discriminant, we have to extract a square root... There is a dedicated function `sqrt` in the library `math`, but it is not yet imported. OK, let's type it anyway, and see how PyCharm copes with it:

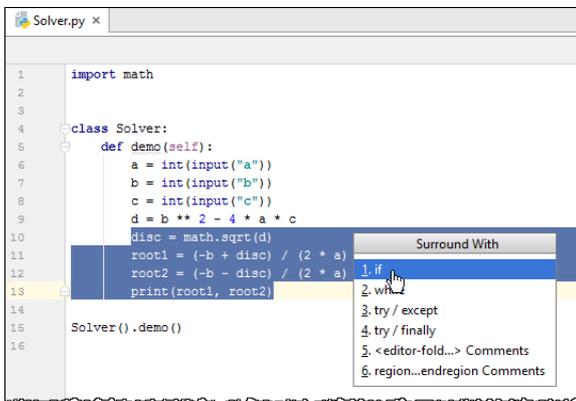


So, we've come to the source code like this:

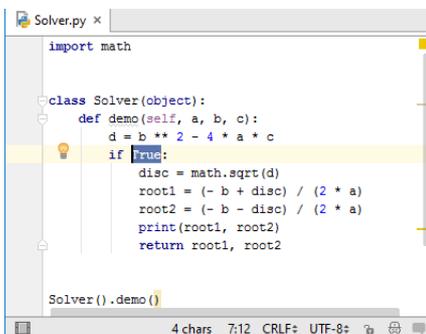
```
import math
class Solver(object):
    def demo(self,a,b,c):
        d = b ** 2 - 4 * a * c
        disc = math.sqrt(d)
        root1 = (- b + disc) / (2 * a)
        root2 = (- b - disc) / (2 * a)
        print (root1, root2)
        return root1, root2
```

However, it lacks some significant analysis. We'd like to analyze the radicand `d`. If it is zero or positive, then the discriminant and the equation roots will be calculated; when the radicand is negative, let's raise an exception. How PyCharm will help completing this task?

Let's surround a block of code with `if` construct. Select the statements to be completed, when `d` is non-negative, and press `Ctrl+Alt+T` (or choose Code | Surround with on the main menu):



Select `if` option from the suggestion list. As you see, PyCharm automatically adds `if True:` and indents the selected lines:



We are not at all interested in a boolean expression, so let's change the selected `True` to `d >= 0`. Next, place the caret at the end of the last line, and press `Enter`. The caret rests on the next line, with the same indentation as the `if` statement; type `else:` clause here, and see PyCharm reporting about the expected indentation:

```

import math

class Solver(object):
    def demo(self, a, b, c):
        d = b ** 2 - 4 * a * c
        if d >= 0:
            disc = math.sqrt(d)
            root1 = (- b + disc) / (2 * a)
            root2 = (- b - disc) / (2 * a)
            print(root1, root2)
            return root1, root2
        else:
            raise ValueError('discriminant is negative')

```

When you press `Enter` again, the caret rests at the indented position. Here you can type the exception expression, using PyCharm's powerful automatic code completion:

```

from django.db import models

class Poll(models.Model):
    Model

```

Reformatting code

Let's look again at our `Solver.py` file. Its right gutter shows yellow stripes. When you hover your mouse pointer over a stripe, PyCharm shows the description of the corresponding problem in the code:

```

import math

class Solver(object):
    def demo(self, a, b, c):
        d = b ** 2 - 4 * a * c
        if d >= 0:
            disc = math.sqrt(d)
            root1 = (- b + disc) / (2 * a)
            root2 = (- b - disc) / (2 * a)
            print(root1, root2)
            return
        else:
            raise ValueError('discriminant is negative')

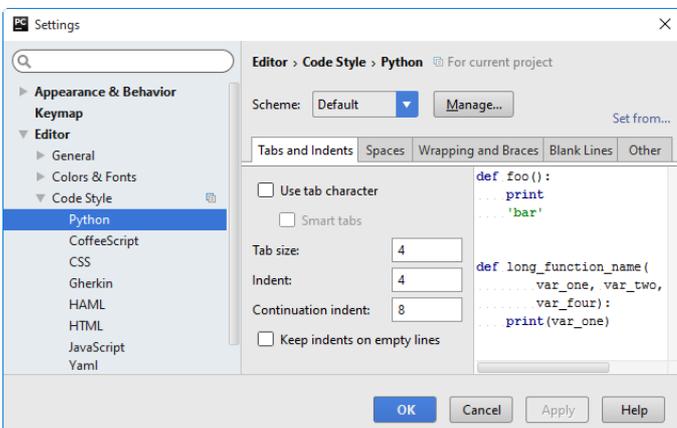
```

The good news is that they are but warnings, and won't affect the results. Bad news is they are too numerous to fix each one by one. Is it possible to make the source code nice and pretty without much fuss?

PyCharm says - yes. This is the [code reformatting feature](#). So let's try to change formatting of the entire file. To do that, press `Ctrl+Alt+L` (or choose `Code | Reformat Code` on the main menu):

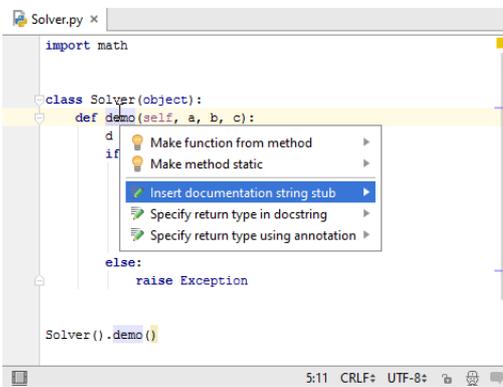
Look at the code now - the PEP8-related drawbacks are all gone.

Note that you can define formatting rules yourself. To do that, open the code [style settings](#), select language (in this case, Python), and make the necessary changes:

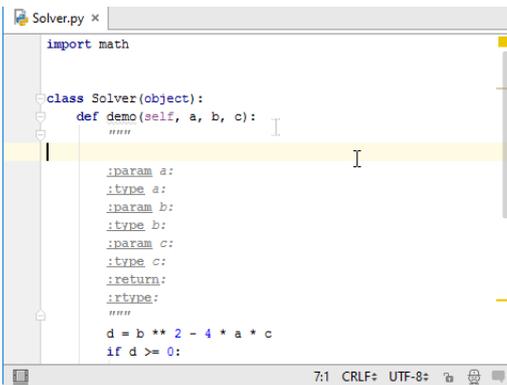


Adding documentation comments

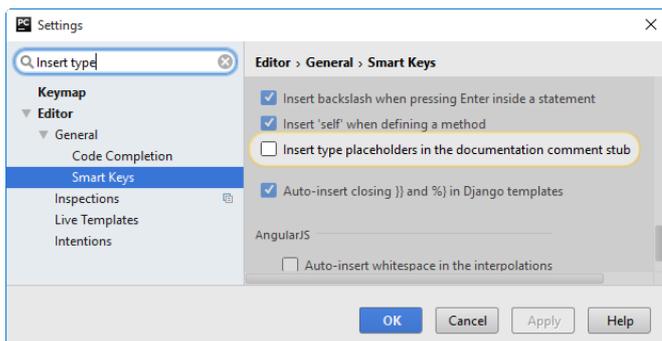
OK, formatting is fixed now, but there are still some stripes left. The inevitable yellow light bulb shows the possibility to add a docstring comment:



Choose this suggestion and see the docstring comment for a certain parameter added:



Note that you have to select the check box Insert type placeholders in documentation comment strings in the [Smart Keys](#) page of the Editor settings:

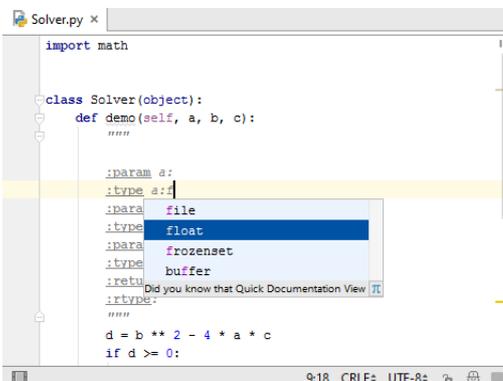


There are several docstring formats, and the documentation comments are created in the format, which you have selected in the [Python Integrated Tools](#) page.

If you so wish, you can change the docstring format to, say, Epytext or plain text.

Type hinting

The documentation comments can be used to define the expected types of parameters, return values, or local variables. Why do we need it all? For example, we'd like to keep under control the types of parameters we pass to the `demo()` method. To do that, let's add the corresponding information to the documentation comment (By the way, mind code completion in the documentation comments!):



Next, when you look at the method invocation, you see that the wrong parameter is highlighted by the PyCharm's inspection Type Checker:

```
Solver.py x
:rtype:
float
d = b ** 2 - 4 * a * c
if d >= 0:
    disc = math.sqrt(d)
    root1 = (- b + disc) / (2 * a)
    root2 = (- b - disc) / (2 * a)
    print(root1, root2)
    return root1, root2
else:
    raise Exception

Solver().demo(2, '173', 0.025)
Expected type 'float', got 'str' instead more... (Ctrl+F1)
```

Expected type 'float', got 'str' instead 28:21 CRLF UTF-8+

Learn more about [type hinting](#) in the PyCharm documentation.

Code Running Assistance

In this section:

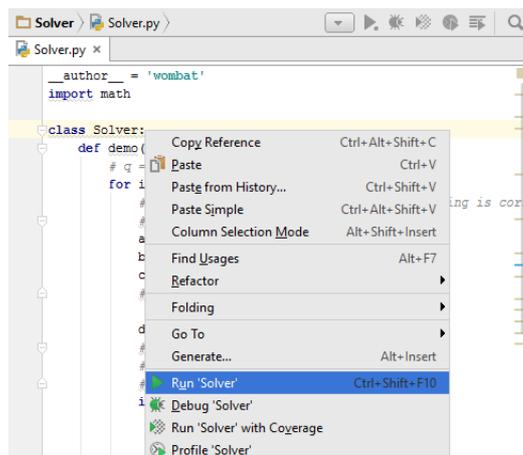
- [Prerequisites](#)
- [First run](#)
- [Run/debug configuration - what is it?](#)
 - [Save run/debug configuration](#)
 - [Edit run/debug configuration](#)
- [Regular run](#)
- [Test run](#)
 - [Selecting a test runner](#)
 - [Creating a test](#)
 - [Running a test](#)
- [Debug run](#)
 - [Breakpoint - what is it?](#)
 - [Setting breakpoints](#)
 - [Debugging session](#)
 - [Working in the Debugger tab](#)
 - [Working in the Console tab](#)
- [REPL - Running in an interactive console](#)

Prerequisites

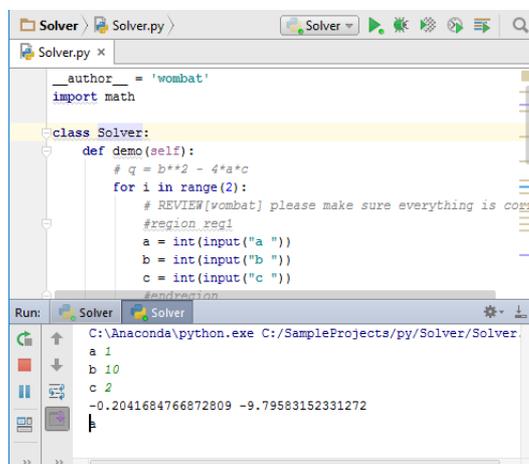
- You are working with PyCharm version 5.0 or higher.
- You have already created a Python project and populated it. You can use a project created in the tutorial [Creating and Running Your First Python Project](#).
- You have Python interpreter already [configured](#). Note that for the current project your Python interpreter version should be 3.0 or higher.

First run

Off we go. Open the class `Solver.py` for editing (), and right-click the editor background. Then choose Run 'Solver' on the context menu:



The script runs and shows its output in the [Run Tool Window](#) :



Let's explore in detail what we've done and what we see.

Run/debug configuration - what is it?

Each script or test you wish to run or debug from within PyCharm, needs a special profile that specifies the script name, working directory, and other important data required for running or debugging. PyCharm comes with a number of such pre-defined profiles, or [run/debug configurations](#), that serve patterns, against

which you can create any number of run/debug configurations of your own.

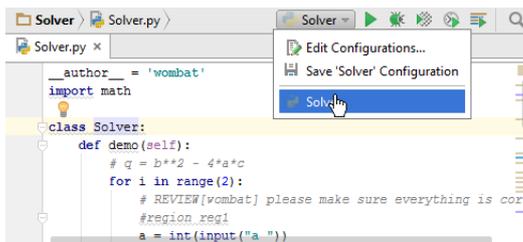
Every time you click the Run or Debug buttons (or choose Run or Debug commands on the context menu), you actually launch the current run/debug configuration in the run or debug mode.

If you look at the very first image, you will notice that in the combobox there is no run/debug configuration at all; on the second image it appears, marked with the green circle. It means that the Solver run/debug configuration has been created automatically by PyCharm, when you've chosen Run 'Solver' on the context menu. Now, as this run/debug configuration is marked with the green circle, it is current.

Look at the main toolbar on the second image: the current run/debug configuration is visible in the combobox. To its right you see the buttons ; the run/debug configuration in the combobox is Solver.

You also see that its icon is shown semi-transparent. What does it mean? It means that the Solver run/debug configuration is **temporary** - PyCharm has created it automatically.

OK, now click the down arrow to reveal the available commands and, below the separator line, the list of existing run/debug configurations:



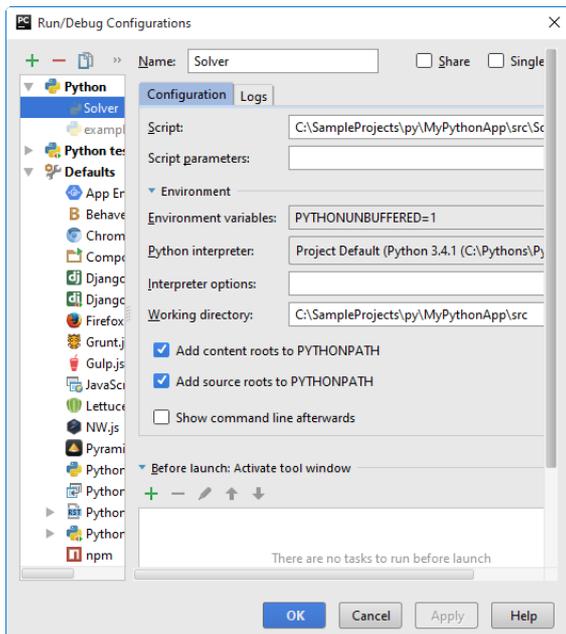
Should you have more run/debug configurations, the list of existing ones will become broader. If you click one of the run/debug configurations in this list, it will become current.

Save run/debug configuration

Choose this command to save the temporary run/debug configuration 'Solver' - now this configuration becomes **permanent**. As such, it gets the normal icon.

Edit run/debug configuration

This command is first in the list. Choose Edit configurations, and see the [Run/Debug Configurations dialog box](#) opens:



Here you see two nodes: Python and Defaults. Under the first node, there is a single `Solver` configuration, under the second node you see a whole lot of run/debug configurations.

What does it mean?

Under the Defaults node, you see only the stubs, or patterns. If you [create a new run/debug configuration](#), it is created on the grounds of the selected pattern. If you change anything under the Defaults node, then the corresponding run/debug configuration you create will be different.

For example, you want to change the Python interpreter to use a remote or other local interpreter. OK, select the interpreter of your choice in the Python page - then any newly created run/debug configuration of the Python type will use this interpreter.

Under the Python node, you see the only run/debug configuration `Solver`. It belongs to the Python type, and is created against the pattern Python. It is denoted with the icon of the normal opacity - which corresponds to the permanent run/debug configuration (remember, it became permanent because you've saved it - however, any specially created run/debug configuration also becomes permanent). As an example, create a new run/debug configuration of the Python type for the same Solver script, and call it 'Solver1'.

If you change anything in one of the existing run/debug configurations, then only this particular run/debug configuration will be affected.

Regular run

You've already executed the Solver script in one of the most [straight-forward ways](#). Let us now explore the other ways to run a script.

As you have already learnt, running a script means in fact launching the current run/debug configuration. So, to run the Solver script, follow this procedure:

1. On the main toolbar, click the run/debug configuration combobox, and make sure that 'Solver' run/debug configuration is the current one.
2. Do one of the following:
 - Click the Run button , located next to the run/debug configuration combobox.
 - Press `Shift+F10`.
 - On the main menu, choose Run | Run.

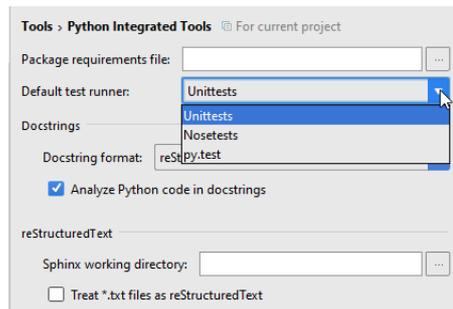
Now you can observe results in the [Run tool window](#).

Test run

We won't discuss here why testing is necessary - let's just assume that it is so, and discuss how PyCharm can help with it.

Selecting a test runner

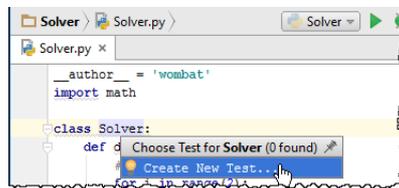
First, choose the test runner. To do that, click  on the main toolbar to open the Settings/Preferences dialog, and under the Tools node, click [Python Integrated Tools](#) page. Here is where you will select the default test runner:



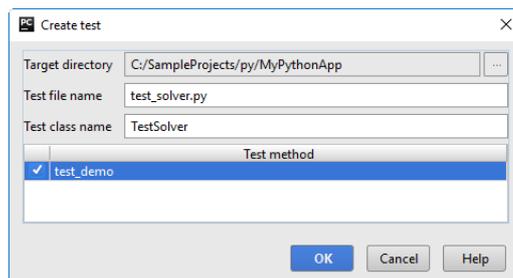
In this case, this is Unittests. Apply changes and close the dialog.

Creating a test

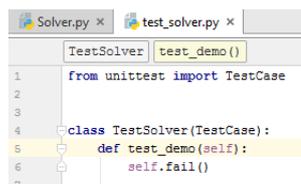
To run a test, you have to create it first. PyCharm suggests a smart way to stub a test out: click the class name and then press `Ctrl+Shift+T`, or on the main menu, choose Navigate | Test. If a test exists, you can jump directly to it; if it doesn't, PyCharm will create it:



Click the suggested action, and PyCharm will show the following dialog:



Click OK, and see the test class opened in the editor:



Great! PyCharm has produced a test class for us. However, this is but a stub, and lacks the actual testing functionality. So, we'll import the class to be tested, and add a test method. The resulting code might be as follows:

```

import unittest

from Solver import Solver

class MyTestCase(unittest.TestCase):

    def test_negative_discr(self):

        s = Solver()

        self.assertRaises(Exception)

    def test_something(self):

        self.assertEqual(True, False)

if __name__ == '__main__':

    unittest.main()

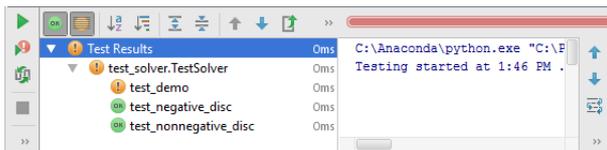
```

TIP if you have a project directory structure, you will need to reflect that in the auto completion of you import statement (`from Solver import Solver`).

Running a test

When ready with the testing code, right-click the test class name - the Run node of the context menu shows Unittests run/debug configuration.

Launch it and observe results in the [Test Runner](#) tab of the Run tool window:



Debug run

First of all, why do we need debugging? Suppose, you hit a run-time error. How to find out its origin? This is where debugging is necessary.

With PyCharm, you can debug your applications without leaving the IDE. The only thing you need to do beforehand, is to place breakpoints in the required places. Let's explore this in details.

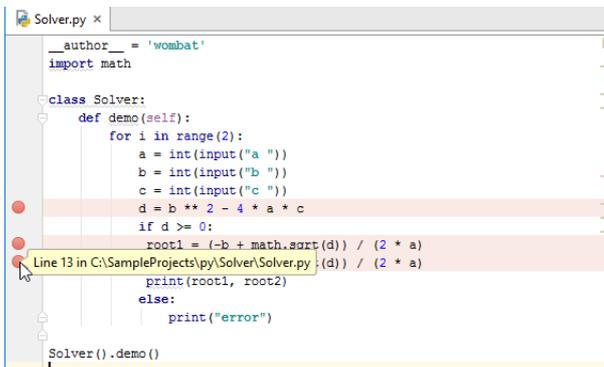
Breakpoint - what is it?

A [breakpoint](#) is a line of the source code, where PyCharm will suspend, when this line is reached. PyCharm discerns several [types of breakpoints](#), each one denoted with its own [icon](#).

Here we'll use the Python line breakpoints.

Setting breakpoints

This is definitely the easiest part of the process. Just click the left gutter on the lines you want to explore - and the breakpoints are there:



Note that each breakpoint is denoted also with a red stripe over the entire line. This color corresponds to a breakpoint that has not yet been reached. Later we'll see how the line at breakpoint changes its color.

By the way, removing breakpoints is same easy - just click the left gutter again.

Hover your mouse pointer over a breakpoint. PyCharm shows a tooltip with the most essential breakpoint information - line number and script address.

However, if you want to change breakpoint settings, you have to right-click a breakpoint. Try changing breakpoint settings for your own, and see how the breakpoint icon changes.

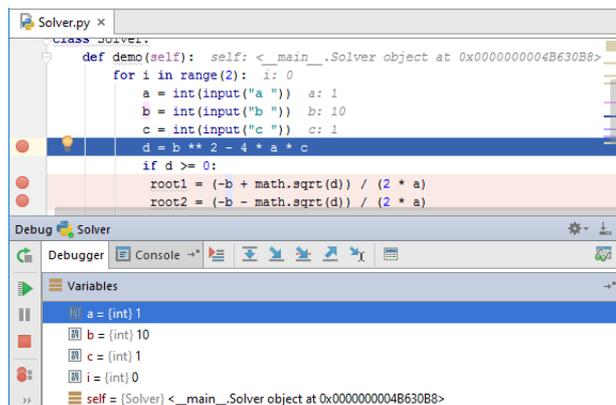
Debugging session

So, we are now ready for debugging. Let's start.

First of all, select the same `Solver` run/debug configuration from the run/debug configurations combobox, and click the Debug icon  to the right.

What happens next?

- PyCharm starts, and then suspends execution at the first breakpoint.
- The line at breakpoint becomes blue. It means that PyCharm has reached the line with the breakpoint, but has not yet executed it.
- Next to the executed lines in the editor, the values of the variables appear.
- The [Debug tool window](#) appears. This tool window shows all the important information related to debugging, and allows managing the debugging process.



Refer to the [product documentation](#) for details.

Working in the Debugger tab

OK, we've paused at the first breakpoint. What's next?

Press `F9` or click . The program will resume and pause at the next breakpoint. This way you can step through all the set breakpoints, observing the variables used in the application. For more information, refer to the part [Debugging](#) of the documentation.

Refer to the section [Debug Tool Window. Debugger](#) for details.

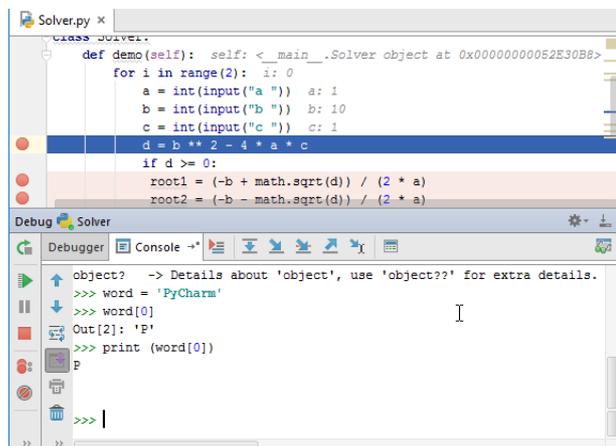
Working in the Console tab

Why do we need it at all? For example, you'd like to see the error messages, or perform some calculations not related to the current application... With PyCharm this is not a problem.

Click the [Console tab](#) to bring it forward, and then, on the toolbar of this tab, click the button :



The Python prompt appears, and the console becomes interactive. Try to execute Python commands in this interactive console:

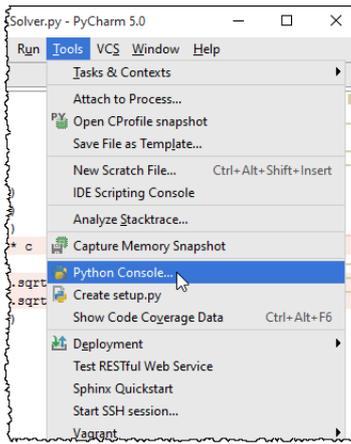


Note that interactive console provides code completion (`Ctrl+Space`) and history (Up/Down arrow keys). Refer to the page [Using Debug Console](#) for more information.

You can always invoke the debug console by using the command `Tools | Open Debug Command Line` on the main menu.

REPL - Running in an interactive console

Finally, if you are used to working with an [Python Console](#), you can also do that right from within PyCharm. To launch the console, choose Tools | Python Console... on the main menu:



Configuring Generic Task Server

On this page:

- [Introduction](#)
- [Prerequisites](#)
- [Specifying the task server](#)
- [Configuring the task server](#)
- [Choosing the response type and using selectors](#)
- [Obtaining issues from the server](#)
- [Summary](#)

Introduction

The Generic connector has been developed as a mechanism enabling the users to configure integration with the servers that PyCharm does not officially support. At the moment, this mechanism:

- Supports the services with REST API.
- Does not support pagination in server responses.
- Allows the following ways of client authentication:
 - [Basic HTTP authentication](#)
 - Sending a preliminary request to the specified address (which is required, for example, by [YouTrack](#)).
- Supports GET and POST requests. At this moment POST requests can contain only [form-encoded data](#), specified as query parameters in the corresponding URL.

Note that a client can only send requests to a server and retrieve responses according to a certain template.

This tutorial is created for the [JIRA 5.0](#) server. However, configuring a generic server described below is valid for any higher version of JIRA, including JIRA OnDemand.

Prerequisites

Make sure that the following prerequisites are met:

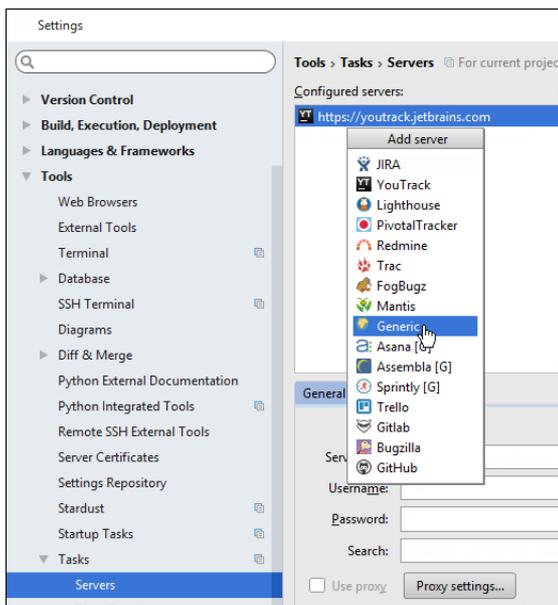
- You are working with PyCharm version 4 or later.
 - If you still do not have PyCharm, download it from [this page](#). To install PyCharm, follow the [instructions](#), depending on your platform.

This tutorial is created with PyCharm version 2016.2.

- You have an account in a tracking system. For example, this tutorial makes use of JIRA.

Specifying the task server

1. Open the [Settings / Preferences Dialog](#) by pressing `Ctrl+Alt+S` or by choosing File | Settings for Windows and Linux or PyCharm | Preferences for macOS. Expand the Tools node, and then click Servers under Tasks.
2. On the [Servers](#) page that opens, click `+` and choose Generic:



3. In the General tab, specify URL of the server, select the check box Use HTTP Authentication, and type your user name and password.
4. Leave the Commit Message tab as is, and then click the Server Configuration tab.

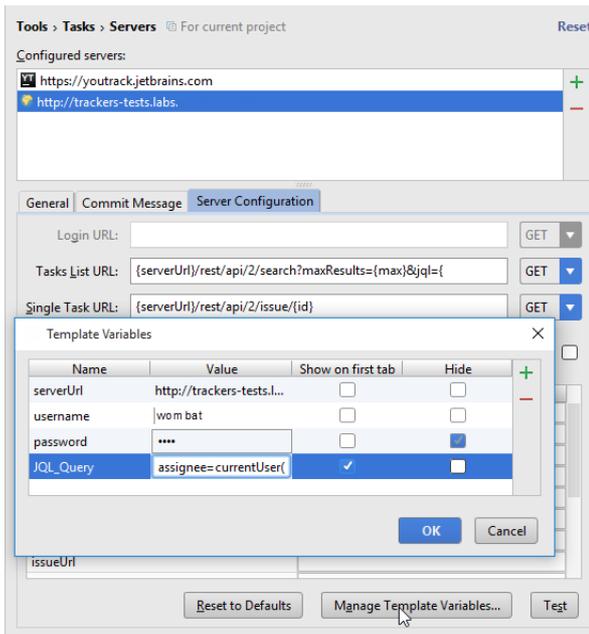
Configuring the task server

The Server Configuration tab contains three fields:

- The Login URL field is disabled, because the check box Use HTTP authentication in the General tab has been selected.
- The Tasks list URL field makes it possible to obtain the list of issues from the server (this list is asked for when you [open a task](#)).

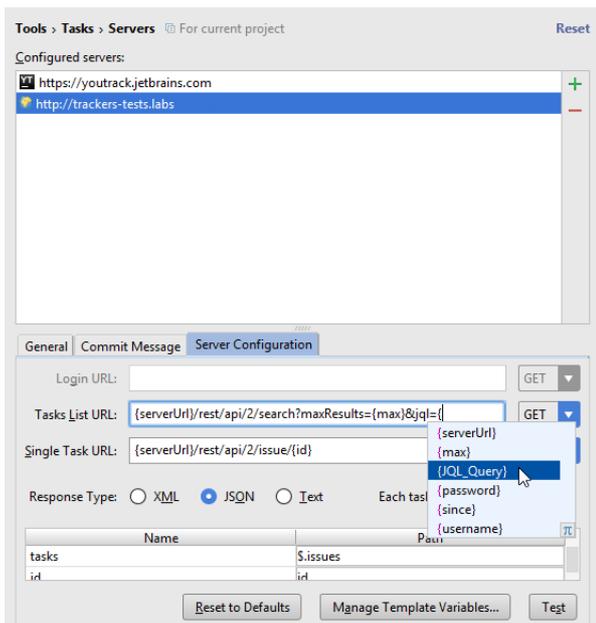
In this field, specify the variable `{serverUrl}` that corresponds to the server URL, specified in the General tab, then type the path `/rest/api/2/search` and expression `?maxResults={max}&jql={JQL_Query}` .

Pay attention to the template variable `{JQL_Query}` , which is not defined so far. To define it, click the button Manage Template Variables, then in the Template Variables dialog box that opens, click `+` and type the name and value of this template variable:



Warning! This is important! For an undefined template variable, the test connection will fail!

Defining a template variable leads to including it in the suggestion list:



Mind the check box Show on first tab in the Template Variables dialog box. If this check box is selected (as it's done in this tutorial), the template variable and its value shows on the General tab of the Servers page.

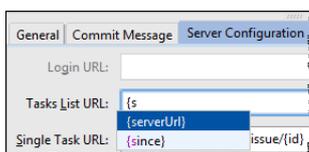
Note It is also possible to use an absolute path in the format `http://<server URL>:<port>/rest/api/2/search<expression>` .

- The Single task URL field makes it possible to obtain a detailed information about a specific issue by its ID. This field is optional, when the check box Each task in separate request is cleared.

To fill in the field, use the following format

<variable `{serverUrl}`> that corresponds to the server URL, specified in the General tab <the path `/rest/api/2/issue/`> <variable `{id}`>

Code completion is available in all these fields.



Choosing the response type and using selectors

You also need to choose the response type and to specify how information about an issue will be extracted from the server response. This is done in the table of selectors in the lower part of the Server Configuration tab.

PyCharm suggests three response types: XML, JSON and text.

- Choose XML if the server responds in the XML format, the selectors are described in XPath.
- Choose JSON if the server responds in the JSON format, the selectors are described in [JSONPath](#).
- Choose Text if the server responds in the plain text format, the selectors are described as regular expressions.

In this tutorial, for example, let's choose JSON and describe the selectors using [JSONPath](#).

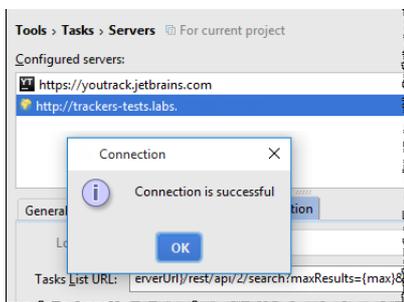
The first three selectors are mandatory:

- tasks: this selector denotes the path in the server response to the descriptions of specific issues.
- id: this selector denotes the path to a unique ID in the description of a specific issue.
- summary: this selector denotes the path to a title of an issue within its description.

To fill out the table, use the server responses. For JIRA, for example, one can observe the server responses immediately in the browser, following a link in the format `http://<server URL>:<port>/rest/api/2/search` .

Obtaining issues from the server

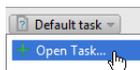
1. To check that the specified settings ensure successful integration with the tracking system, click Test:



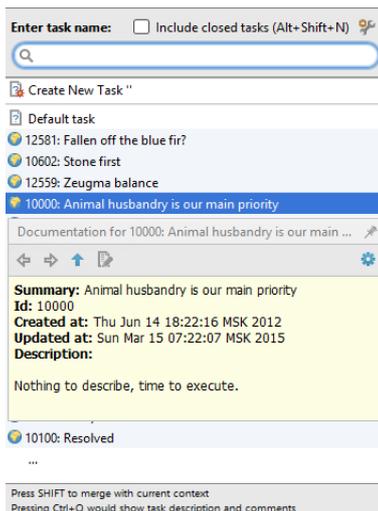
If the connection is successful, click OK and close the Settings dialog.

2. Do one of the following:

- On the main menu, choose Tools | Tasks & Contexts | Open Task
- Press `Shift+Alt+N` .
- Click the tasks combo on the main toolbar:



3. In the list of tasks, choose the one you want to open, and hit `Enter` :



Summary

In this tutorial, we have:

- assigned JIRA as the generic task server.
- configured this task server.
- specified response type and selectors.
- made sure that connection is successful.
- retrieved issues from the configured server.

Configuring PyCharm to Work on the VM

Consider the situation when you work on your project on one platform, but want to deploy and run it on a different one. This is where PyCharm helps a lot with its extensive support for remote interpreters.

The task of running a project on a virtual machine falls into several major steps:

- First, you need to define a virtual box.
- Second, you need to configure a remote interpreter to run in this virtual box.
- Finally, you need to launch your script in the remote console.

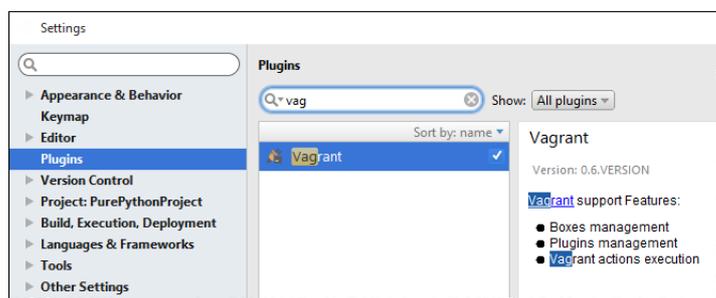
Before you start

Make sure that outside of PyCharm you have already done the following:

- Installed [Oracle's Virtual Box](#).
- Installed [Vagrant](#).
- Added the following executables to your system path:
 - `vagrant.bat` from your Vagrant installation. This should be done automatically by the installer.
 - `VBoxManage.exe` from your Oracle's VirtualBox installation.

In PyCharm, make sure that Vagrant plugin is enabled: on the main toolbar, click , and in the Settings/Preferences dialog box, open the page [Plugins](#).

Actually, this plugin is enabled by default:

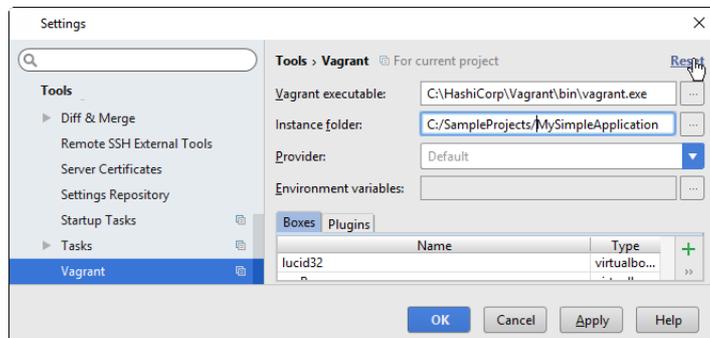


Now all the preliminary steps are over, and we are ready to start.

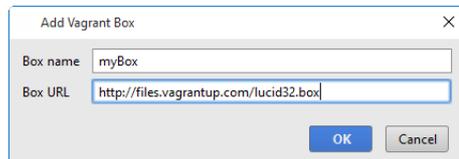
Creating a virtual box

In the Settings/Preferences dialog ( on the main toolbar) click the page [Vagrant](#), and enter the Vagrant executable and Vagrant instance folder.

If the boxes are already defined, they appear in the list, and you can select one.

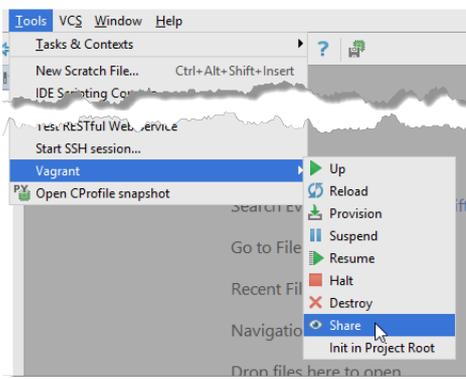


If there is no suitable virtual box, click  to create a new one. Enter the box name and URL for downloading:

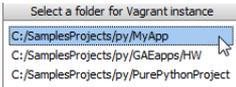


After clicking OK, PyCharm downloads the VM template. Thus, the virtual box is created and added it to the environment.

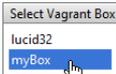
Pay attention to the Vagrant menu commands under the Tools menu. These commands correspond to the standard Vagrant actions. Once a Vagrant box is created, you have to initialize it in project root. To do that, on the main menu, point to Tools | Vagrant, and then choose Init in Project Root:



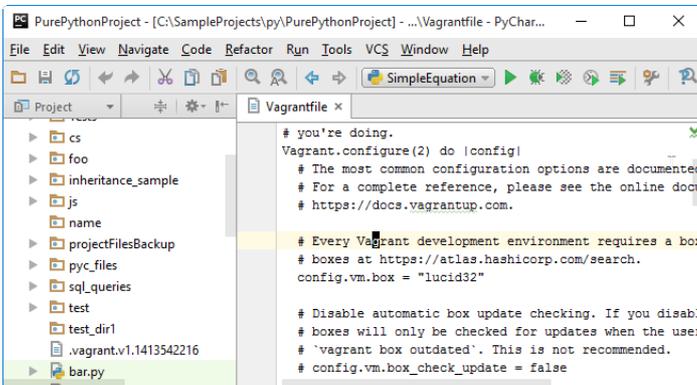
First, choose folder, if required. Note that you have this choice if there are several projects opened in the same window:



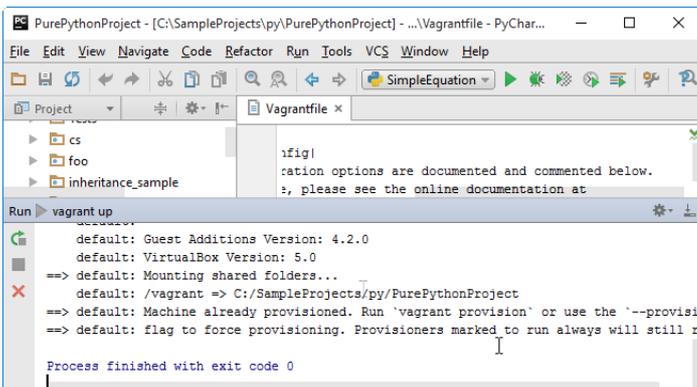
Choose the Vagrant box you are going to initialize:



Thus a `Vagrantfile` is created, and you can view and change it as required:



After initialization, perform the `vagrant up` command (choose Up on the Tools | Vagrant menu). PyCharm runs the `vagrant up` command, and shows its output in the console:

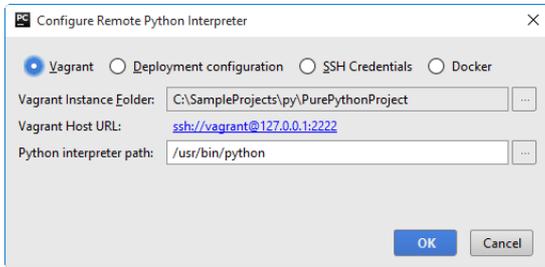


Configuring remote interpreter via virtual box

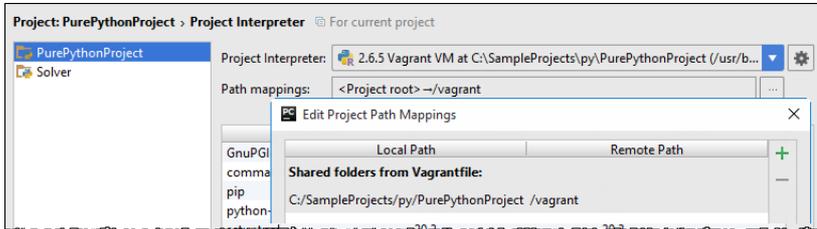
Now open the Settings/Preferences dialog box again (on the main toolbar), and click the page [Project Interpreter](#). Here you can select an interpreter from the drop-down list, but what if none of the suggested interpreters meets your needs? Then click the icon to define your own one. Choose Add Remote on the drop-down menu:



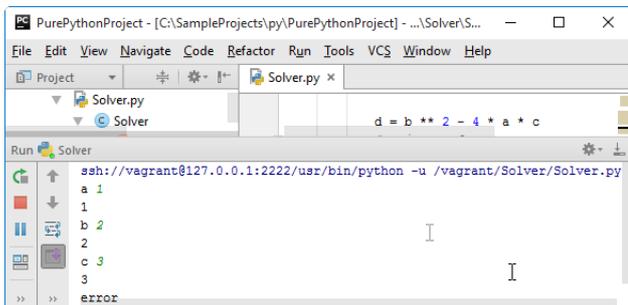
In the Configure Remote Python Interpreter dialog box, you have to specify the server settings. These settings can be taken from the Vagrant configuration file you've already defined. All the server setting fields are filled with the values, taken from the Vagrant configuration file:



Note that the path mappings are defined automatically. However, you can click  to add your own path mappings:



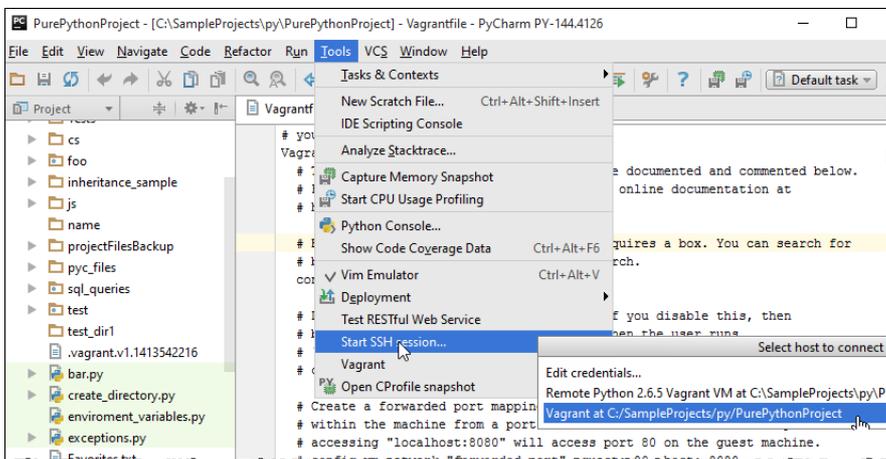
From this point on, you can run any script of your project on the VM:



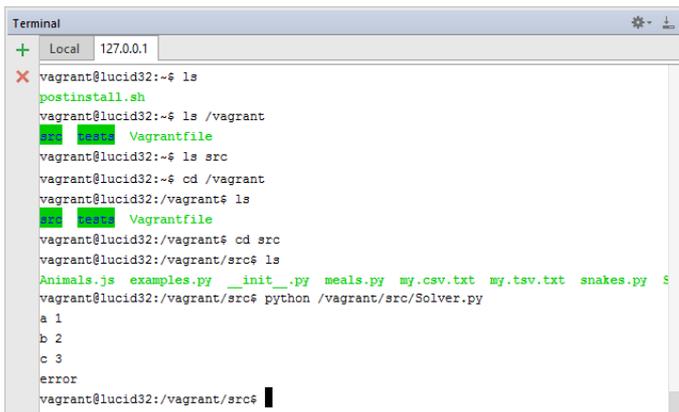
What's next? Let's log in to the virtual box via SSH.

Connecting to the SSH terminal

Why do we need it all? PyCharm lets you log in to your virtual box via SSH and work in its console without leaving the IDE. So, on the main menu choose Tools | Run SSH session.... If you have more than one host already defined, select the one you want to connect to:



Now that you have direct access to your virtual box, let's first make sure your project directory is properly mapped. To do that, just view the contents of the default vagrant's shared folder `/vagrant`, and launch one of the available scripts, for example, `Solver.py`:



Creating and Applying Live Templates (Code Snippets)

In this section:

- [What this tutorial is about ?](#)
- [Prerequisites](#)
- [Creating a live template of your own](#)
 - [1. Creating a stub live template](#)
 - [2. Defining template abbreviation and context](#)
 - [3. Defining template text](#)
 - [4. Editing template variables](#)
 - [Side note about \\$END\\$](#)
 - [5. Memorising the new live template](#)
- [Using a live template of your own](#)

What this tutorial is about ?

PyCharm comes with quite a bit of the various live templates... There are several pre-defined live template for Python. This tutorial aims to walk you through creating a live template for a Python class declaration, and using this live template.

Python programming is out of scope of this tutorial. Refer to the Python documentation for details.

The basics and usage of live templates are also not discussed here. You can find all the necessary information about the types, abbreviations, variables and storage of the live templates in the section [Live Templates](#); to learn how to use the live templates, refer to the section [Creating Code Constructs by Live Templates](#).

Prerequisites

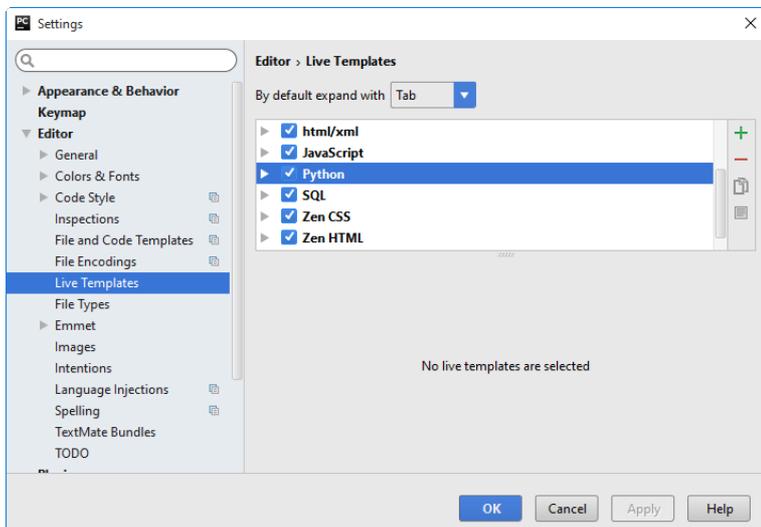
Make sure that you are working with PyCharm version 2.7 or higher. If you still do not have PyCharm, download it from [this page](#) . To install PyCharm, follow the instructions, depending on your platform.

This tutorial has been created using PyCharm Professional version 2016.1.

Creating a live template of your own

1. Creating a stub live template

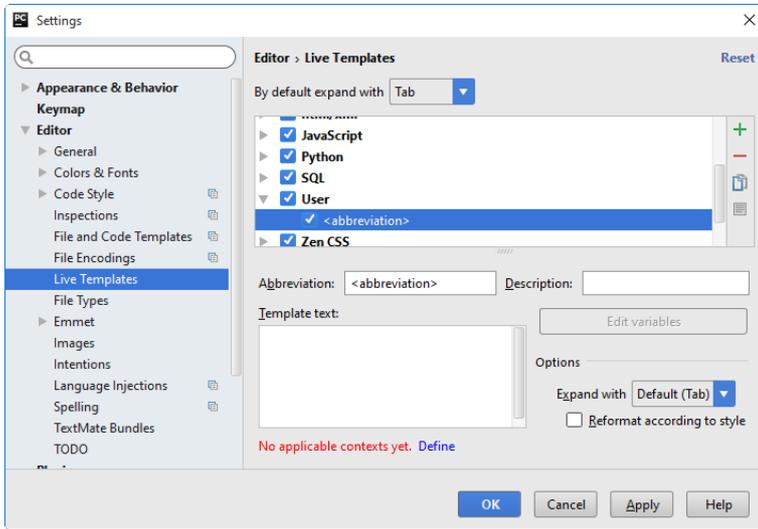
- Open the Settings/Preferences dialog ( on the main toolbar, or `Ctrl+Alt+S`), expand the Editor node, and click [Live Templates](#):



- Click **+**. First, choose Template Group... and specify its name (in our case, it is `user`). The created group gets the focus.
- Click **+** again. This time, choose Live Template. The new template appears in the group that holds the focus, namely `user` .

What happens next?

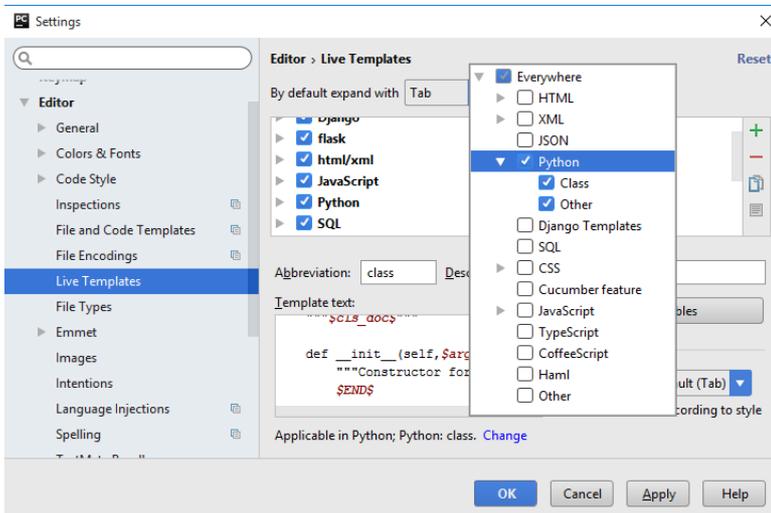
- First, under the group `user` , there is a stub live template that by now is called `<abbreviation>` .
- Second, the fields for entering the template abbreviation, description, body text, and context appear:



2. Defining template abbreviation and context

So, let's enter the [template abbreviation](#). In our example, we'll type the word `class` in the field Abbreviation, and then enter description (which is optional, but nevertheless...).

Next, let's select the context where the new template will apply. By now, you see that the context is not defined - so click the link Define, and select the context (in our case, this is Python). In the future, when the context is already defined, the link changes to Change.



The expansion key will be `Tab`, as specified by default.

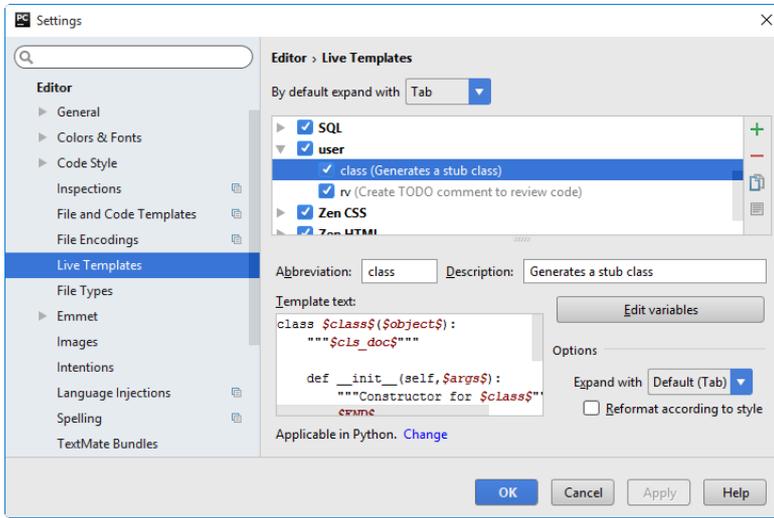
3. Defining template text

Type the following code in the field Template text:

```
class $class$($object$):
    """$cls_doc$"""

    def __init__(self, $args$):
        """Constructor for $class$"""
    $END$
```

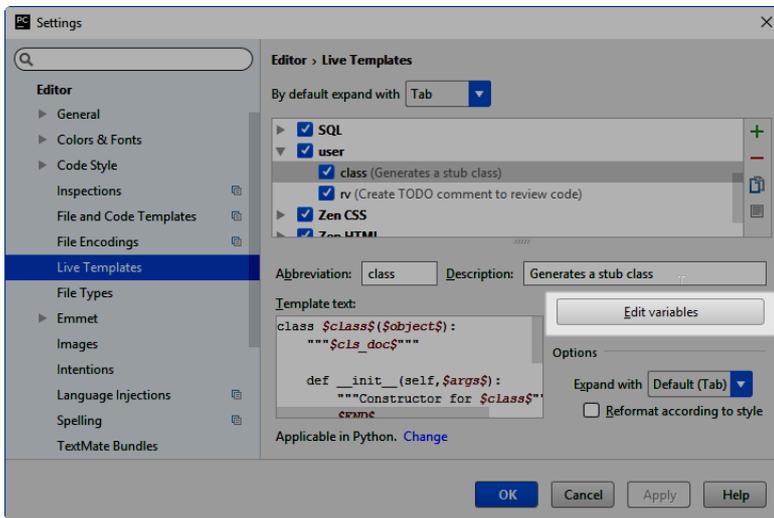
The portions of a template text enclosed in `$` signs are the [template variables](#). You can easily tell them from the entire template text, since they stand out with color:



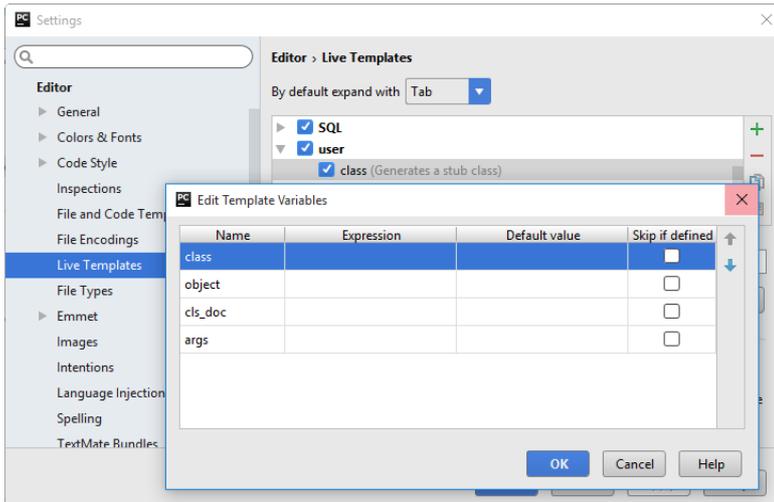
These template variables are void yet, so let's define them first... All, except one (wait a bit to learn why).

4. Editing template variables

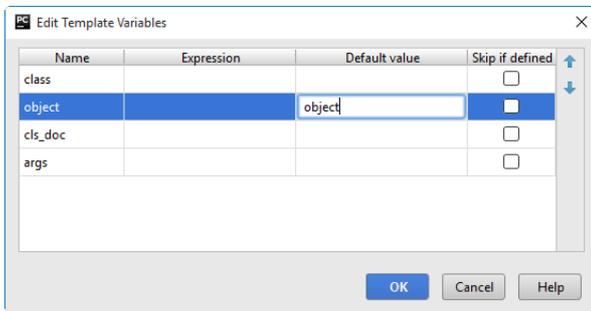
Click the button `Edit variables`:



In the dialog box `Edit template variables`, you see the list of all (but one!) variables:



For the variable `$object$`, let's define its default value (`object`), and click `OK`:



Side note about \$END\$

You have already noticed that the template variable `END` was not suggested for editing. This is because the template variable `END` is predefined, and thus NOT EDITABLE. It always stands for the position of the cursor after the template expansion and filling in all the required fields. So in our case, the cursor will rest at the end of the new class declaration.

5. Memorising the new live template

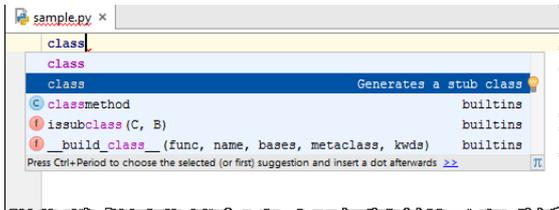
You only have to click OK in the Settings/Preferences dialog. The new live template (or a code snippet) that expands into a stub Python class is ready. Let's just put it to a test..

Using a live template of your own

First, create a Python file (`Alt+Insert` - Python file), and call it `sample.py` .

The new Python file opens for editing. Next, let's create a class declaration in it. To do that, type the template abbreviation `class` .

Wow... our new live template is now in the suggestion list - you can tell it by description that we've typed just in case:

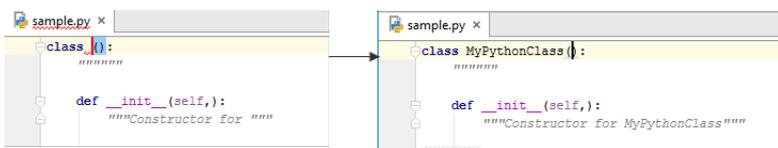


Press `Tab` to choose this option.

As expected, the abbreviation expands into a stub Python class. The red line marks the next entry point: when you type in the class name (variable `class`), it will be entered in this particular location.

Note, by the way, that the variable `class` has been used more than once - in the class declaration, in the comment for the class constructor . In this case , the field for `class` has been filled in automatically.

Having typed the class name, press `Enter` , and see that the red line (frame) moves to the next field. Thus, you have to fill in all the required fields, and press `Enter` at the end.



Debugging JavaScript with PyCharm

In this section:

- [Overview](#)
- [Prerequisites](#)
- [Creating a project](#)
- [Preparing a sample code](#)
- [Setting breakpoints](#)
- [Configuring a server](#)
 - [Creating a server](#)
 - [Configuring connection and mappings](#)
 - [Defining project default server](#)
 - [Viewing the server](#)
- [Deploying file to the server](#)
- [Debugging](#)
 - [Starting the debugger session](#)
 - [Examining the debugger information](#)

Overview

The possibility to debug JavaScript is vital for the web developers. With IntelliJ IDEA-based products, such debugging becomes quite easy. To illustrate the JavaScript debugging capabilities with PyCharm, we'll create a very basic script that just shows some numbers in a browser page, and then debug it on a server.

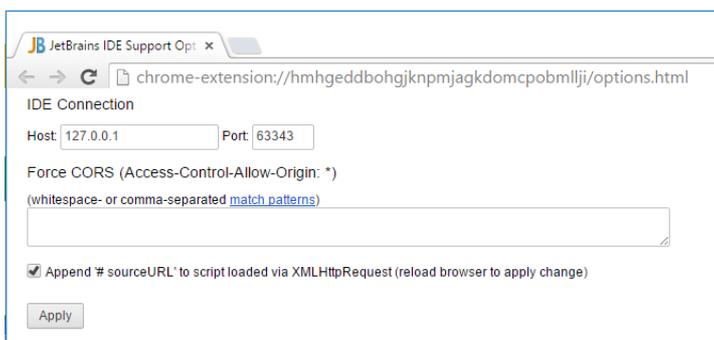
In case of debugging on an external web server, the application files are deployed on it, and you have their copies on your computer. No matter, whether the web server itself is running on a physically remote host or on your machine, application files deployed on it are treated as remote.

When a remote HTML file with a JavaScript injection is opened, the debugger tells PyCharm the name of the currently processed file and the number of the line to be processed. PyCharm opens the local copy of this file and indicates the line with the provided number. This behaviour is enabled by specifying correspondence between files and folders on the server and their local copies. This correspondence is called mapping, it is set in the debug configuration.

Prerequisites

Make sure that:

- You are working with PyCharm Professional Edition version 3.0 or higher.
- You are working with Google Chrome.
- You have [JetBrains IDE Support extension](#) installed and activated. If you launch the debugger for the very first time, PyCharm will warn you about the necessity to install the extension JetBrains IDE Support. For example, with Chrome, you can tell that JetBrains extension is installed and activated, when  icon shows to the right of the address bar, and it is non-transparent.
- The JetBrains Chrome Support Extension has port number 63343 and host 127.0.0.1:



- You are using [XAMPP](#) as an application server.

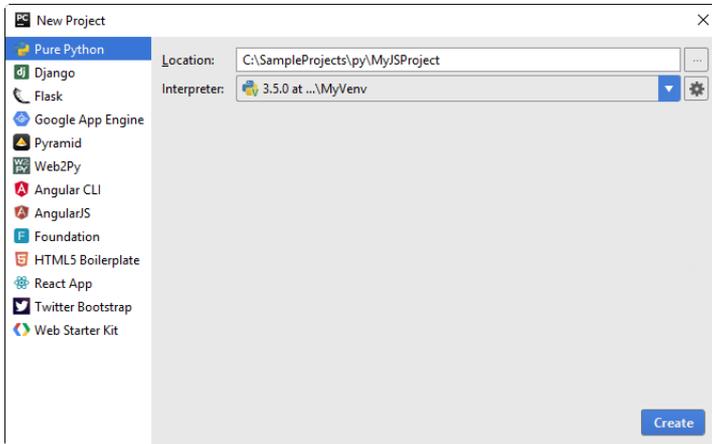
This tutorial is created with the following tools:

- PyCharm Professional 5.0.
- Google Chrome.
- [XAMPP](#).

Creating a project

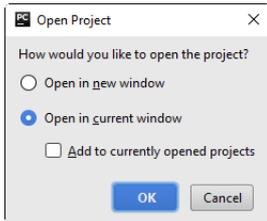
On the main menu, choose File | New Project, and do the following:

- choose the Pure Python project type
- specify the project location `C:\SampleProjects\py\MyJSProject`
- choose the project interpreter



When ready, click Create.

Choose to open this new project in a separate window, not adding it to any of the currently open projects:



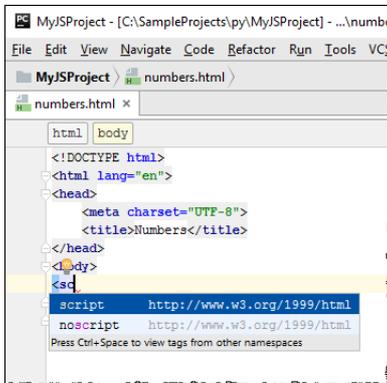
Preparing a sample code

First, let's create a HTML page. To do that, with the [Project Tool Window](#) having the focus, press `Alt+Insert`, choose HTML file on the pop-up menu, and enter the file name `numbers`.

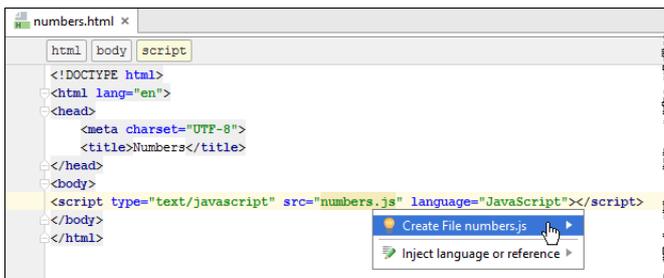
PyCharm stubs out an HTML file with some initial contents. Next, embed a reference to a JavaScript file into this HTML file. To do that, type the following code inside the `<body>` tags:

```
<script type="text/javascript" src="numbers.js" language="JavaScript"></script>
```

Mind the code completion, which is available while you type:



When ready, pay attention to the highlighted file name `numbers.js`. This is a reference to a non-existent JavaScript file. Place the caret at this name and press `Alt+Enter` (or click the yellow light bulb ); a quick fix is suggested – create the missing file:

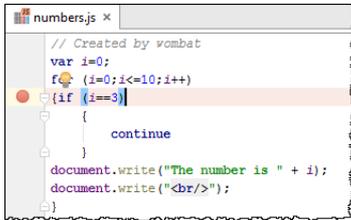


Choose this quick fix, and have the stub JavaScript file ready. Next, enter the following code:

```
var i=0;
for (i=0;i<=10;i++)
{
  if (i==3)
  {
    continue;
  }
  document.write("The number is " + i);
  document.write("<br/>");
}
}
```

Setting breakpoints

Now let's set breakpoints to our JavaScript file. This is most easy – just click the left gutter at the line you want the script to suspend:



Configuring a server

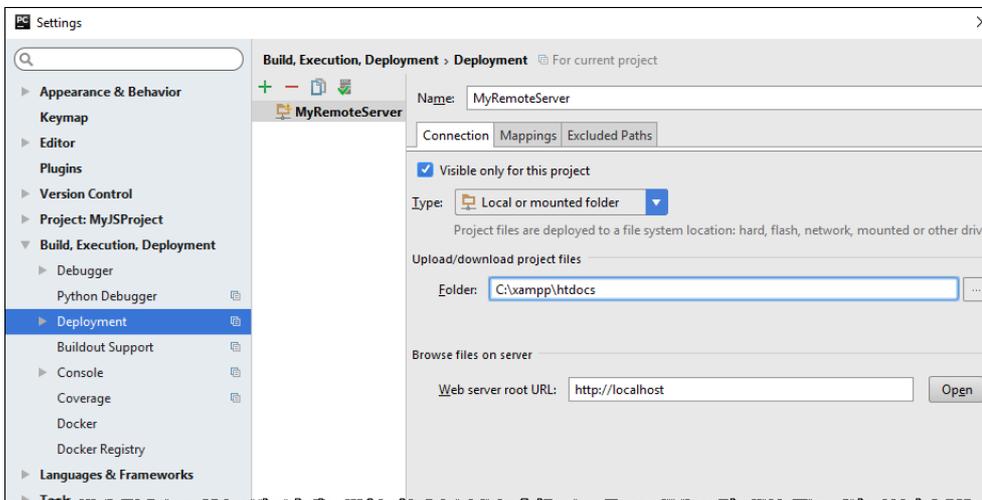
Creating a server

To create a server, follow these steps:

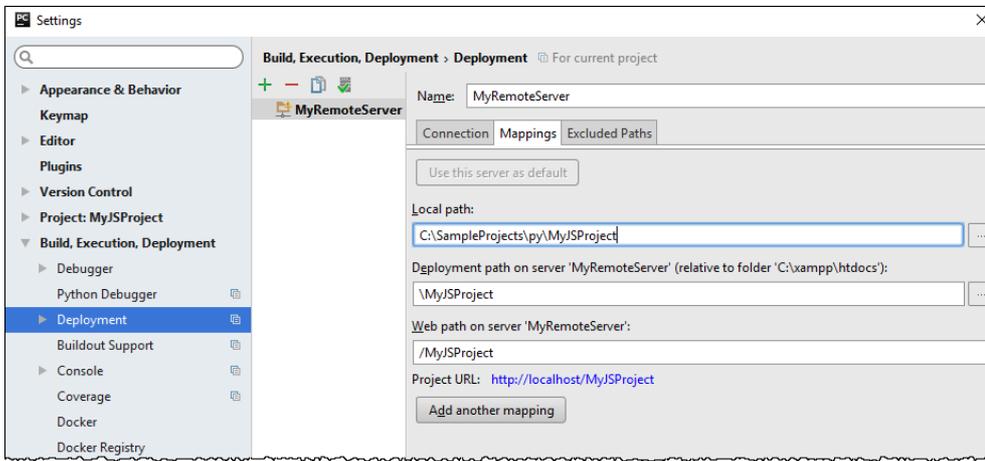
1. Open the Settings/Preferences dialog (press `Ctrl+Alt+S` or click  on the main toolbar)
2. Expand the node Build, Execution, Deployment, and click the page [Deployment](#).
3. On the [Deployment](#) page, click `+`.
4. Specify the server name `MyRemoteServer` and type `local or mounted server`.

Configuring connection and mappings

Then, configure the server. In the Connection tab, specify the directory where your local files will be uploaded; in our case, this directory is `C:\xampp\htdocs` - it means that the local files will be uploaded to this directory:

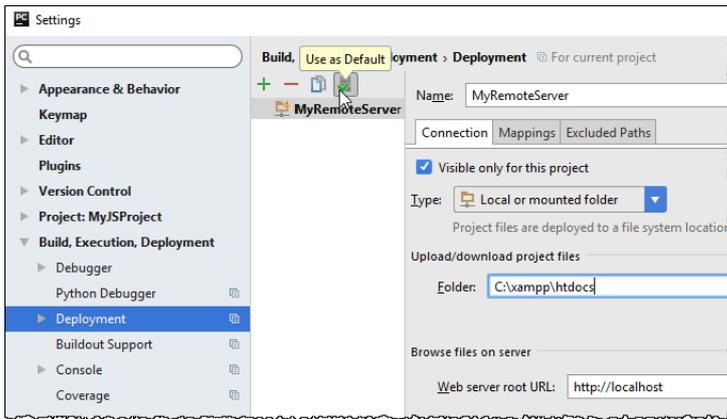


Next, click the Mappings tab. Here define the local path, the deployment path on the server (which is relative to the folder specified in the Connection tab), and the Web path on the server:



Defining project default server

Make the server the project default. To do that, click  button in the [Deployment toolbar](#).

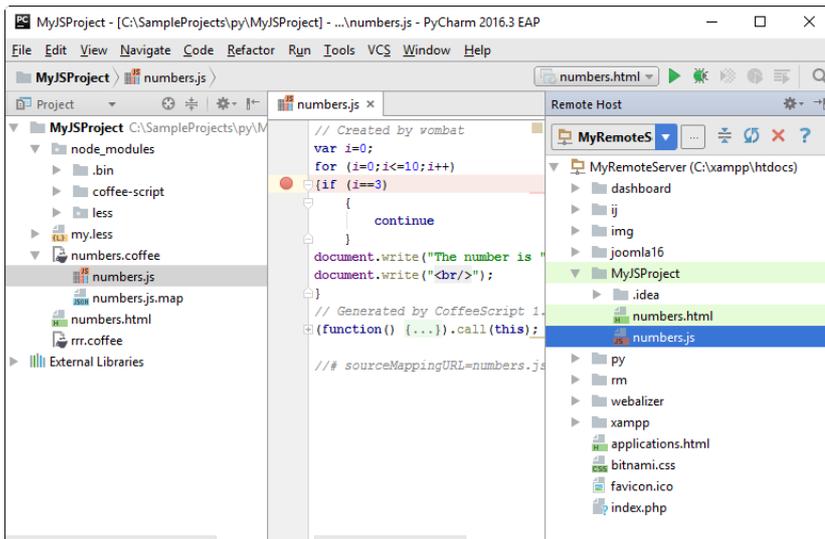


Click OK to apply changes and close the Settings/Preferences dialog.

Finally, copy your project (`C:\SampleProjects\py\MyJSProject`) to `C:\xampp\htdocs` .

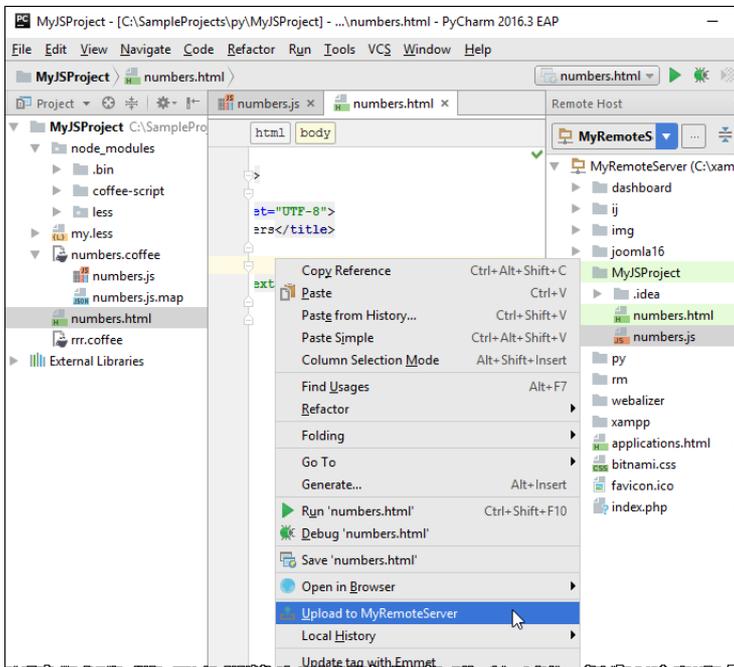
Viewing the server

Let's make sure the server is up and running, and, which is even more important, visible to PyCharm. To do that, on the main menu, choose `Tools | Deployment | Browse Remote Hosts`. The [Remote Hosts tool window](#) shows the newly created server:

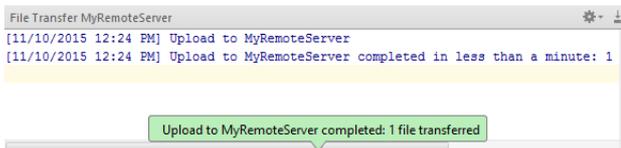


Deploying file to the server

With PyCharm, it's just a snap... For example, you can easily do that via the main menu: choose `Tools | Deployment | Upload to MyRemoteServer`, or use the context menu:



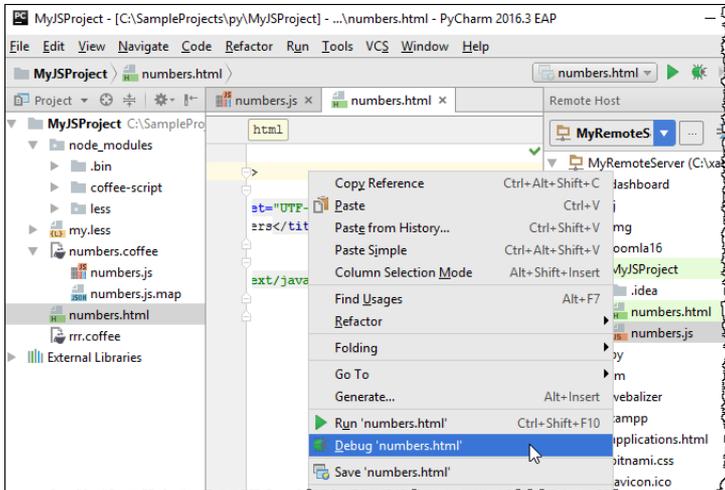
The upload result is shown below:



Debugging

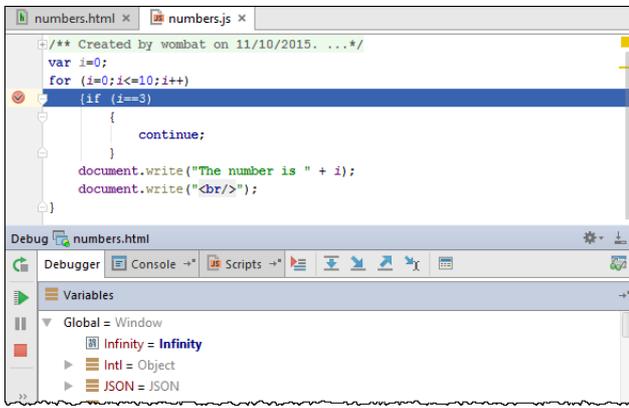
Starting the debugger session

All the preliminary steps are done, and it's time to proceed to the debugging session. To start it, right-click the background of the file `numbers.html`, and choose `Debug 'number.html'` on the context menu - thus you will launch the debugger with the default [temporary run/debug configuration](#):

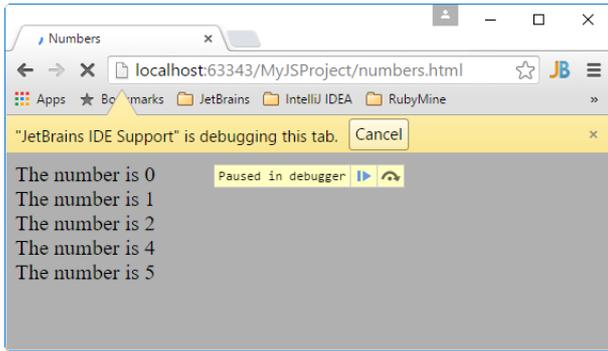


Examining the debugger information

When the debugging session is launched, your HTML page appears in the browser, and the [Debug tool window](#) opens. Your program execution suspends when the first breakpoint is hit. Such a breakpoint is marked with a blue stripe:



As you step through your application, the corresponding information appears in the [Debug tool window](#), and in the page of your web browser:



To step through the script, click or ; to terminate the debugger session, close the yellow banner, or click Cancel.

Debugging Python Code

In this section:

- [What these tutorials are about](#)
- [Before you start](#)
- [Preparing an example](#)
- [Placing breakpoints](#)
- [Starting the debugging session](#)

What these tutorials are about

In this tutorial, we're going to debug the Python code. With the example code provided below you can try all of the features mentioned in these tutorials.

Learning all the [debugging features and capabilities](#) is out of scope. With this tutorial you'll learn the most important ways to debug your code by example.

Before you start

Make sure that:

- You are working with PyCharm. If you still do not have PyCharm, download it from [this page](#). To install PyCharm, follow the [instructions](#), depending on your platform.
This tutorial has been created with PyCharm version 2017.1.
- You have [created a project](#).

Preparing an example

Copy the following code into a file in your project (though it is recommended to type this code manually):

```
import math

class Solver:

    def demo(self, a, b, c):
        d = b ** 2 - 4 * a * c
        if d > 0:
            disc = math.sqrt(d)
            root1 = (-b + disc) / (2 * a)
            root2 = (-b - disc) / (2 * a)
            return root1, root2
        elif d == 0:
            return -b / (2 * a)
        else:
            return "This equation has no roots"

if __name__ == '__main__':
    solver = Solver()

    while True:
        a = int(input("a: "))
        b = int(input("b: "))
        c = int(input("c: "))
        result = solver.demo(a, b, c)
        print(result)
```

Placing breakpoints

To place breakpoints, just click the left gutter next to the line you want your application to suspend at.

Refer to section [Breakpoints](#) for details.

//todo add image

Starting the debugging session

OK now, as we've added breakpoints, everything is ready for debugging.

Deployment in PyCharm

On this page:

- [What this tutorial is about](#)
- [Before you start](#)
- [Preparing an example](#)
- [Configuring a deployment server](#)
 - [What is specified in the Connection tab?](#)
 - [What is specified in the Mapping tab?](#)
- [Browsing remote hosts](#)
- [Deployment tools](#)
 - [Uploading](#)
 - [Comparing remote and local versions](#)
 - [Downloading](#)
- [Synchronizing changes](#)
- [Automatic upload to the default server](#)
 - [Defining a server as default](#)
 - [Enabling automatic upload](#)
- [Uploading external changes](#)
- [Summary](#)

What this tutorial is about

This tutorial aims to take you step-by-step through configuring an managing deployment of your code to remote hosts, using PyCharm.

Before you start

Make sure that:

- You are working with PyCharm version 5.0 or higher. This tutorial is prepared with PyCharm 2016.1.
- You have access right to a remote host you want your code to be deployed on.

Also note that this tutorial is created on Windows 10 and makes use of the default keyboard shortcuts scheme. If you are working on a different platform, or use another keyboard scheme, the keyboard shortcuts will be different.

Preparing an example

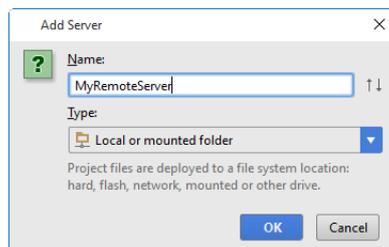
You can use the [everlasting sample project](#).

Thus the preliminary steps are done, and we are ready to take off.

Configuring a deployment server

On the main toolbar, click  to open the Settings/Preferences dialog, and choose the page [Deployment](#) (actually, you can access the same page by choosing Tools | Deployment | Configuration on the main menu).

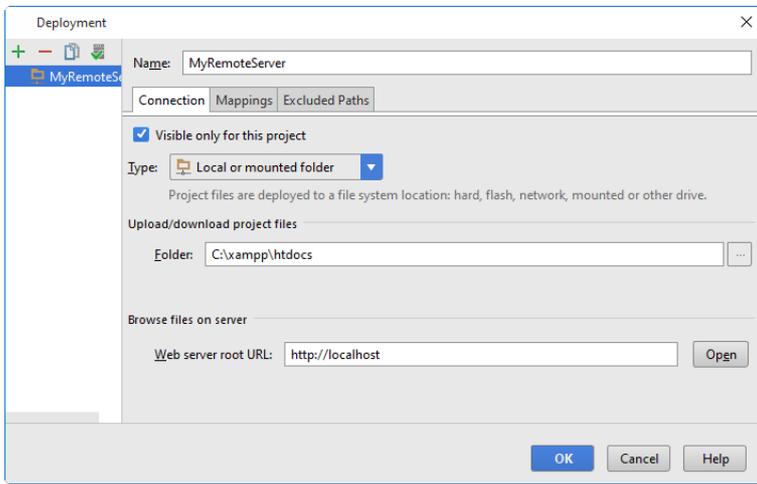
Click , then in the Add Server dialog box, type your server name (`MyRemoteServer`) and select its type (in our case, this is Local or mounted folder):



OK, the new server is added, but it is still void... It only shows the Web server root URL `http://localhost`, where you will actually browse your uploaded files.

What is specified in the Connection tab?

Select the directory where the project files will be uploaded. In our case, this is the local folder `C:\xampp\htdocs` (You can either type this path manually, or press `Shift+Enter` to open the [Select Path dialog](#).)

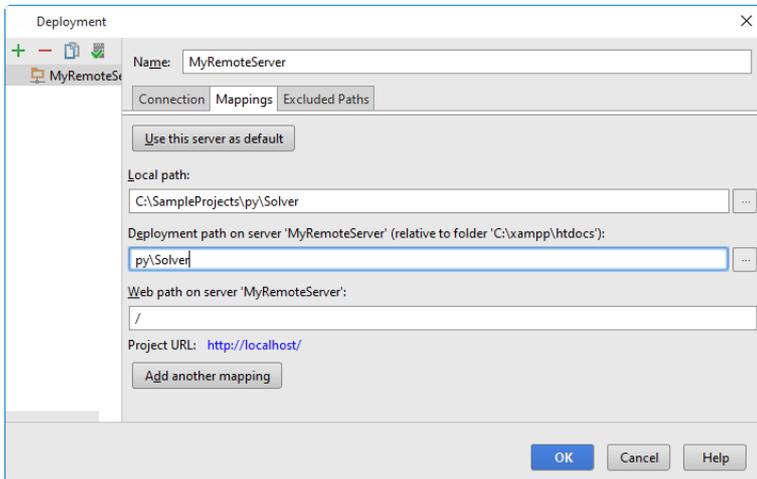


What is specified in the Mapping tab?

Next, choose the **Mappings** tab. By default, the Local path field contains the project root. However, you can select any other directory within your project tree. Let's assume the default value.

In the Deployment path field (which is by default empty), you have to specify the folder on your server, where PyCharm will upload data from the folder, specified in the Local path(in this example, it's py\Solver). This path is specified relative to the folder `C:\xampp\htdocs` !

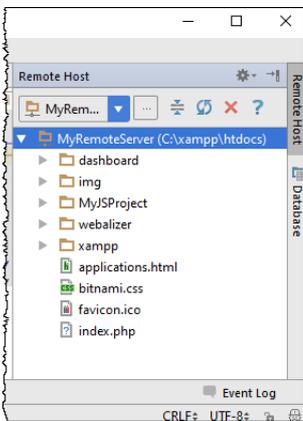
And, finally, let's accept the default value for Web path on server 'MyRemoteServer':



OK, apply changes, and the server is ready to use.

Browsing remote hosts

You can easily make sure your server is up and running. Just choose the command **Tools | Deployment | Browse Remote Hosts** on the main menu, and the **Remote Hosts tool window** appears at the right edge of the PyCharm's frame:



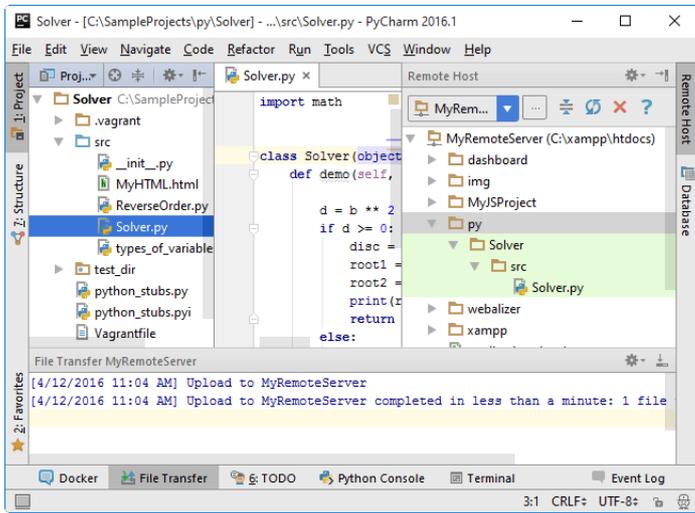
Deployment tools

Next, let's perform some actions, and see what happens.

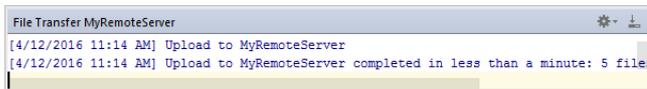
Uploading

First, let's upload one of the files to the remote server. This how it's done...

In the **Project tool window**, right-click a file you want to upload. In our case, let it be the file `so1ver.py` . On the context menu, choose **Upload to MyRemoteServer**, and see the upload results!



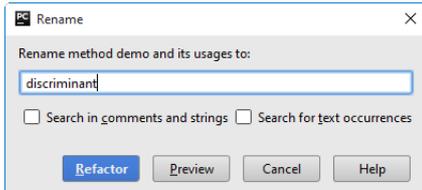
You can also upload contents of each directory within your project. For example, right-click the parent directory of the `Solver.py`, namely, `src`, choose Upload to MyRemoteServer on the context menu. Wow! We have the entire directory uploaded to the server:



Comparing remote and local versions

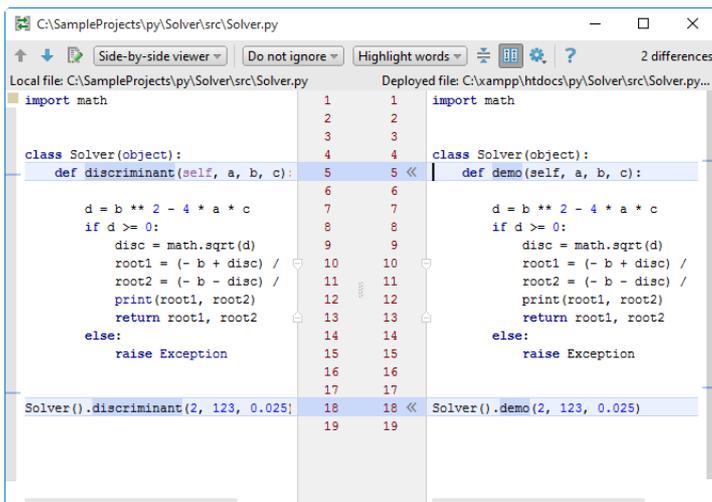
There is a local and a remote copy of the file `Solver.py`, and they are identical. Let's change the local version.

To do that, place the caret at the method declaration, and press `Ctrl+Shift+Alt+T` (or choose Refactor | Refactor This on the main menu). The popup menu shows all refactorings, available in the current context. Let's choose Rename refactoring, and [rename a method](#):



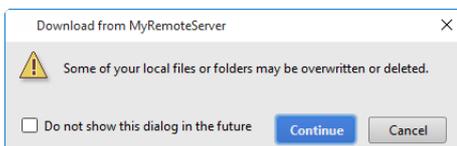
Perform refactoring and see the method name and its usage changed.

OK, now we've changed the local version. Let's make sure PyCharm knows about these changes. To do that, again go to the [Remote Host Tool Window](#) tool window, and right-click `Solver.py`. On the context menu, choose Compare with Local Version. PyCharm opens the differences viewer, where you can accept changes or reject them, using the buttons `», «, ↵, ⌫, X`:

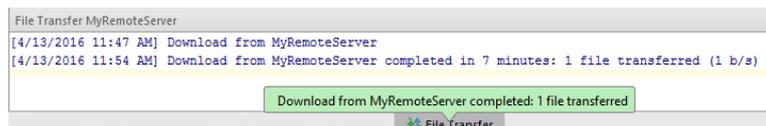


Downloading

In the [Remote Host Tool Window](#) tool window, right-click the file `Solver.py`, and choose Tools | Deployment | Download from here on the main menu. PyCharm immediately shows a warning:



Do not be afraid, and click Continue:

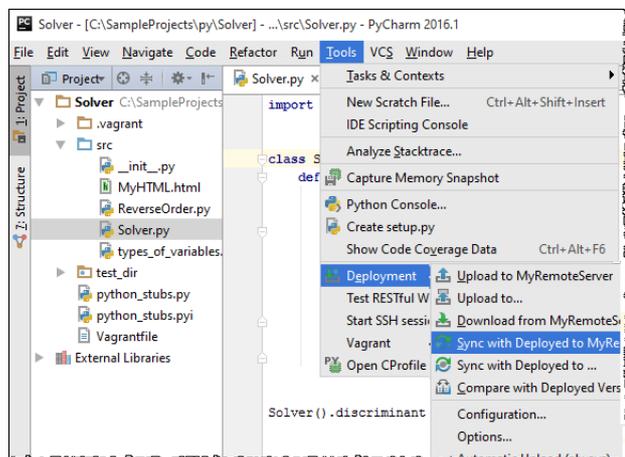


You can also download an entire directory, if it has been previously uploaded to the server. For example, if you click the parent directory `src` and choose the same command, all nested files will be downloaded from the server.

Synchronizing changes

Make a preliminary step - roll the changes to the `Solver.py` file back (`Ctrl+Z`). You again see the class `Solver.py` with the renamed method.

Next, click `Solver.py`, and on the main menu choose `Deployment | Sync with Deployed to MyRemoteServer`:



PyCharm shows differences viewer, where you can accept individual changes or reject them, using the buttons `»`, `«`, `↵`, `⌫`, `X`.

Automatic upload to the default server

When a user needs to have the exact same files on the server as in a PyCharm project, automatic upload can be of help. Automatic upload means that whenever a change is saved in the IDE, it will be deployed to the default deployment server.

Defining a server as default

A deployment server is considered default, if its settings apply by default during automatic upload of changed files. To define a deployment server as the default one, follow these steps:

1. Choose the desired server in the [Deployment page](#) (in our case, `MyRemoteServer`). You can open this page in two possible ways: either `Settings/Preferences | Build, Execution, Deployment | Deployment`, or `Tools | Deployment | Configuration` on the main menu.
2. Click .

Enabling automatic upload

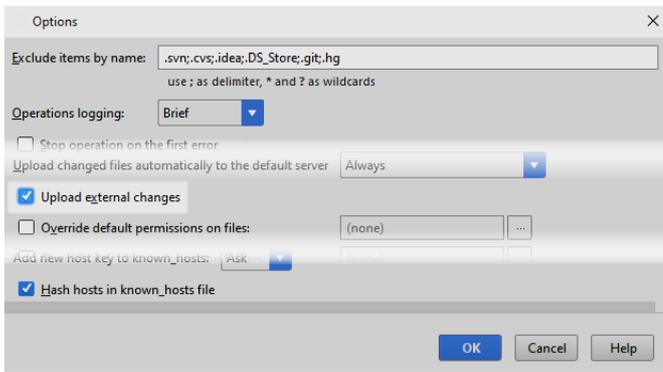
As soon as the default server is set, you can make upload to this server automatic. This can be done in the following two ways:

- First, open the deployment [Options](#) (`Settings/Preferences | Deployment | Options` or `Tools | Deployment | Options` on the main menu), and in the field `Upload files automatically to the default server` choose `Always`, or `On explicit save action`. The difference between these two choices is explained in the [field description](#).
- Second, on the main menu, select the check command `Tools | Deployment | Automatic upload`. Note that automatic upload in this case is performed in the `Always` mode.

It is worth mentioning that the option `Always` is not recommended for deployment to production: incomplete code can be uploaded while developing, potentially breaking the production application.

Uploading external changes

By default, PyCharm uploads only the files changed by itself. If the files are changed by some other process, such as a [VCS branch change](#), [compilation of SASS or LESS](#) or a [File Watcher](#), they are not automatically uploaded. To change this behavior and autoupload these changes as well, enable the `Upload external changes` option:



Summary

Congrats! You've passed this very basic tutorial. What you've done?

- Created and configured a server of your own.
- Uploaded and downloaded files and folders.
- Compared local and remote versions.
- Configured the server as default.
- Enabled automatic upload of external changes.

Exploring Navigation and Search

In this section:

- Exploring Navigation and Search
 - [What these tutorials are about](#)
 - [Before you start](#)
 - [Preparing an example](#)
 - [What's next?](#)
- [Part 1. Finding by Name, Jumping to Recent and Searching Everywhere](#)
- [Part 2. Navigating to a Declaration, Implementation and Test](#)
- [Part 3. Finding Usages](#)
- [Part 4. Using the Navigation Bar](#)
- [Part 5. Navigating to the UI Elements](#)
- [Part 6. Django-Specific Navigation](#)

What these tutorials are about

In this series of tutorials, we're going to navigate around your code in the most efficient way. With the example code provided below you can try all of the features mentioned in these tutorials.

The tutorials are located in the ascending order: the first one describes the most basic navigation facilities of PyCharm, while the last ones relate to alternative ways of navigation.

The parts 1-5 use the same example code. The part 6 relates to Django and thus makes use of the code example from [Creating and Running Your First Django Project](#)

Learning all the [navigation features and capabilities](#) is out of scope. With these tutorials you'll learn the most important ways to navigate around your code by example.

Before you start

Make sure that:

- You are working with PyCharm. If you still do not have PyCharm, download it from [this page](#). To install PyCharm, follow the [instructions](#), depending on your platform.
This tutorial has been created with PyCharm version 2017.1.
- You have [created a project](#).

Preparing an example

Do the following:

1. Add directory `Animals` under your project root (`Alt+Insert` | Directory).
2. Create the following Python files (`Alt+Insert` | Python File):
 - `Mammalia.py`
 - `Carnivorae.py`
 - `Herbivorae.py`
 - `Dog.py`
3. Open these files for editing (`F4`) and add the following code:
 - `Mammalia.py` :

```
from Animals.Carnivorae import Carnivorae
from Animals.Herbivorae import Herbivorae

class Mammalia(object):
    extremities = 4
    def feeds(self):
        print("milk")
    def proliferates(self):
        pass
class Marsupialia(Mammalia):
    def proliferates(self):
        print("poach")
class Placentalia(Mammalia):
    def proliferates(self):
        print("placenta")

class TasmanianDevil(Marsupialia, Carnivorae):
    pass

class Duckbill(Marsupialia, Herbivorae):
    pass

class Cat(Carnivorae, Placentalia):
    pass

class Tiger(Placentalia,Carnivorae):
    pass

class Cow(Placentalia, Herbivorae):
    pass

Cat.feeds()
```

– Carnivorae.py :

```
from Animals.Mammalia import Mammalia

class Carnivorae(Mammalia):
    def food(self):
        print("meat")
    pass
```

– Herbivorae.py :

```
from Animals.Mammalia import Mammalia

class Herbivorae (Mammalia):
    def food(self):
        print("grass")
    pass
```

What's next?

Let's begin with [Part 1](#) and explore the most basic and often-used means of navigation.

Part 1. Finding by Name, Jumping to Recent and Searching Everywhere

In this section:

- [Navigate to file/class/symbol](#)
- [Navigate to file](#)
- [Navigate to class](#)
- [Navigate to symbol](#)
- [Recent files](#)
- [Searching everywhere](#)
- [Jumping to a line](#)
- [Summary](#)
- [What's next?](#)

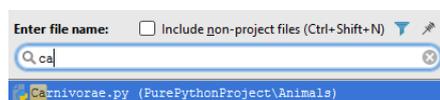
Note To be able to try the functionality described in this tutorial and follow it step by step, you'll need an example source code. It can be found [here](#).

Navigate to file/class/symbol

This is one of the most powerful PyCharm's navigation and search features that enables you to find actually any file, class or symbol by its name, and jump directly to it. This is how it's done.

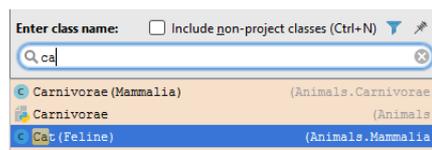
Navigate to file

Press `Ctrl+Shift+N`, and type `ca` in the pop-up window that opens:

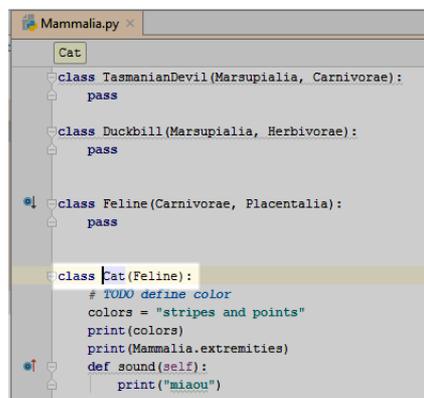


Navigate to class

Next, let's jump to a specific class. To achieve this goal, press `Ctrl+N`, and type, for example, `ca`. The suggestion list shows all classes with the letter `ca` somewhere inside their names. Note that each class is followed by its fully qualified path:

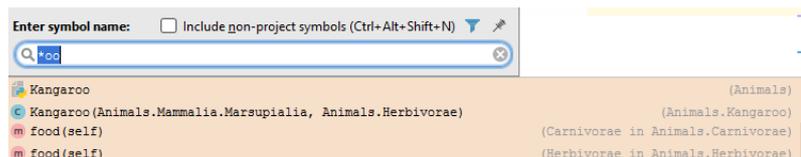


Select `Cat` in the suggestion list, and press `Enter`. This time, the file opens with the caret at the class declaration:

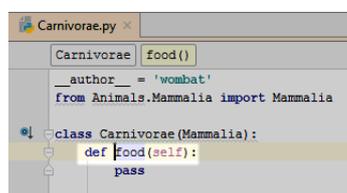


Navigate to symbol

Next, let's jump to a certain member of a class. Press `Ctrl+Shift+Alt+N`, and type `*oo` (`*` is a mask for any amount of any characters.)



From the suggestion list, select `food` from `Carnivorae` and press `Enter`. The file `Carnivorae.py` opens at the method declaration:

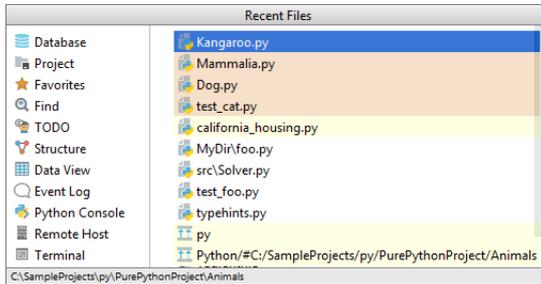


Recent files

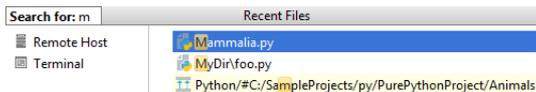
Most of the time, you switch between a handful of files you are actively working on. The other files are less necessary. PyCharm has the features making this workflow very convenient.

Suppose you are editing a function, and at the same time you're writing a test for this function - it means that you are switching back and forth.

This is where [navigating to recent files](#) comes to help: press `Ctrl+E`. Note that the most recent file is listed first and selected by default - you just need to press `Enter`.



Speed search is available in the recent files pop-up. Just start typing what you are looking for, and you'll see the result that matches the search string:

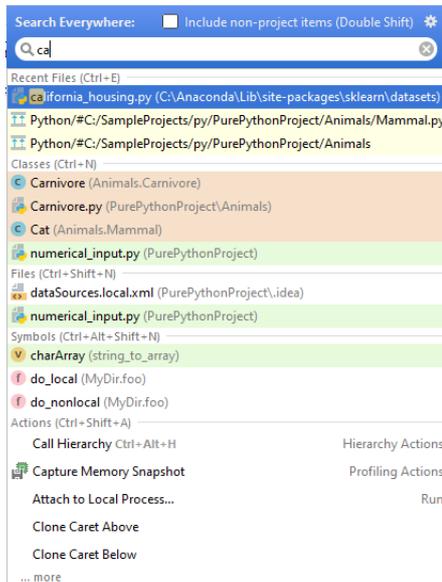


Note that you can also jump to a tool window (e.g., open a new terminal session).

What if you are interested in the recently edited (not just visited) files? In this case, use the keyboard shortcut `Ctrl+Shift+E`. Note that speed search and navigation to the tool windows are also available.

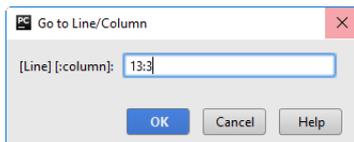
Searching everywhere

PyCharm gives you the chance to find anything anywhere, and jump directly to the search result. Thus you can find the same classes, files and symbols, actions and settings. To do that, just double-press `Shift`, or click the magnifier glass icon  in the right-hand corner of the PyCharm window, and here you are:

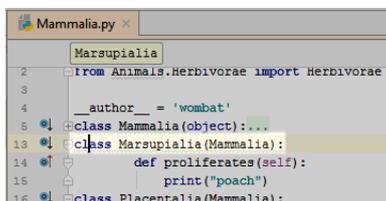


Jumping to a line

And finally, let's open a file for editing (`F4`). Let it be `Mammalia.py`. Then, press `Ctrl+G`:



You'll find yourself at the specified line, and the specified column:



Summary

This brief tutorial is over. You've mastered the following features:

- Finding any [file, class or symbol](#) by its name, and navigating to it.
- Jumping to [recent files](#).
- [Searching everywhere](#).
- Jumping to the specified location in the currently edited file.

What's next?

Let's proceed with [Part 2](#) and see how to jump to a declaration and implementation. Besides that, you see how to create a test, or navigate to an already created one.

Part 2. Navigating to a Declaration, Implementation and Test

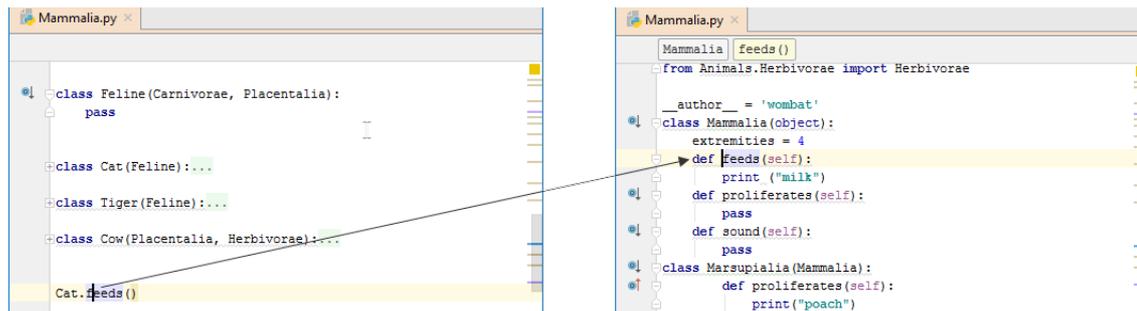
In this section:

- [Navigating to declaration](#)
- [Navigating to implementation](#)
 - [Side note about pin](#)
- [Navigating with gutter icons](#)
- [Jumping to a test](#)
- [Summary](#)
- [What's next?](#)

Note To be able to try the functionality described in this tutorial and follow it step by step, you'll need an example source code. It can be found [here](#).

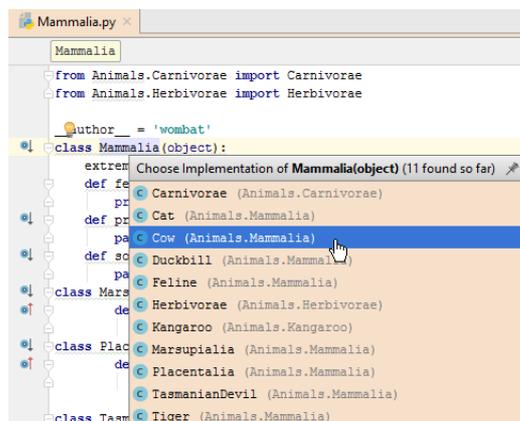
Navigating to declaration

Place the caret at the method `feeds` of the instance of the class `Cat`, and press `Ctrl+B`. PyCharm jumps to the declaration of the method `feeds`, which is declared in the class `Mammalia`:

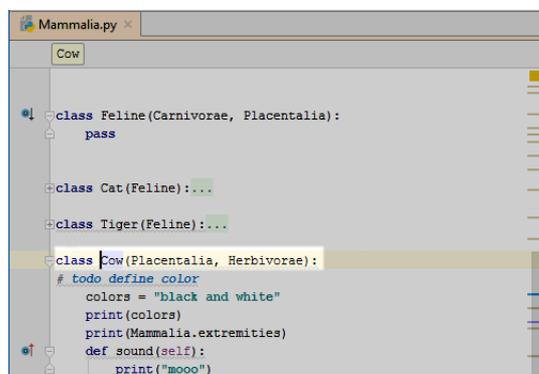


Navigating to implementation

Now, place caret at the declaration of the class `Mammalia` and try to find out which other classes implement it. To do that, press `Ctrl+Alt+B`. You see a rather long list of classes implementing `Mammalia`:



Ok, choose whichever implementation you need (for example, `Cow`), and press `Enter`. PyCharm navigates to the selected implementation and places the caret at the class `Cow` declaration:



Should you select, for example `Carnivorae`, which resides in a separate file, this file would open in a separate editor tab.

Side note about pin

Presumably, you have already noticed the pin icon  in the upper-right corner of the pop-up window. The same icon appears, for example, in the quick documentation lookup (`Ctrl+Q`). If you click this pin, the whole pop-up window will be "pinned", which in the case of navigation and search means that all the encountered occurrences will be presented in the [Find](#) tool window.

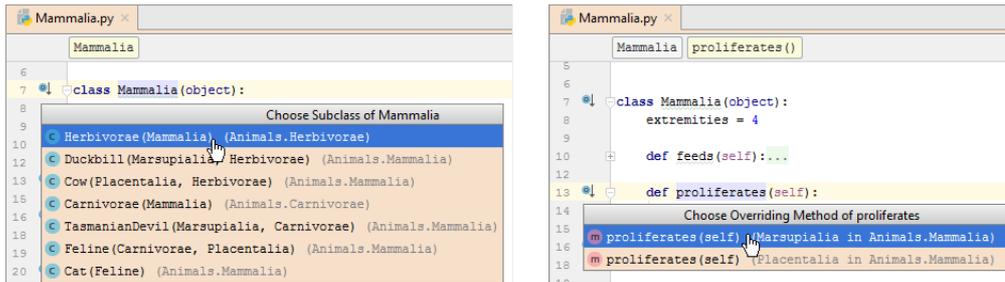
Navigating with gutter icons

Now, let's look at the left gutter. You see there a number of icons with the arrows pointing up or down. What does it mean?

If you hover your mouse pointer over an icon, PyCharm will show the list of child classes or overriding methods (in case of the down arrow), or the parent classes (in case of the up arrow):



What happens, if you click an icon? If a certain class is subclassed, or a method is overridden in more than one class, PyCharm will suggest to select the desired target from the list:

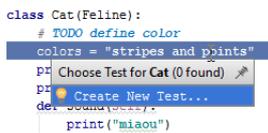


After that, PyCharm jumps to the selected target, and places the caret at the class (method) declaration. If there is only one superclass/subclass, or method, then such a navigation is done silently.

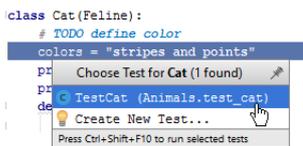
Jumping to a test

Note that testing functionality and setting up the test runner are out of scope of this tutorial. Refer to the section [Testing](#) and [Testing Frameworks](#) for details.

Place the caret at the class `Cat` in the file `Mammalia.py`. Then, press `Ctrl+Shift+T`. You see the suggestion to create a test:



When you've created a test, on pressing `Ctrl+Shift+T`, you see both the existing test, and suggestion to create a new one:



Summary

This brief tutorial is over. You've mastered the following features:

- Navigating to a [declaration](#).
- Navigating to an [implementation](#).
- Using the gutter icons  or  to jump to an implementation or declaration.
- Creating a test and jumping to it.

What's next?

Let's proceed with [Part 3](#) and see how to find usages of a class or symbol.

Part 3. Finding Usages

In this section:

- [Introduction](#)
- [Finding all usages](#)
- [Other types of finding usages](#)
 - [Changing search options, or finding usages via dialog](#)
 - [Viewing usages as a list](#)
 - [Viewing usages in the current file](#)
- [Summary](#)
- [What's next?](#)

Note To be able to try the functionality described in this tutorial and follow it step by step, you'll need an example source code. It can be found [here](#).

Introduction

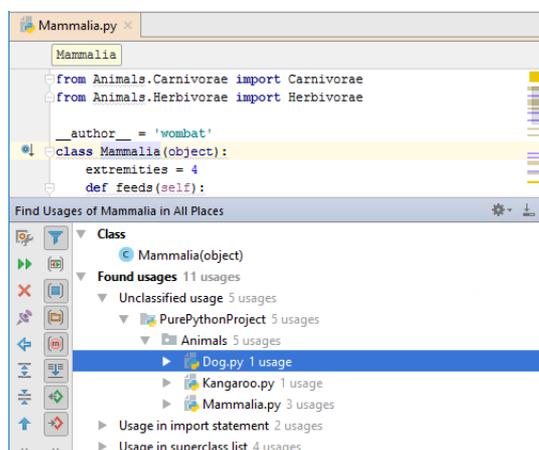
[Finding usages](#) is another search facility, which can be also perceived as the navigation feature. Suppose, you want to find all the usages of a certain class or method across the entire project, which could amount to a huge number of occurrences!

This is where PyCharm comes to help. Let's see how.

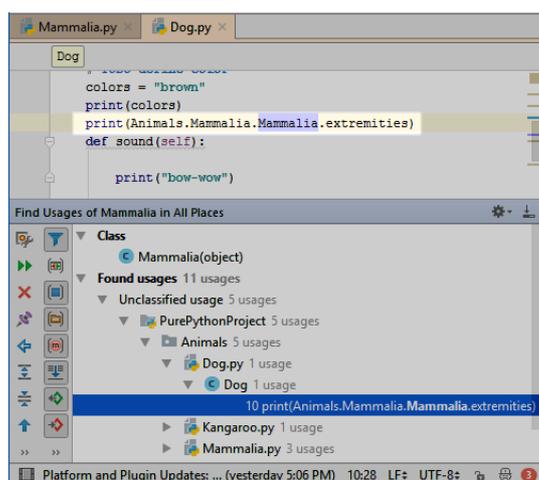
Finding all usages

For example, let's find all usages the class `Mammalia`, and then navigate to one of the encountered usages.

Place the caret at the class declaration, and press `Alt+F7`. The encountered usages of this class show up in the [Find](#) tool window:



If you select one of the usages in the [Find](#) tool window and press `Enter`, PyCharm will open the corresponding file in the editor, with the caret at the class `Mammalia` usage:

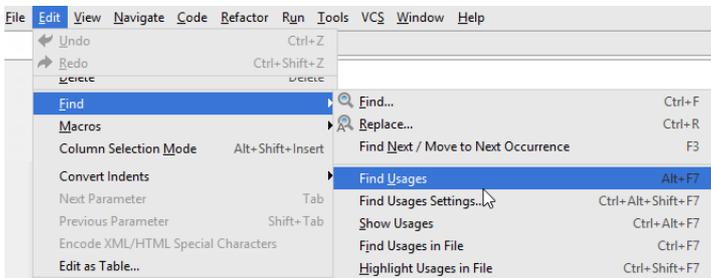


By the way, you can find this action on the context menu of any symbol; for example, on the context menu of the class `Mammalia`.

This way you can find usages of a symbol with the default settings (across the entire project, overwriting the contents of one tab in the [Find](#) tool window).

Other types of finding usages

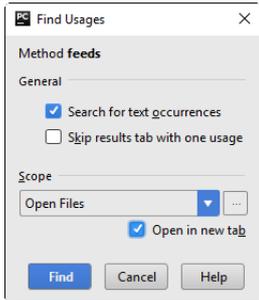
Besides finding usages, PyCharm provides several other actions performing actually the same task, but in a slightly different manner. All these actions are available on the main menu (Edit | Find):



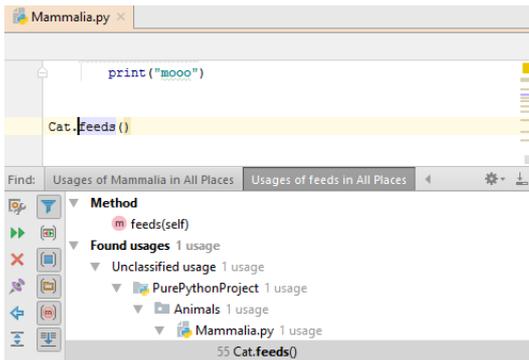
Some of these actions are mapped to the keyboard shortcuts by default. Let's explore them in details.

Changing search options, or finding usages via dialog

Place the caret at the declaration of a symbol, for example, at the declaration of the method `feeds`, and press `Ctrl+Shift+Alt+F7`. You see the dialog box, where you can change the search options. For example, you prefer to look for usages of a method in the open files, and open each search results in a new tab in the `Find` tool window:



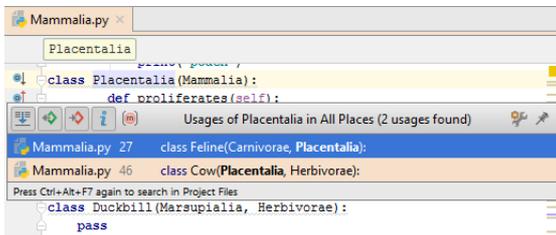
Click the button Find: PyCharm shows a new tab in the `Find` tool window, with the found usage of the method `feeds`. Double-click this entry (or use the arrow keys and `Enter`) - PyCharm opens the editor with the caret at the corresponding usage:



Viewing usages as a list

In some cases, viewing usages in the `Find` tool window seems inconvenient. PyCharm provides an action that shows usages as a pop-up list. For example, let's see the usages of the class `Placentalia`.

So place the caret at the class declaration, and then press `Ctrl+Alt+F7` (or choose Edit | Find | Show Usages on the main menu):



If you select one of the usages using the arrow keys and `Enter` (for example, class `Feline`), PyCharm will jump to the corresponding usage and place the caret at it.

Now look at the toolbar of the pop-up window. If you still think that it would be nice to view usages in the `Find` tool window, click the pin button. The pop-up list disappears, and instead you see the search results in the well-known `Find` tool window.

Finally, if you are not happy with the search options, click to show the `dialog box`.

Viewing usages in the current file

To view usages of a symbol in the current file, press `Ctrl+Shift+F7` (or choose Edit | Find | Highlight Usages on the main menu):

```
Mammalia.py x
Placentalia
from Animals.Carnivorae import Carnivorae
from Animals.Herbivorae import Herbivorae

__author__ = 'wombat'
class Mammalia(object):...
class Marsupialia(Mammalia):...
class Placentalia(Mammalia):
    def proliferates(self):...
class TasmanianDevil(Marsupialia, Carnivorae):...
class Duckbill(Marsupialia, Herbivorae):...
class Feline(Carnivorae, Placentalia):...
class Cat(Feline):...
class Tiger(Feline):...
class Cow(Placentalia, Herbivorae):...
```

As you see, each of the usages is marked with a stripe in the right gutter. When you hover your mouse pointer over such a marker, a balloon with the description of a specific usage appears. If you click the stripes, you can navigate from one usage to another.

Summary

This tutorial is over - congrats! Here you've learned how to:

- Use the various Find Usages operations.
- Use the right gutter as the source of information.

What's next?

Let's proceed with [Part 4](#) and see how to navigate using Navigation bar only.

Part 4. Using the Navigation Bar

In this section:

- [Introduction](#)
- [How to make the Navigation Bar visible?](#)
- [How to open a file using the Navigation Bar ?](#)
- [What to do if the list of files in the Navigation Bar is too long ?](#)
- [Is the right-click menu available on the Navigation Bar?](#)
- [Summary](#)
- [What's next?](#)

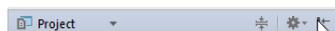
Note To be able to try the functionality described in this tutorial and follow it step by step, you'll need an example source code. It can be found [here](#).

Introduction

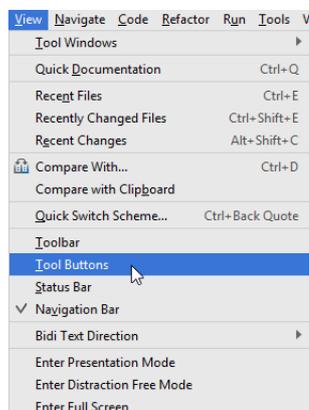
The Navigation Bar is an alternative to the [Project tool window](#) that can be perceived as a means of navigation. For example, to open a file, locate it in the Project tool window and press `F4`, use the right-click menu to find usages or find in path etc.

However, the Project tool window eats the screen space. So to save the screen space for the editor, let's get rid of the Project tool window.

In this tutorial we'll navigate through the source code using the [Navigation Bar](#) only! So let's close the Project tool window by clicking the button!":



Then, (to completely get rid of it) clear the check command Tool Buttons on the View menu:

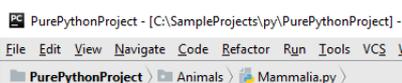


How to make the Navigation Bar visible?

If you don't see the Navigation Bar, make it visible. To show the Navigation Bar, do one of the following:

1. On the View menu, select the check command Navigation Bar.

Now you see the Navigation Bar on top of PyCharm window, under the title bar and the main menu:



2. Press `Alt+Home`.

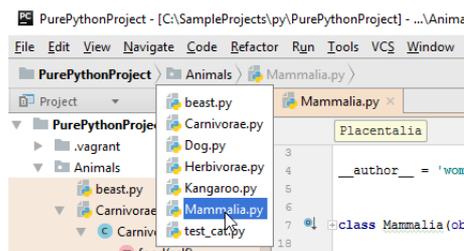
The Navigation Bar shows in the middle of the editor pane (or PyCharm window, if no files are open in the editor). To hide the Navigation bar, press

`Escape`.

Now see the buttons starting at the project root, all the way down to the currently-visible file.

How to open a file using the Navigation Bar ?

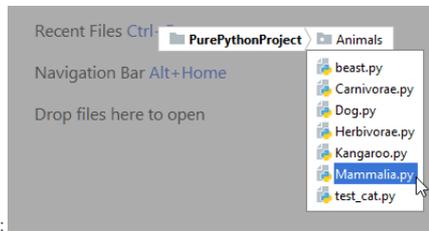
If you want to open the file `Mammalia.py` in the folder `Animals`, you simply click `Animals` on the Navigation Bar, and then click `Mammalia.py`:



However, you can use the keyboard only, not using your mouse pointer. This is how it's done:

1. Show the Navigation Bar (`Alt+Home`).
2. Press `Enter`. The nested directories and files are shown, and you should use the arrow keys to navigate through the list. Repeat this step as required.

3.

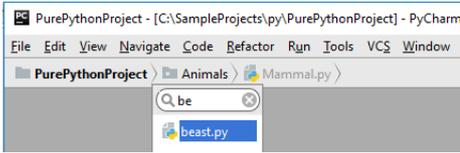


Finally, press `Enter` to open the desired file:

What to do if the list of files in the Navigation Bar is too long ?

If you have too long a list of files in your folder, then all its content won't fit in the Navigation Bar window. In this case, use [speed search](#) - it works in the Navigation Bar as well as in the tool windows.

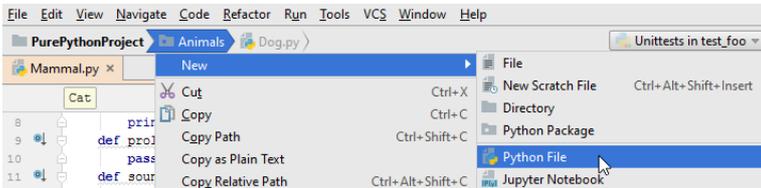
Just click the desired folder in the Navigation bar and start typing:



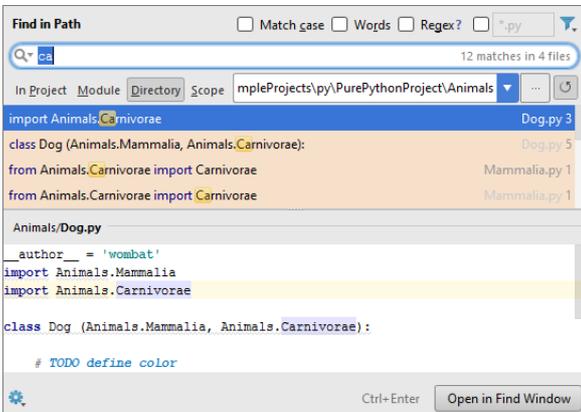
Is the right-click menu available on the Navigation Bar?

The answer is yes. Right-click the desired button in the Navigation Bar and choose any of the available commands.

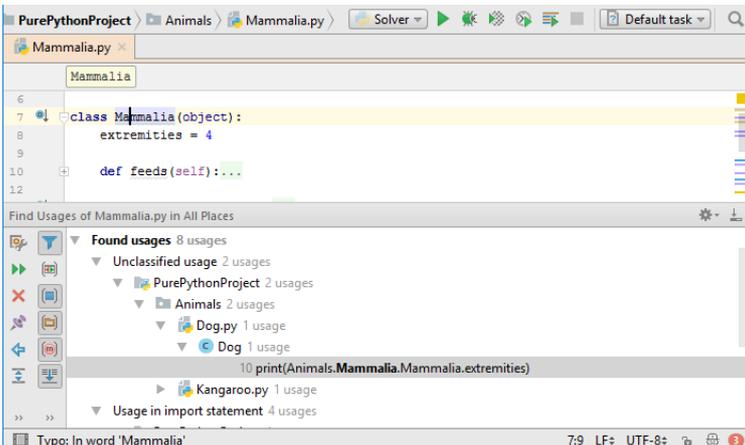
For example, PyCharm allows easy creation of a file or folder from the Navigation Bar. To do that, right-click the desired folder button in the Navigation Bar, and choose one of the available file types on the New menu:



Also, use the command Find in Path. Right-click the folder `Animals` in the Navigation Bar, choose Find in Path, type `ca` in the dialog box that opens, and observe results:



Or you can use Find Usages:



Summary

This tutorial is over - congrats! Here you've learned how to:

- Navigate around your source code using the Navigation Bar only

_ navigate around your source code using the navigation bar only.

- Use the keyboard shortcuts for navigation purposes.

What's next?

Let's proceed with [Part 5](#) and learn how to navigate between the various parts of PyCharm's UI. Learn also how to do that using the keyboard shortcuts.

Part 5. Navigating to the UI Elements

In this section:

- [Introduction](#)
- [Using the Switcher](#)
- [Jumping between PyCharm's components](#)
- [Summary](#)
- [What's next?](#)

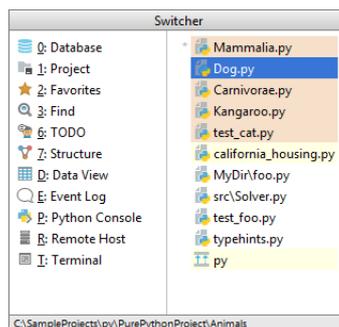
Note To be able to try the functionality described in this tutorial and follow it step by step, you'll need an example source code. It can be found [here](#).

Introduction

In this tutorial, we'll explore the various types of navigation between the elements of PyCharm's UI.

Using the Switcher

The **Switcher** works same as `Alt+Tab` / `⌘+Tab` and is an extremely efficient pop-up.



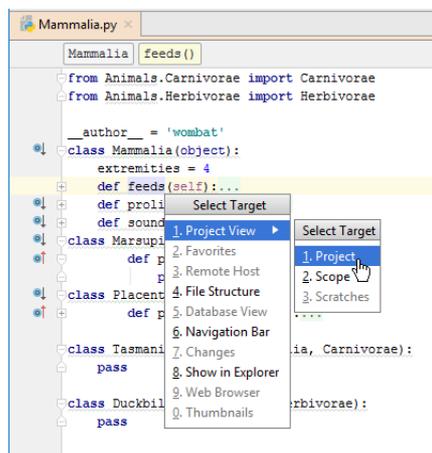
To show this pop-up, press `Ctrl+Tab` - and keep holding `Ctrl`. As soon as you release the `Ctrl` button, the Switcher disappears, leaving you with what you've just selected.

While the Switcher is visible, move around with the arrow keys, or `Tab` / `Shift+Tab` to move forward/backwards through the Switcher columns.

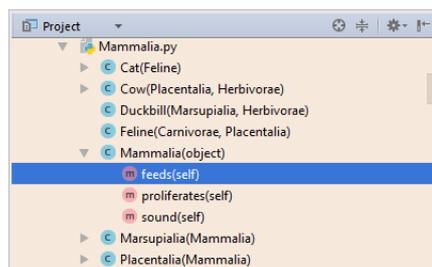
Jumping between PyCharm's components

Suppose you've selected a class member in the editor, and would like to find it in the Project view. This is how it's done...

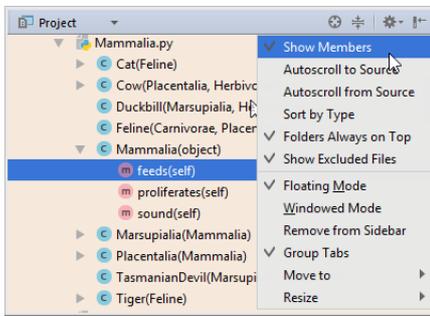
Place the caret at the method `feeds` of the class `Mammalia.py`, and press `Alt+F1` (Navigate | Select In...). From the list of available components, choose Project View, and then Project.



The caret resides at method `feeds` of the class `Mammalia.py` in the Project View:

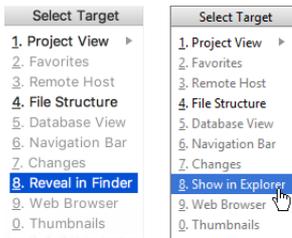


You only have to make sure that the option Show Members is selected:

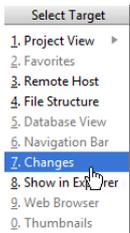


Naturally, if this option is not selected, only the class `Mammalia.py` will get the focus.

If you want to find the opened in the editor file in the File Explorer/Finder, you can also do it using Select In.... Just select the option Reveal in Finder (for macOS) or Show in Explorer (for Windows and *NIX):



Finally, suppose that a file opened in the editor has some changes that should be pushed to a Git repository. To jump to the Version Control tool window, again press `Alt+F1` and choose Changes in the list of components:



Summary

This tutorial is over - congrats! Here you've learned how to:

- Work with the [Switcher](#).
- Use Select In... functionality to find a file in the Project or Version control tool windows, or to find a file in the Explorer/Finder.

What's next?

Let's summarize everything we've learned in this series of tutorials:

- [Find by name, jump to recent and search everywhere](#) - the most basic navigation facilities.
- [Navigate to a declaration, implementation or test](#).
- [Find usages of a symbol](#)
- [Use the Navigation Bar](#) to open files, apply the context menu, and go for the keyboard shortcuts.
- Finally, [this tutorial](#) relates to the navigation to the elements of PyCharm UI.

You see that PyCharm has many navigation facilities. Refer to [navigation](#) and [search](#) sections of PyCharm documentation.

However, PyCharm provides Django support as well. So if you are a Django developer, proceed with [Part 6](#) and see how to navigate between Django components.

Part 6. Django-Specific Navigation

This feature is supported in the Professional edition only.

In this section:

- [Introduction](#)
- [Jumping from a view to a template](#)
- [Jumping from a template to a view](#)
- [Jumping between views and urls.py file](#)
- [Navigating to implementation and declaration](#)
- [Jumping to tests](#)
- [Summary](#)

Introduction

Finally, we'll explore the Django-specific navigation.

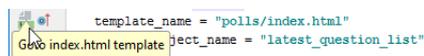
Note that same type of navigation is available for the other frameworks: Pyramid, Flask etc.

Use the example from the "first steps" guide [Creating and Running Your First Django Project](#).

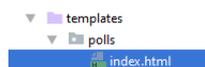
Jumping from a view to a template

Open the file `views.py` for editing (F4). In the left gutter, next to the line `template_name = "polls/index.html"`, you see the icon .

Hovering the mouse pointer over this icon reveals the following pop-up window:

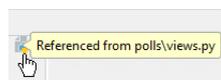


Clicking this icon results in jumping directly to the template `index.html`, that resides in the folder `polls` under `templates`:



Jumping from a template to a view

In the left gutter of the template file `index.html`, you see the icon . If you hover your mouse pointer over this icon, you'll see the following pop-up window:

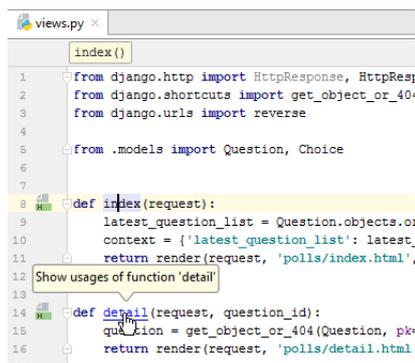


Clicking this icon leads you directly to the corresponding view.

Jumping between views and urls.py file

PyCharm allows you to easily navigate between a particular view and the corresponding url. This is how it's done:

1. Hover your mouse pointer over the view name, while keeping `Ctrl` key pressed, and see the view name turning into a hyperlink:



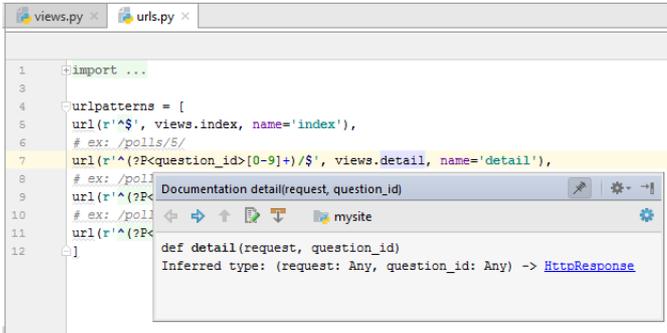
2. If you click this hyperlink, you'll jump directly to the corresponding URL:



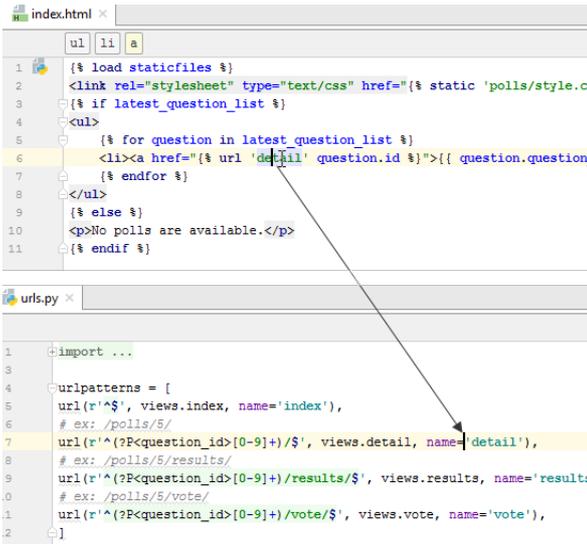
Vice versa, you can also jump from a URL to the corresponding view. Again, in the `urls.py` file, hover your mouse pointer over the view name, while keeping `Ctrl` key pressed, and see the view name turning into a hyperlink.

`Ctrl` + click the view name - and find yourself in the corresponding view.

By the way, if you click a pin icon in the pop-up, this pop-up becomes "pinned" - that is, turns into a tool window:

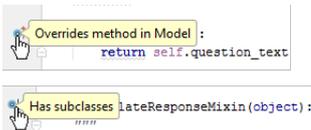


Finally, open a template in the editor (F4), say, `index.html` and hover your mouse pointer over `detail`, while keeping `Ctrl` key pressed, and see the name turning into a hyperlink. Click this hyperlink - you immediately jump to the file `urls.py`:

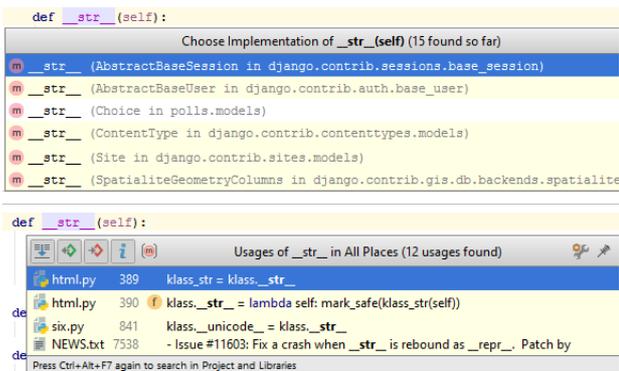


Navigating to implementation and declaration

In a Django project, you see the same icons and as in a pure Python project. When you hover your mouse pointer over such an icon, you see a pop-up window, like the following:

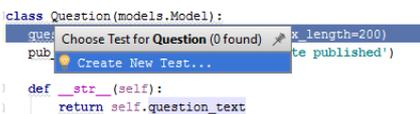


You can also jump to a declaration or implementation of a symbol. To do that, just place the caret at the implementing/overriding symbol and press `Ctrl+B` or `Ctrl+Alt+B` (or choose `Navigate | Declaration` or `Navigate | Implementation` on the main menu):

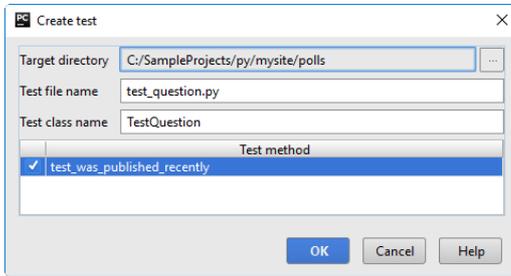


Jumping to tests

Django applications can be tested same as the pure Python ones. Same way you can create tests for the Django projects, for example, press `Ctrl+Shift+T`:



Fill in the form in the Create test dialog:



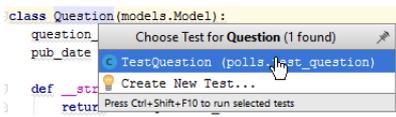
You'll see the code like:

```
from unittest import TestCase

class TestQuestion(TestCase):
    def test_was_published_recently(self):
        self.fail()
```

The test class naturally fails, but it's up to you to write some meaningful code.

When a test already exists, you can jump to it, using the same keyboard shortcut `Ctrl+Shift+T`:



(Note that choosing a particular test runner can be done in the Settings/Preferences dialog, on the page [Python Integrated Tools](#).)

Summary

This brief tutorial is over. You've mastered the following features:

- Used the gutter icons to navigate between views and templates.
- Navigated between views and urls.
- Repeated how to jump to an implementation or declaration.
- Repeated how to navigate to an existing test or create a new one.

File Watchers in PyCharm

This feature is supported in the Professional edition only.

In this section:

- [What this tutorial is about](#)
- [Prerequisites](#)
 - [Installing Node.js plugin](#)
 - [Installing LESS and CoffeeScript compilers](#)
- [Configuring file watchers](#)
 - [Configuring file watcher for LESS files](#)
 - [Configuring file watcher for CoffeeScript](#)
- [Editing file watchers](#)
- [Troubleshooting, or if an error occurs?](#)

What this tutorial is about

This tutorial aims to walk you step by step through using file watchers in PyCharm.

The basics of file watchers, in particular, usage of LESS and CoffeeScript, are out of scope of this tutorial.

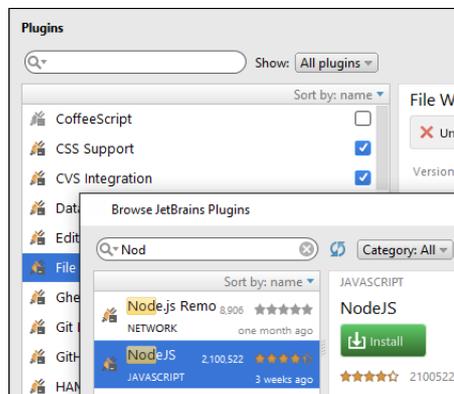
Prerequisites

Make sure that:

- You are working with [Professional edition of PyCharm](#).
- [Node.js](#) is downloaded and installed. It is advisable, depending on your particular operating system, to add the path to Node.js executable to the Path environment variable.
- Before you start working with file watchers, make sure that the File Watchers plugin is enabled. The plugin is bundled with PyCharm and is activated by default. If the plugin is not activated, enable it on the [Plugins](#) page of the [Settings / Preferences Dialog](#) as described in [Enabling and Disabling Plugins](#).
- It is advisable to familiarize yourself with the matter in advance. Please read the section [Using File Watchers](#).
- In this tutorial, we'll work with [Less](#) and [CoffeeScript](#) files. So, before you start your workout, perform some preliminary steps.

Installing Node.js plugin

First, download and install Node.js plugin. It is not bundled; so to have it installed, open the [Plugins](#) page (click  on the main toolbar, remember ?) and look for this plugin in the JetBrains repository:

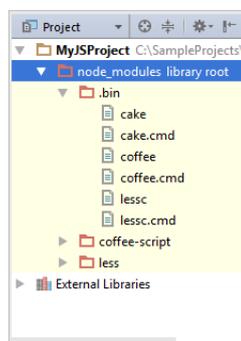


For the changes to take effect, restart PyCharm. After restart, you will notice a new page under the Languages and Frameworks node in the Settings/Preferences dialog - [Node.js and NPM](#).

Installing LESS and CoffeeScript compilers

Open Settings/Preferences () , and open then the page [Node.js and NPM](#). On this page, specify the Node interpreter (its version is determined automatically), and then click  - one time to install less, and the other time to install coffee-script.

As you've noticed already, LESS and CoffeeScript are installed locally, so the corresponding compiler files are written under the project root:



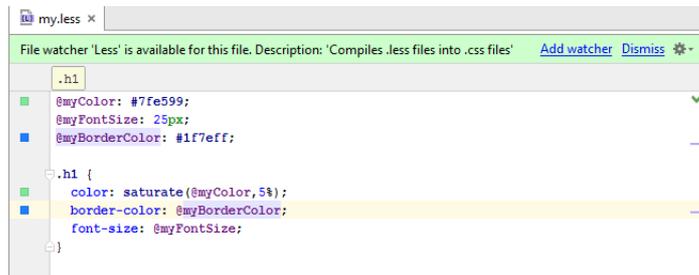
These files will be required a little bit later. Now, it's time to start!

Configuring file watchers

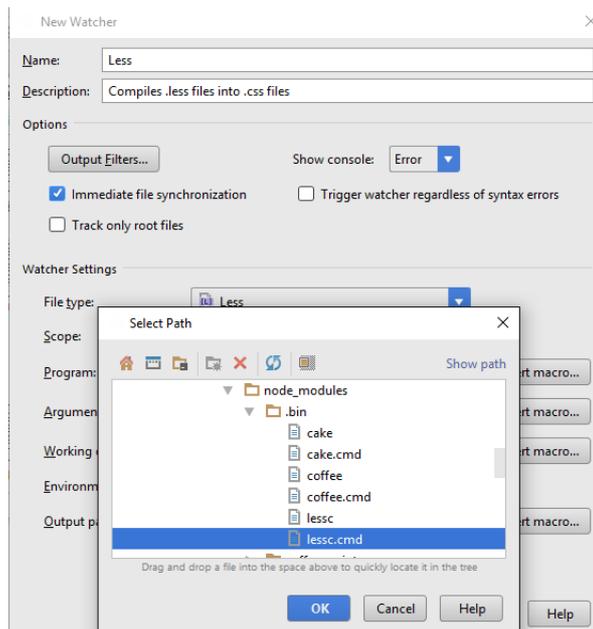
When PyCharm detects that you are working with a file it can "watch," it prompts to set up a File Watcher.

Configuring file watcher for LESS files

For example, when you open for editing a LESS file, PyCharm shows a notification banner:



Click the link Add watcher. PyCharm shows the following dialog box, where you have to specify your file watcher type (Less here), executable (`lessc.cmd` here), and select the option to generate output from stdout:



Looking at this configuration, you can easily figure out what the file watcher actually does:

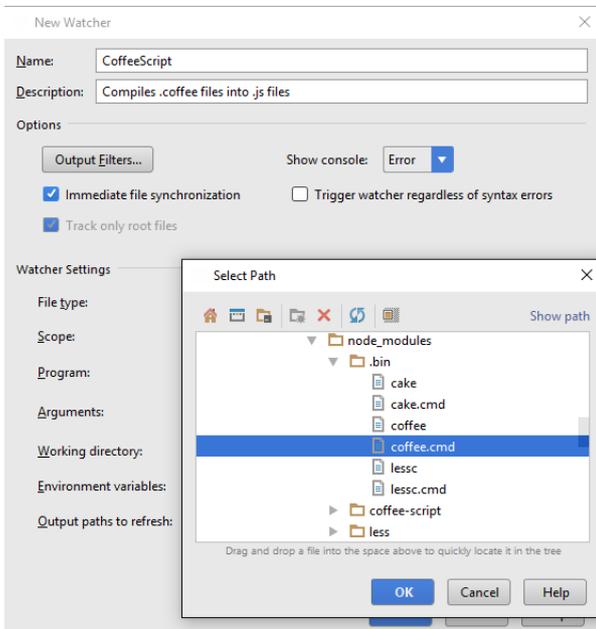
- Watches for changes on all Less files within your project.
- Compiles files with the extension `.less` into the files with the extension `.css`, using the compiler `lessc.cmd`, specified in the field Program.

Configuring file watcher for CoffeeScript

Next, open for editing a CoffeeScript file. PyCharm immediately prompts you to configure a file watcher for it:



Again, click Add watcher and specify the file watcher settings, in particular, the CoffeeScript executable:

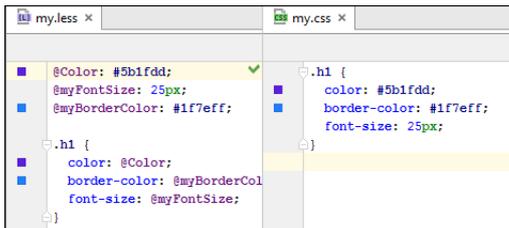


What does this file watcher do?

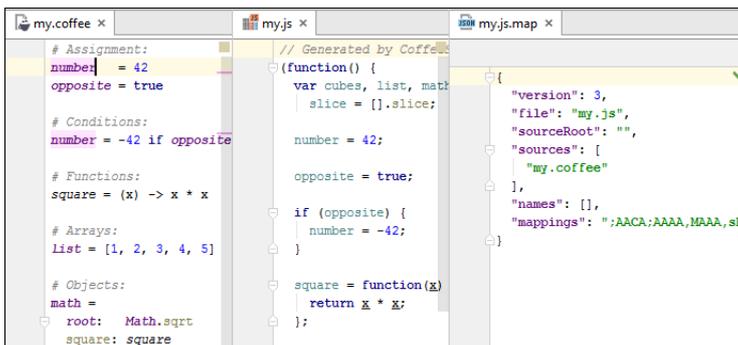
- It also tracks changes in all CoffeeScript files in your project.
- If compiles files with the extension `.coffee` into the files with the extension `.js`, using the compiler `coffee.cmd`, specified in the field Program.
- It compiles files with the extension `.coffee` into the files with the extension `.map`, using the compiler `coffee.cmd`, specified in the field Program.

Editing file watchers

OK, here we are. Open for editing the file `my.less`, and change something, for example, rename the variable `@myColor` to `@Color`, and change its value. The file watcher immediately processes the changed source file, and produces an output file with the extension `.css`:



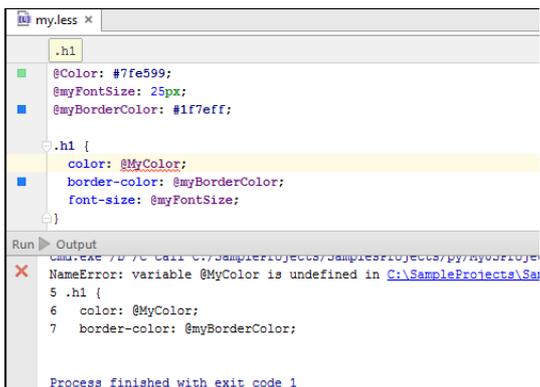
Next, open for editing a CoffeeScript file and change something there. The configured file watcher generates a JavaScript file and a source mapping file:



Note that in either case PyCharm shows generated files in the [Project Tool Window](#) under the source files.

Troubleshooting, or if an error occurs?

If a command line tool executed by the File Watcher fails, PyCharm shows its output in the [Run tool window](#):



Helpful for troubleshooting, isn't it?

Finding and Replacing Text in File Using Regular Expressions

On this page:

- [Example code](#)
- [Finding and replacing a string using regular expression](#)
- [Changing case of the characters](#)

Example code

Consider the following XML code fragment:

```
<new product="ij" category="105" title="Multiline search and replace in the current file" />
<new product="ij" category="105" title="Improved search and replace in the current file" />
<new product="ij" category="105" title="Regex shows replacement preview" />
...
```

Finding and replacing a string using regular expression

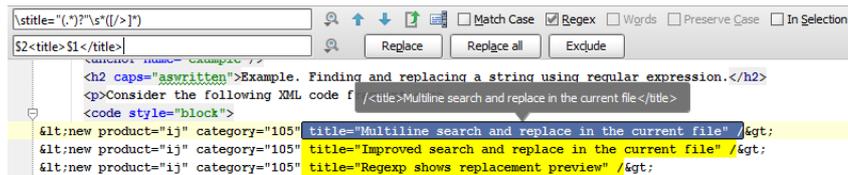
Suppose you want to replace an attribute within an element (`title`) with an expanded tag `<title></title>`, which contains some arbitrary string in double-quotes within.

This is how it's done.

1. With the XML file in question opened in the editor, press `Ctrl+R`. The Replace pane appears on top of the editor.
2. Since you want to replace all the `title` attributes, regardless of the actual strings contained therein, use regular expressions. Make sure that the check box `Regex` is selected. Thus everything you type in the Search and Replace fields will be perceived as the regular expressions.
3. In the Search field, start typing regular expression that describes all `title` attributes. Note that though the regular expression `\stitle=".*?"\s*/>]*` matches contents of the `title` attribute, it is recommended to capture the groups for referencing them in the Replace field:

```
\stitle="(.*?)"\s*/>]*
```

Note that for the regular expressions replacement preview is shown at the tooltip:



4. Then, in the Replace field, type the following regular expression:

```
$2<title>$1</title>
```

5. Click `Replace`, or `Replace All`.

As you see, the second capture group (`/>`) is moved ahead to close the `<new>` element, while the first capture group `<`, which matches any string in double quotes, is moved to the element `<title>`.

Changing case of the characters

Suppose now that you want to change characters of the search strings. Again make sure that `Regex` check box is selected.

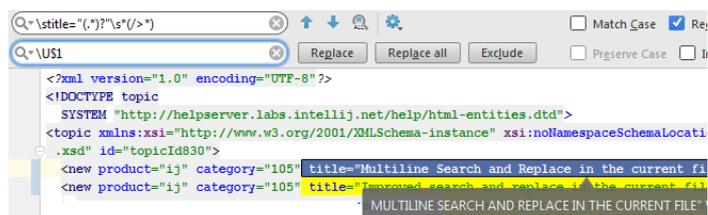
In the Search field, type the search expression:

```
\stitle="(.*?)"\s*/>]*
```

Next, fill in the Replace field with the following expression:

```
\U$1
```

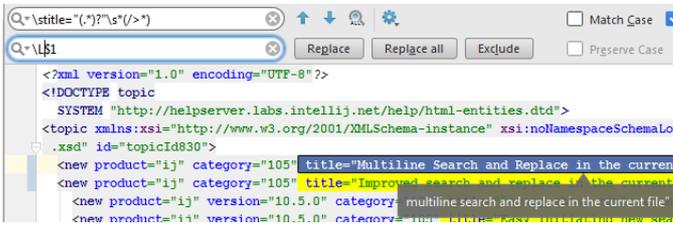
The found occurrences are replaced with the upper-case characters:



Next, let's make the strings all lower-case. To replace occurrences with the lower-case characters, type the following replacement string:

```
\L$1
```

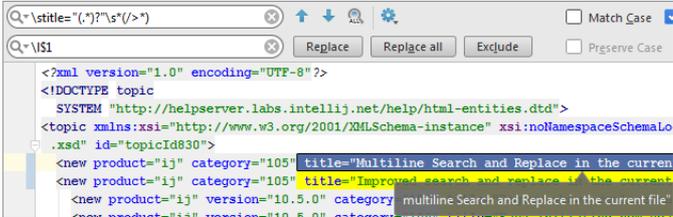
Then the suggested replacement will be:



And finally, if you want change the case of the first letter only, type the following replacement string:

\\\$1

PyCharm suggests the following replacement:



Migrating from Text Editors

This guide is for the Python developer who uses a text editor such as Vim, Emacs, or Sublime Text and needs information about switching to PyCharm. It includes a brief discussion of the IDE concept, then explores specific differences.

In this section:

- [What is an IDE?](#)
- [The PyCharm interface](#)
- [Projects](#)
- [Your first project](#)
- [IdeaVim](#)
- [Emacs and emacsIDEAs](#)
- [Customizing and extending](#)
- [Multiple cursors](#)
- [Tabs and split windows](#)
- [Running code](#)
- [Tips and tricks](#)
- [Additional resources](#)

What is an IDE?

Python developers have long used a variety of tools to write their code. Although the lines are blurry, these tools fall into two broad camps: text editors and integrated development environments (IDEs). Text editors, to oversimplify, focus on the editing of a single file, giving a very lean-and-mean experience.

IDEs, as the name implies, have a broader view. They want to look at your entire project, and all of your coding-related activities, and unify these into a consistent, powerful UI. While these two segments overlap, the overall scope is the primary difference.

Because an IDE like PyCharm looks at all of the code – in your project, in your dependencies, and in the platform itself – it can provide much assistance in coding activities. This analysis and assistance shows in many powerful facilities:

- Auto-completion speeds up accurate coding by finishing your typing
- Quick fixes spot common mistakes and provide language-specific corrections
- Code Intentions suggest optimizations and improvements for common Python patterns
- Code Refactoring takes the drudgery out of frequently-used Python refactoring
- Templates automate recurring tasks
- Code Navigation analyzes the structure and semantics of all your code, and code it uses, to provide rich ways to move around in your software

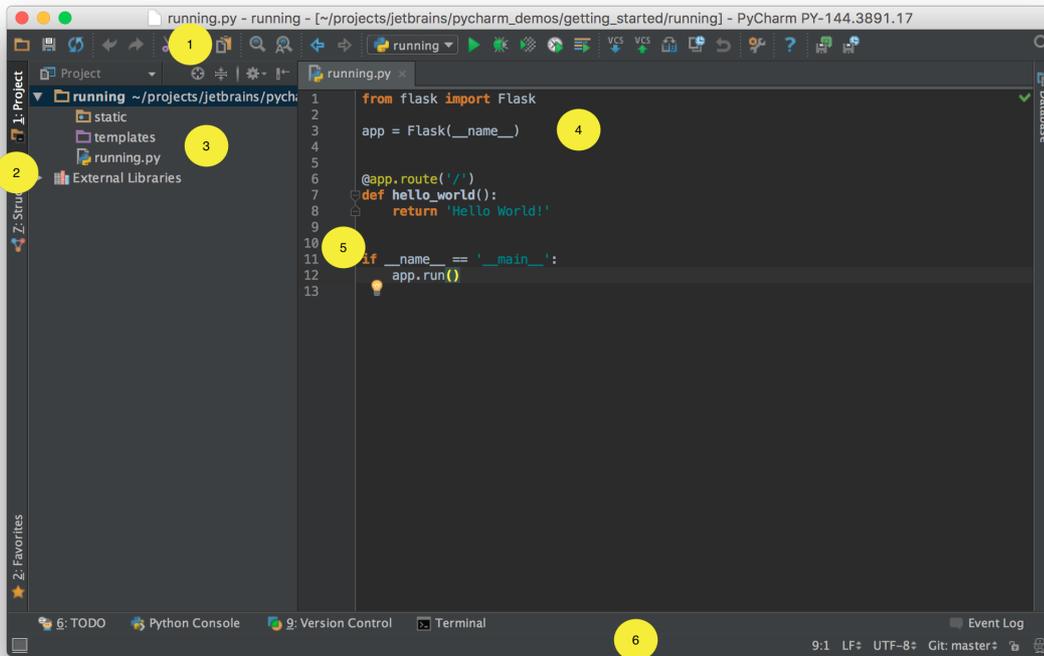
While text editors provide some of these facilities, PyCharm goes beyond string-oriented operations and addresses the *semantics* of your code and the language, providing intelligent assistance. PyCharm understands Python, and understands your code, so it can jump to where a symbol is defined, or where your class is used, or refactor a method and its usages across the code base. In fact, these capabilities extend beyond Python: from its brethren in the IntelliJ family, PyCharm inherits first-class IDE support for web development and many other areas.

Beyond these coding assistance facilities, PyCharm ties in related operations into the same environment: version control, database management, profiling, test running, remote environments, and more. Each of these facilities is delivered in an integrated fashion: applying a change while view a diff can automatically trigger a re-run of your tests.

Finally, this power comes with a product-focused mindset: everything just works. While other tools have a long list of *possible* capabilities, you're on your own to assemble and maintain the aggregate. That's fine for people that like to tinker. With PyCharm as your IDE, these features work, out-of-the-box, and are professionally supported, year after year.

The PyCharm interface

Since an IDE such as PyCharm has such a broad scope, the user interface (UI) requires more orientation than a simple text editor. For example, here is a typical project in PyCharm:



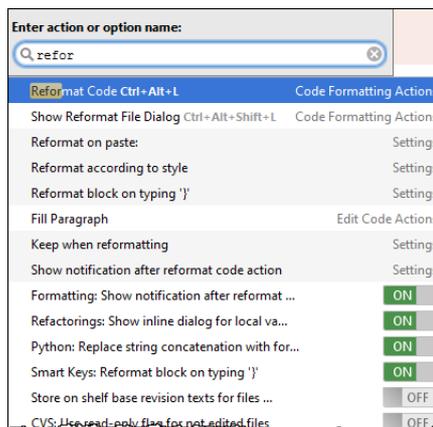
As it turns out, PyCharm's [project-oriented IDE interface](#) is well organized and easy-to-understand:

1. The toolbar provides quick access to certain actions. These actions are also available in the menu, via keyboard shortcuts, and through action searches.
2. The tool windows docked on the left, right, and bottom sidebars [give a rich interface](#) into IDE features such as projects, structure, databases, run windows, and more. These tool window icons can be organized in rich ways.
3. The project tool window navigates resources inside, and external assets linked into, the project.
4. The editor provides a tabbed interface for editing files, databases, and other assets.
5. The left and right gutters provide folding, breakpoints, and show results of inspections.
6. The status bar gives several indicators and operations.

Still too much? If you are transitioning from a text editor and want the spartan look, PyCharm has several solutions:

- Windows and toolbars can be hidden until needed
- [Distraction-free mode](#) and the other modes remove most of the visual chrome, focusing on the code with interaction done via keyboard shortcuts and action searches

In fact, PyCharm has embraced the modern, uncluttered trend by surfacing most of its operations via a keyboard-centric, search-oriented interaction. Don't want to click in the menus to reformat your code? Not only is there a keystroke sequence for that, but you can hit `Ctrl+Shift+A` to search the actions for "Reformat Code":



Projects

As mentioned, PyCharm looks beyond a file and instead looks at a project as a complete software solution. Not just the software that accompanies the project, but the related tasks as well.

The project contains more than just these artifacts. In a project, you might have 3 files open in tabs, in a certain order. That information is saved in the project's metadata, in a `.idea` subdirectory in the project's root folder. Because of this, the next time you open the project, you will be restored to that configuration. This applies beyond configuration: PyCharm [autosaves changes](#) as you edit, backed by a [local history](#) that can return the code to its previous states. PyCharm has a tremendous amount in its IDE that is part of the project's configuration: Python settings, scenarios for running code, connection information for databases, and more.

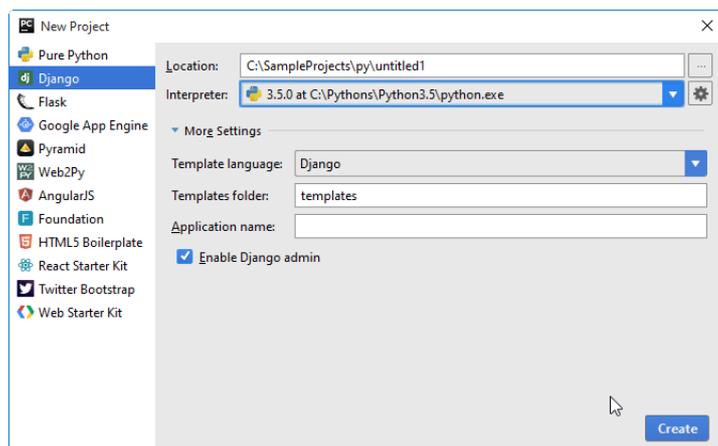
Your first project

If you currently use a text editor, your first step into PyCharm will ask you about creating a project. You have a number of options and opportunities. Let's discuss some of them.

First, you might be starting from a blank slate, or from an existing directory on disk, or with a remote repository that you have yet to check out. PyCharm [has specific facilities to help](#) in all of these cases. It's really nice, for example, to be able to browse your GitHub account's repositories when choosing your project's starting point.

Next, you might have one or more Python interpreters that you want to use with your project. PyCharm [supports locally-installed Python 2 and 3](#), as well as making or re-using [virtual environments](#). It also supports Anaconda-based interpreters, remote interpreters, Vagrant, and Docker. You can set these up before you start PyCharm's "New Project" wizard, or specify (or even create) these during project creation. As well, you can change your mind after you create your project.

With PyCharm Professional Edition, we make it easy to get started with popular types of projects. For example, creating a Django project will generate a directory structure following Django best practices and add a "run configuration" that starts up the Django server in a special tool window.



IdeaVim

Long-time `vim` users have a lot invested in how they use their editor. Fortunately JetBrains has a very powerful and well-supported IntelliJ plugin called [IdeaVim](#) which serves as a "Vim emulation plug-in for IDEs based on the IntelliJ platform".

With this plugin, you can enter a Vim emulator mode inside PyCharm. More information is available at the GitHub page linked above, as well as the [Twitter account](#) and a PyCharm [IdeaVim screencast](#). Although a bit dated, here is an [epic review](#) with an in-depth look by a former `vim` user.

[This tutorial](#) explains how to work with `IdeaVim` plugin in PyCharm.

It's also possible to configure `vim` as an [external tool](#), as described in [this article](#).

Emacs and emacsIDEAs

`Emacs` and `vim` are two popular text editors for hard-core developers. PyCharm provides a [keymap](#) that matches many Emacs key sequences.

While PyCharm doesn't have a plugin for `Emacs` that matches the full breadth of `IdeaVim`, the community has one active plugin named [emacsIDEAs](#) that provides some useful functions from Emacs and Emacs extensions such as AceJump.

You can configure `Emacs` as an [external tool](#) for PyCharm and use it for editing files. This process is described in [this tutorial](#).

Customizing and extending

PyCharm's IntelliJ foundation is very broad (many IDEs built atop it) and very mature, which also leads to another benefit: PyCharm is very customizable. For example, while it ships with a light theme by default, it also provides an optional, darker theme called "Darcula" which matches modern trends in editors. Beyond tweaking theme, numerous look-and-feel options can be customized: font sizes, color schemes, toolbar settings, balloon behavior, and more.

"How it looks" is just the beginning of the customization options. We discussed switching to a different [bundled keymap](#). But you can extend and customize the keymap in many ways, mapping different keystrokes to different actions. Beyond that, PyCharm's behavior can be extended: installing plugins, custom "templates" for actions, changing the code intentions, etc.

Multiple cursors

Here's a power feature that developers from other tools swear by: multiple cursors. Sublime Text pioneered this concept, which IntelliJ added for all the IDEs atop its platform. With [PyCharm's multicursor](#), multiple carets can be used, with IDE actions applying to each caret. This is shown in detail in the [PyCharm Multiple Selection screencast](#).

Tabs and split windows

Mature text editors such as Emacs and vim are renowned for the rich ways they can split the screen into multiple areas, allowing multiple files to be visible at once. These tools go beyond just the basics: you can move around between these areas in rich ways, split vertically or horizontally, re-split, and do all of this from the keyboard.

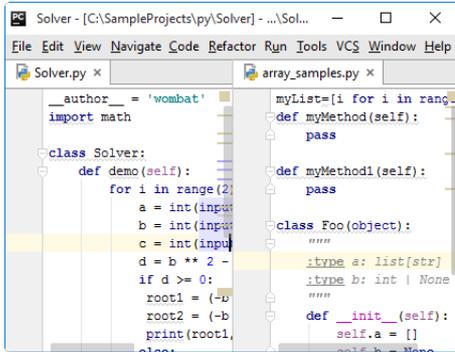
PyCharm also has a [mature set of capabilities](#) for showing multiple files. At its most basic, you can open multiple files into tabs, which can be re-organized manually or sorted alphabetically. Tabs can be pinned or detached into a separate window.

```

1  __author__ = 'wombat'
2  __project__ = 'MySimplePythonApplication'
3
4  ② import math
5
6  class Solver:
7      # the first line
8      # the second line
9      def demo(self):
10         while True:
11             a = int(input("a "))
12             b = i ① input(prompt) builtin
13             c = i ④ raw_input(prompt) builtin
14             d = b
15             if d >= 0:
16                 # REVIEW[wombat] please make sure that

```

Beyond tabs, you can [split the editor window](#) into independent panes to show more than one file/tab at a time. Splitting can be done vertically or horizontally. You can re-arrange currently-open files by moving a tab to the opposite group.



Finally, all of the above can be done in a keyboard-centric way, making it easy to work with multiple files and windows, without using your mouse.

Running code

While text editors provide tools for handling the running of Python code, PyCharm takes this much further with a facility called [run configurations](#). These let you define parameters that are associated with running the Python (or other languages such as JavaScript) code. Then, when you run that run configuration, all the associated settings are applied.

Moreover, PyCharm runs your code in a specialized tool window which provides many features for working with running code: stopping, re-running, etc. The output is displayed in a mini-console and errors show a traceback with lines that can be clicked to jump to that file's line number. This tool window, as well as running and re-running your code, can all be driven from a keyboard.

These run configurations also apply to special kinds of running: test runner configurations with a specialized window for showing test output, debug configurations with the visual debugger, a test coverage run configuration, a profiler run configuration, and even a concurrency diagram configuration. All of these provide real value to the professional Python developer's workflow.

Finally, PyCharm Professional adds new types of run configurations for the frameworks it supports. For example the Pyramid run configuration knows about the Pyramid configuration file and capabilities unique to Pyramid.

Tips and tricks

Running from command-line

While PyCharm is traditionally launched like other applications (from the desktop), you can also [work with PyCharm from the command line](#). These can be setup [during initial install](#). For example, on Linux and macOS, `/usr/local/bin/charm` can be used to open a file in PyCharm. You can also run inspections and use PyCharm's diff tool.

Opening a single file

You don't have to create a project to open some particular file. PyCharm's "Open" menu (or the command-line "charm" program) can open a single file, either in an existing project's window or in a new window.

Scratch Files

If you need a temporary editor for a snippet of code or text, and don't want to save it in a project or even a file, PyCharm provides a facility called [scratches](#). You can create either a scratch file (which is associated with a language and is thus has syntax highlighted, code completion, is runnable, etc.) or a scratch buffer (small chunks of text).

These scratches have many capabilities: you can list them (up to five), close and delete them, associate them with a language, and re-organize them.

Scratches are saved in a special area in the project.

Built-in Tips

When you first install PyCharm, it provides you a friendly tip each time you start it up. It also lets you turn these tips off. If you later want to [view more tips](#), look for "Tip of the Day" under "Help".

Installing plugins

As mentioned several times so far, PyCharm and IntelliJ have a tremendous ecosystem of [IDE plugins](#). PyCharm itself has almost 500 plugins, covering many different categories.

In fact, the PyCharm IDE itself is composed of plugins. This is why, for example, PyCharm inherits many web features from WebStorm: they share the same core functionality via plugins.

Via PyCharm's Settings/Preferences dialog, [plugins](#) can be browsed, installed, updated, and removed. Also, plugins are checked for updates along with PyCharm itself. If a plugin is updated, you will be asked to download it and restart PyCharm.

Polyglot Development

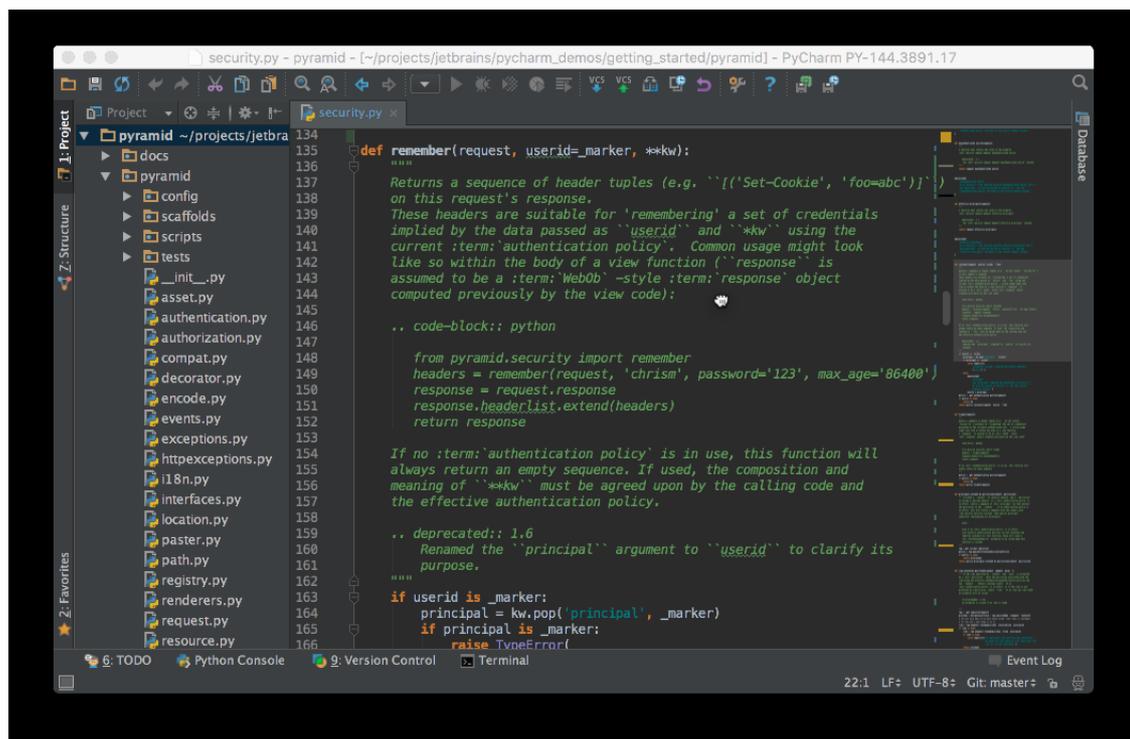
While text editors support many languages alongside Python, PyCharm is unique in the power it brings beyond Python. For example, PyCharm is a first-class web development IDE. It doesn't just syntax-highlight and autocomplete HTML, CSS, and JavaScript. It provides the full spectrum of features: semantically-aware code inspection, refactoring, debugging, launching a live server, translating SASS files to CSS, browsing npm run scripts, etc.

This applies beyond web development as well: IPython Notebook emulation, reStructuredText, and more. How is PyCharm able to do this? As mentioned above, it shares JetBrains plugins with its other IntelliJ-based IDE brethren.

Minimap

Other tools, such as Sublime Text and Visual Studio (via add-on), provide a visual mode of scrolling through a large document. With this, the normal scroll bar is replaced with a graphical thumbnail of the document and your current scroll location in it.

PyCharm users can also get this with the [CodeGlance](#) JetBrains plugin, which is written to conform to the selected theme.



Additional resources

- [Getting Started](#) series of screencasts on YouTube
- PyCharm [home page](#), [blog](#), and [Twitter account](#).
- [Official help system](#), [keymap](#), and [using online resources](#).
- Ask questions in the [community forum](#) or [StackOverflow topic](#).

PyCharm Refactoring Tutorial

On this page:

- [What this tutorial is about](#)
- [Prerequisites](#)
- [Preparing an example](#)
 - [Simplifying rational number](#)
- [Extracting a method](#)
- [Inlining a local variable and changing method signature](#)
- [Using quick fix](#)
- [Moving the function to another file](#)
- [Further changes of the class Rational](#)
 - [Adding magic methods](#)
 - [Extracting method and using a quick fix](#)
- [Extracting a superclass](#)

What this tutorial is about

This tutorial shows some refactorings available in PyCharm, using the example of a simple class that makes use of the rational numbers.

Prerequisites

Make sure that the following prerequisites are met:

- You are working with PyCharm version 2016.2 or later.
- A project is already created.

Preparing an example

Create a Python file `rational.py` in your project and add the following code:

```
from collections import namedtuple

class Rational(namedtuple('Rational', ['num', 'denom'])):

    def __new__(cls, num, denom):
        if denom == 0:
            raise ValueError('Denominator cannot be null')
        if denom < 0:
            num, denom = -num, -denom
        return super().__new__(cls, num, denom)

    def __str__(self):
        return '{}/{}'.format(self.num, self.denom)
```

Simplifying rational number

Let's simplify a rational number by dividing numerator and denominator by the greatest common divisor:

```
from collections import namedtuple

class Rational(namedtuple('Rational', ['num', 'denom'])):
    def __new__(cls, num, denom):
        if denom == 0:
            raise ValueError('Denominator cannot be null')
        if denom < 0:
            num, denom = -num, -denom

        x = abs(num)
        y = abs(denom)
        while x:
            x, y = y % x, x
        factor = y

        return super().__new__(cls, num // factor, denom // factor)

    def __str__(self):
        return '{}/{}'.format(self.num, self.denom)
```

Extracting a method

Now, let's extract the search for a greatest common divisor to a separate method. To do that, select the statements

```
x = abs(num)
y = abs(denom)
while x:
    x, y = y % x, x
factor = y
```

and press `Ctrl+Alt+M`. In the dialog box that opens type the method name (`gcd`) and then click OK:

```
@staticmethod
def gcd(denom, num):
    x = abs(num)
    y = abs(denom)
    while x:
        x, y = y % x, x
    factor = y
    return factor
```

Inlining a local variable and changing method signature

Let's get rid of the variable `factor`, by using [inline variable](#) refactoring. To do that, place the caret at the variable in question and press `Ctrl+Alt+N`. All the detected `factor` variables are inlined.

Next, change the parameter names using [Change Signature](#). To do that, place the caret at the method declaration line and press `Ctrl+F6`. In the dialog box that opens, rename the parameters `denom` and `num` to `x` and `y` respectively, and click `↕` to change the order of parameters.

You end up with the following code:

```
@staticmethod
def gcd(x, y):
    x = abs(x)
    y = abs(y)
    while x:
        x, y = y % x, x
    return y
```

Using quick fix

Now, let's convert the existing static method to a function. To do that, press `Alt+Enter`, from the suggestion list choose Convert static method to function and press `Enter`:

```
from collections import namedtuple

class Rational(namedtuple('Rational', ['num', 'denom'])):
    def __new__(cls, num, denom):
        if denom == 0:
            raise ValueError('Denominator cannot be null')
        if denom < 0:
            num, denom = -num, -denom

        factor = gcd(num, denom)

        return super().__new__(cls, num // factor, denom // factor)

    def __str__(self):
        return '{}/{}'.format(self.num, self.denom)

def gcd(x, y):
    x = abs(x)
    y = abs(y)
    while x:
        x, y = y % x, x
    return y
```

Moving the function to another file

Now, we'll move the function to a separate file and add an import statement. To do that, place the caret at the function `gcd` declaration and press `F6`. In the dialog box that opens specify the fully qualified path of the destination file `util.py`. This file does not exist, but it is created automatically:

```
def gcd(x, y):
    x = abs(x)
    y = abs(y)
    while x:
        x, y = y % x, x
    return y
```

The import statement is also added automatically. Thus the file `rational.py` looks as follows:

```
from collections import namedtuple

from util import gcd

class Rational(namedtuple('Rational', ['num', 'denom'])):
    def __new__(cls, num, denom):
        if denom == 0:
            raise ValueError('Denominator cannot be null')
        if denom < 0:
            num, denom = -num, -denom

        factor = gcd(num, denom)

        return super().__new__(cls, num // factor, denom // factor)

    def __str__(self):
        return '{} / {}'.format(self.num, self.denom)
```

Further changes of the class Rational

Adding magic methods

Next, let us add declarations of the magic methods for the addition/subtraction operations on the objects of the class `Rational`:

```

from collections import namedtuple

from util import gcd

class Rational(namedtuple('Rational', ['num', 'denom'])):
    def __new__(cls, num, denom):
        if denom == 0:
            raise ValueError('Denominator cannot be null')
        factor = gcd(num, denom)
        if denom < 0:
            num, denom = -num, -denom
        return super().__new__(cls, num // factor, denom // factor)

    def __str__(self):
        return '{}/{}'.format(self.num, self.denom)

    def __add__(self, other):
        if isinstance(other, int):
            other = Rational(other, 1)

        if isinstance(other, Rational):
            new_num = self.num * other.denom + other.num * self.denom
            new_denom = self.denom * other.denom
            return Rational(new_num, new_denom)

        return NotImplemented

    def __neg__(self):
        return Rational(-self.num, self.denom)

    def __radd__(self, other):
        return self + other

    def __sub__(self, other):
        return self + (-other)

    def __rsub__(self, other):
        return -self + other

```

Extracting method and using a quick fix

Next, we'll extract an expression `Rational(other, 1)` into a separate method. To do that, place the caret at the aforementioned expression, press

`Ctrl+Alt+M` and in the dialog box that opens, type the new method name `from_int`.

Finally, place the caret at the method `from_int` declaration, press `Alt+Enter`, select Make method static from the suggestion list, and then press

`Enter`:

```

@staticmethod
def from_int(other):
    return Rational(other, 1)

```

Finally, let's change the name of the parameter `other` to `number`. To do that, place the caret on the parameter and press `Shift+F6`.

Extracting a superclass

Next, we'll move the implementations of the methods `__radd__`, `__sub__` and `__rsub__` into a superclass. Also, we'll make the methods `__neg__` and `__add__` abstract.

This is how it's done... Place the caret at the class `Rational` declaration, on the context menu point to Refactor | Extract and choose Superclass.... Next, in the dialog box that opens, specify the name of the superclass (here it's `AdditiveMixin`) and select the methods to be added to the superclass. For the methods `__neg__` and `__add__`, select the checkboxes in the column Make abstract.

End up with the following code:

```
from abc import abstractmethod, ABCMeta
from collections import namedtuple

from util import gcd

class AdditiveMixin(metaclass=ABCMeta):
    @abstractmethod
    def __add__(self, other):
        pass

    @abstractmethod
    def __neg__(self):
        pass

    def __radd__(self, other):
        return self + other

    def __sub__(self, other):
        return self + (-other)

    def __rsub__(self, other):
        return -self + other

class Rational(namedtuple('Rational', ['num', 'denom']), AdditiveMixin):
    def __new__(cls, num, denom):
        if denom == 0:
            raise ValueError('Denominator cannot be null')
        factor = gcd(num, denom)
        if denom < 0:
            num, denom = -num, -denom
        return super().__new__(cls, num // factor, denom // factor)

    def __str__(self):
        return '{}/{}'.format(self.num, self.denom)

    def __add__(self, other):
        if isinstance(other, int):
            other = self.from_int(other)

        if isinstance(other, Rational):
            new_num = self.num * other.denom + other.num * self.denom
            new_denom = self.denom * other.denom
            return Rational(new_num, new_denom)

        return NotImplemented

    def from_int(self, number):
        return Rational(number, 1)

    def __neg__(self):
        return Rational(-self.num, self.denom)
```

Sharing PyCharm Settings

On this page:

- [Why sharing?](#)
- [What this tutorial is about](#)
- [Before you start](#)
- [Preparing an example](#)
 - [Creating copies of the various schemes](#)
 - [Creating a copy of the code style scheme](#)
 - [Creating a copy of a keymap](#)
 - [Creating a copy of the editor color scheme](#)
 - [Where the copies of the setting are stored?](#)
 - [Putting the project under version control](#)
- [Exporting settings](#)
- [Importing settings](#)
- [More information](#)

Why sharing?

It is quite possible that you use one PyCharm license on different machines and on the different operating systems, for example, on your desktop at work and on your laptop at home. Another option is to share settings across a team, so that all the teammates use the same code style, live templates, or file templates. That's why it is vital to be able to share PyCharm settings, which is done by means of export and import.

PyCharm provides Export Settings command that produces the archive `settings.jar`. This archive contains the global settings differing from the defaults, and is stored locally. If you want to share your global settings, you have to make this archive accessible. You can do this in the numerous ways:

- Put this archive under version control
- Place it in your Dropbox
- Just write it to a flash drive and stick into your pocket.
- etc.

As soon as the archive of the global settings becomes accessible, it can be imported into another installation of PyCharm using the Import Settings command.

What this tutorial is about

This tutorial aims to walk you through exporting PyCharm global settings, sharing them via VCS, and importing them to a different PyCharm installation.

Out of scope of this tutorial are:

- Python programming
- Information about the [project and IDE settings](#).
- Information about using specific version control tools, Dropbox etc.

Before you start

Make sure that you are working with PyCharm. This tutorial is created with PyCharm version 2017.1.

Preparing an example

- First, we'll change some settings (for example, code style, keymap, and editor color scheme). This step is required, because we are going to export the archive of global settings, that contains only the differences against the default settings.
- Next, we'll see where the copies of the schemes reside. This step is optional, and is intended just for your information.
- Finally, we'll put the project under version control. This step can be also considered optional, since, as mentioned above, you can make your archive of global settings accessible in numerous ways.

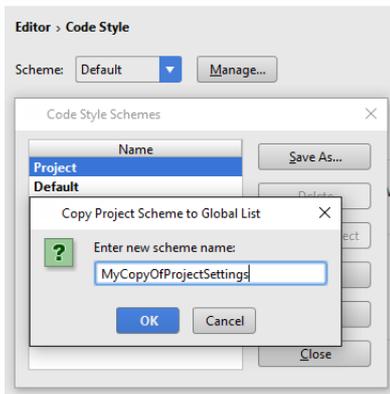
Creating copies of the various schemes

We are going to create copies of the various schemes (remember, the pre-defined schemes are not editable) to further change and share them. Click  on the main toolbar to open the Settings/Preferences dialog.

Creating a copy of the code style scheme

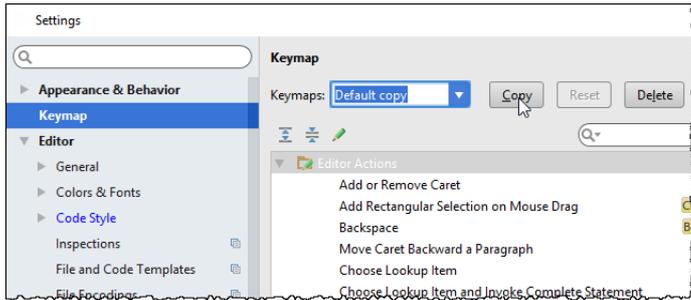
Follow these steps:

1. On the [Code Style](#) page of the editor settings, click Manage.
2. Then, in the Code Style Schemes dialog, select a scheme you want to copy, click the button Save As, and type the new scheme name:



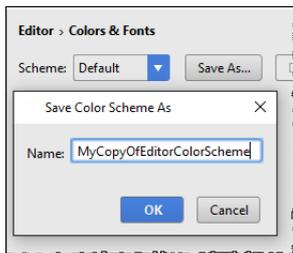
Creating a copy of a keymap

In the [Keymap](#) page of the Settings/Preferences dialog, choose a scheme and click Copy:



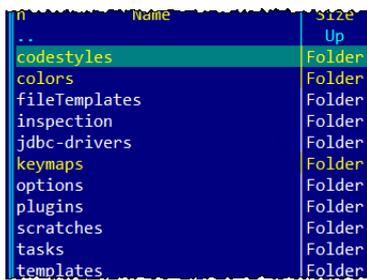
Creating a copy of the editor color scheme

Expand the [Editor](#) node of the Settings/Preferences dialog, in the [Colors and Fonts](#) page, choose the desired scheme, and click the button Save As. Then type the name of the new editor color scheme:

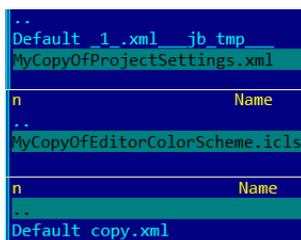


Where the copies of the setting are stored?

As it is mentioned in the page [Project and IDE Settings](#), storage location of the settings depends on your platform. For example, for Windows, the settings are stored in your user home, so it makes sense to look for the copies of the schemes there. Here they are, under the `.PyCharm20XX.X\config` directory:



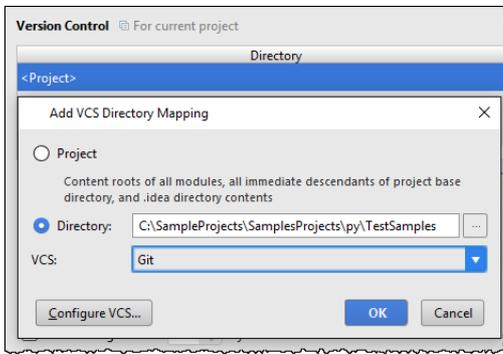
The copies of the new schemes are added to the global settings, and as such, become sharable:



Putting the project under version control

In this example, we use [PyCharm's Git integration](#). It is supposed that you have Git integration enabled.

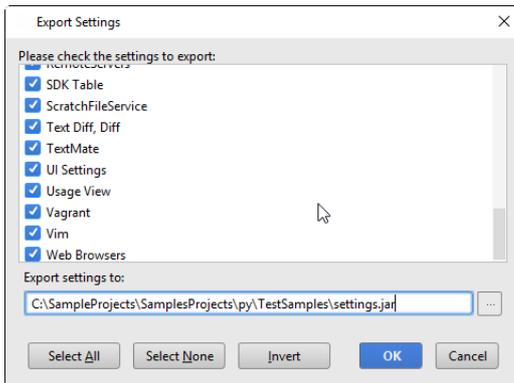
To put the project under version control, click  on the main toolbar to open the Settings/Preferences dialog, then on the [Version Control](#) page click  to add a new content root, and choose Git as the version control system:



Refer to the section [Associating a Directory with a Specific Version Control System](#) .

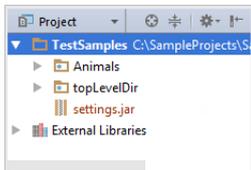
Exporting settings

Let's export the settings. To do that, on the main menu, click File | Export Settings..., and then, in the Export Settings dialog, click the ellipsis to choose the target location. Since we want to use the PyCharm's Git integration, it makes sense to place the exported archive under the project root, to make the archive visible in the [Project Tool Window](#), and make use of the [Version Control Tool Window](#). Next, unselect the check boxes to the left of the unnecessary settings, and then click OK:



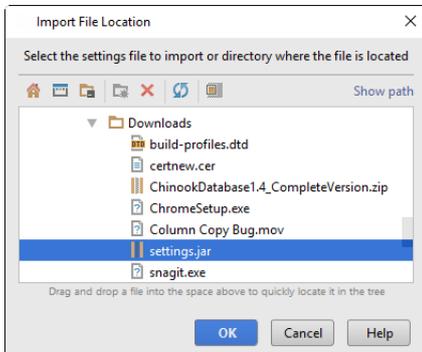
The `settings.jar` archive is created in the specified location. Since we have chosen the project root, we see this archive (marked as unversioned file) in the Project tool window, and in the Version Control tool window.

Actually, export is done... the only thing left is to push the archive to the repository. So first, press `Ctrl+Alt+A` to add the new file to version control, and commit changes:

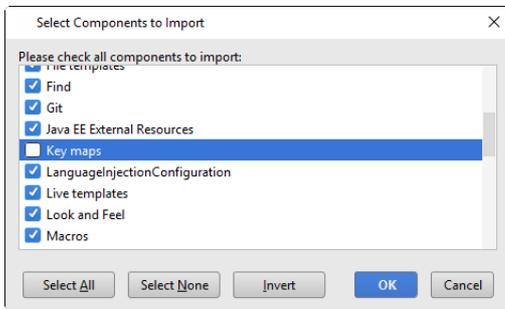


Importing settings

What's next? Let us suppose that your teammates have access to your repository and can download your settings archive. Then one has to choose File | Import Settings... in his/her PyCharm installation, and locate the downloaded archive (or its parent directory):



Next, one has to specify which settings should be imported - one can actually unselect, for example, keymap settings, if he/she is quite happy with his/her keymap:



After clicking OK, PyCharm suggests to restart, for the new schemes to take effect.

More information

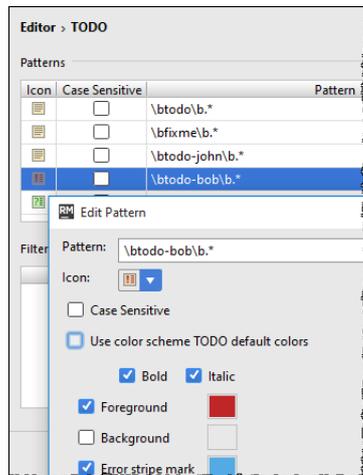
You may find it useful to read PyCharm documentation. Refer to the page [Exporting and Importing Settings](#) .

TODO Example

Consider the following example: creating and viewing TODO items for each of the team members.

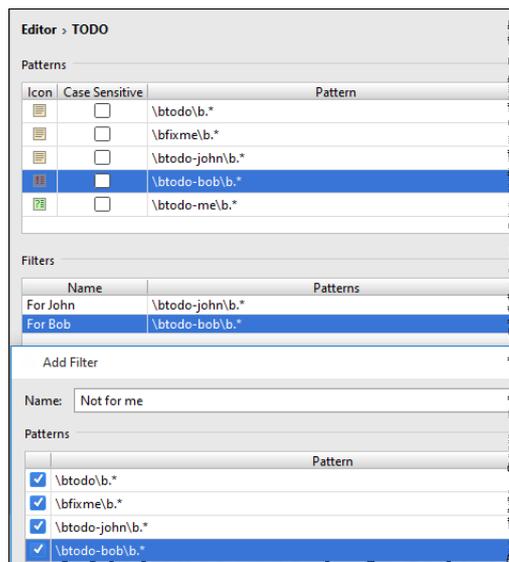
Creating patterns for TODO items

On the **TODO** page of the Settings dialog, click **+** in the Patterns section, and **create additional TODO patterns**, for example, `todo-John`, `todo-Bob` and `todo-me`, with new icons, and custom color schemes:



Creating filters

Next, create several filters, which you will use to show the TODO items, say, for each of the developers, and not for your good self. For this purpose, in the Filters section, click **+**, and specify the filter names, for example, `For John`, `For Bob`, and `not for me`. Associate these filters with the patterns:



Creating TODO items in source code

Now, in the source code, create TODO items: in the line of code, where you want to add a note, press `Ctrl+Slash`, or `Ctrl+Shift+Slash`, and type `TODO` that matches one of the patterns, followed by some meaningful description:

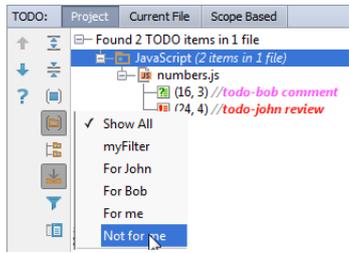
```
//TODO-bob region Description
math = {
  root: Math.sqrt,
  square: square,
  cube: function(x) {
    return x * square(x);
  }
};
race = function() {
  var runners, winner;
  // todo-me have a break
}
//endregion
//TODO-john fix
winner = arguments[0], runners = 2
```

Viewing TODO items

Having produced a number of TODO items across the whole project, review them in the TODO tool window: click the TODO button on the tool window bar to show the tool window. By default, all the encountered TODO items are displayed.

Let's now show the TODO items for Bob and John, and hide the other items: click the filter icon **▼** on the toolbar of the TODO tool window, and select Not for

me in the menu:



Using the Advanced Vagrant Features in PyCharm

On this page:

- [What this tutorial is about](#)
- [Prerequisites](#)
- [Using the built-in SSH terminal to connect to a Vagrant machine](#)
 - [Starting connection](#)
 - [Working with SSH](#)
- [Working with shared folders](#)
 - [Adding a path mapping](#)
 - [Reloading Vagrant](#)
- [Specifying Vagrant instance folder](#)
- [Managing Vagrant plugins through settings](#)
- [Providers support](#)
- [\(Re-\)provisioning a Vagrant machine](#)
- [Working with Environment variables](#)

What this tutorial is about

This tutorial describes how to use the advanced features of Vagrant integration in PyCharm.

The details of using [Vagrant](#) and [Oracle VirtualBox](#) are out of scope of this tutorial. Refer to the respective documentation for more information.

Prerequisites

Pay attention to prerequisites mentioned in the [Vagrant: Working with Reproducible Development Environments](#) page. Also make sure that a Vagrant box is created and initialized.

It is highly recommended to read the [PyCharm documentation](#).

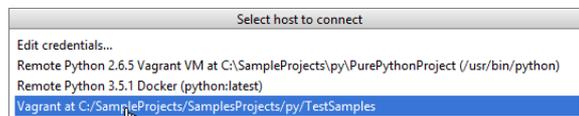
Using the built-in SSH terminal to connect to a Vagrant machine

PyCharm features a [built-in SSH terminal](#) which can be used to connect to a remote machine.

Starting connection

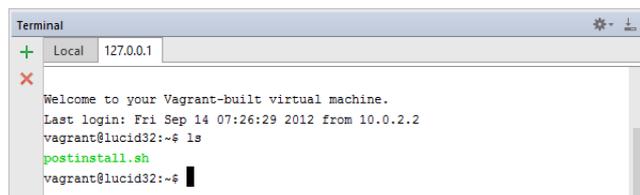
To connect to a Vagrant machine, choose Tools | Start SSH session... on the main menu. This opens a list of hosts we can connect to.

The [configured Vagrant machine](#) is added to this list automatically. Clicking it opens connection to the SSH endpoint exposed by this Vagrant machine. The Edit credentials... menu item allows entering connection information manually. However, we'll confine here to using the virtual box.



Working with SSH

So, after choosing Vagrant, PyCharm connects to the Vagrant machine using SSH server and shows a terminal to work with:



In the SSH terminal you can do the following:

- Scroll through the history of commands using up and down arrow keys
- Perform clipboard operations

Working with shared folders

Vagrant allows sharing folders between the host machine and the Vagrant machine. You can use these folders, for example, to automatically map web root contents from the current PyCharm project to the Apache virtual host directory on the Vagrant machine.

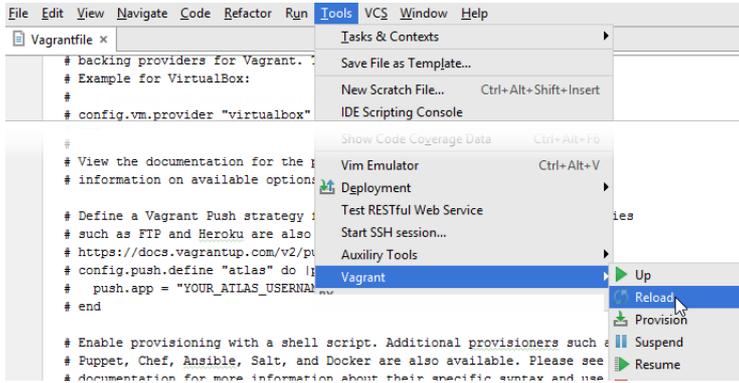
Adding a path mapping

Open `Vagrantfile` for editing (`F4`), and add a configuration entry for path mapping:

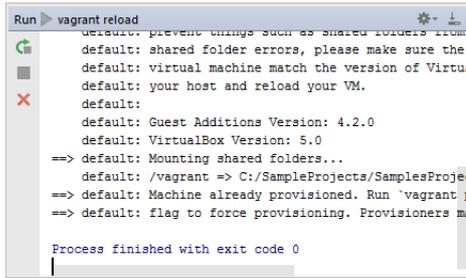
```
Vagrant.configure("2") do |config|
  config.vm.synced_folder "src/", "/srv/website"
end
```

Reloading Vagrant

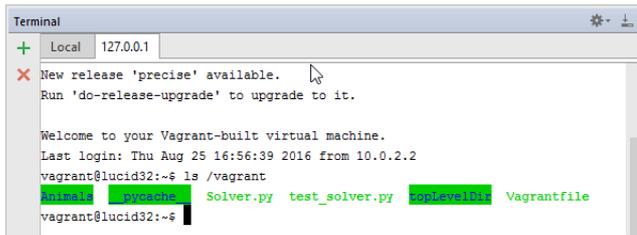
Reload `Vagrantfile` by choosing `Tools | Vagrant | Reload` on the main menu:



The results of command execution show up in the Run tool window:

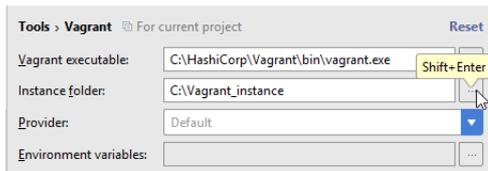


Once the Vagrant machine is reloaded, a new path mapping becomes available. For example, when connecting to the Vagrant machine using the built-in SSH terminal, we see the contents of the `/vagrant` folder that maps to the PyCharm local project folder. Be careful: deleting files from this folder will delete files on both ends!



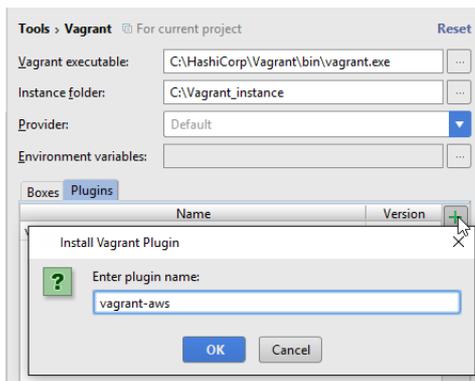
Specifying Vagrant instance folder

By default, the `Vagrantfile` and all other Vagrant specifics (like Puppet manifests) are placed in the root of a PyCharm project. Since this is not always desirable, the instance folder where PyCharm should look for `Vagrantfile` can be configured through the `Vagrant` page of the settings/preferences:



Managing Vagrant plugins through settings

In the `Vagrant` page of the settings/preferences, you can also manage Vagrant plugins. In the tab `Plugins`, use the toolbar buttons to install, uninstall and update plugins.



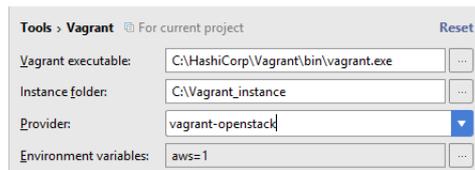
You can also install licenses, for example, for the [VMWare Fusion Provider](#) which allows running Vagrant machines on VMWare.

Providers support

Vagrant works with [Oracle VirtualBox](#) as the virtualization platform by default. However, you can change the virtualization platform using `providers`, so the virtual

machines can be run by a system other than VirtualBox, for example, [VMWare](#) or [Amazon EC2](#). Find available providers at the [Vagrant plugins list](#).

For each command, you should pass the provider to be used to Vagrant. To simplify this process and have PyCharm automatically add the provider name to every Vagrant command, specify the provider in the [Vagrant](#) page of the settings/preferences. All providers installed on your machine, are available from the settings. Once selected, a provider will be used to execute all Vagrant commands in PyCharm:



(Re-)provisioning a Vagrant machine

A [Vagrantfile](#) can contain a series of provisioners which can launch installation and configuration routines once a virtual machine is running.

The Tools | Vagrant | Provision main menu command invokes the configured provisioners on an already running Vagrant machine, without having to first destroy the virtual machine.

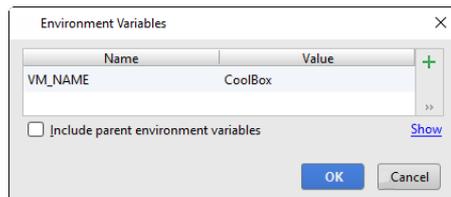
Refer to the [provisioning](#) documentation for details.

Working with Environment variables

Environment variables in [Vagrantfile](#) are useful for:

- the Puppet node
- Puppet environment
- custom facts
- AWS keys
- ...

In the [Vagrant](#) page of the Settings/Preferences dialog box, specify project-specific environment variables to be passed to [Vagrantfile](#) :



Once set, these environment variables are referred to in a [Vagrantfile](#) , using the syntax `#{ENV['name_of_variable']}` :

```
Vagrantfile x
config.vm.provider "virtualbox" do |vb|
  vb.gui = true
  vb.customize ["modifyvm", :id, "--name", "#{ENV['VM_NAME']}"]
end
```

Using Live Templates in TODO Comments

On this page:

- [Overview](#)
- [Creating TODO pattern and filter](#)
- [Creating live template and variables](#)
- [Using the REVIEW items](#)

Overview

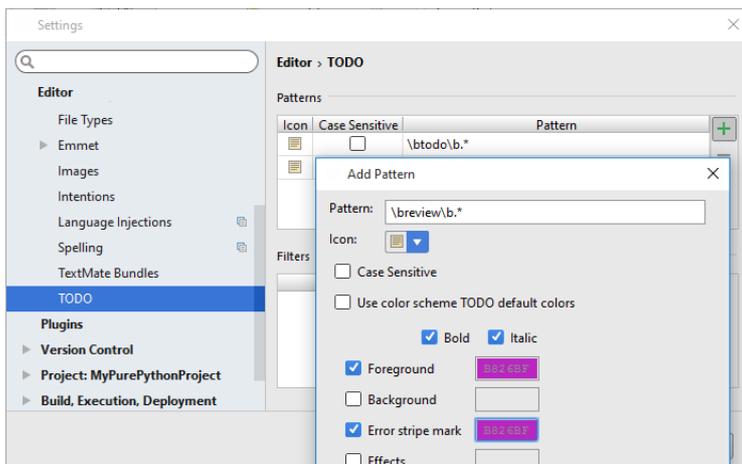
Let's explore an advanced PyCharm's facility to create a [live template](#) for the TODO items' text. Why do we need it at all? For example, you want your team mates to create unified TODO items, with the user name automatically filled in, followed by some arbitrary text.

This is how it's done.

Creating TODO pattern and filter

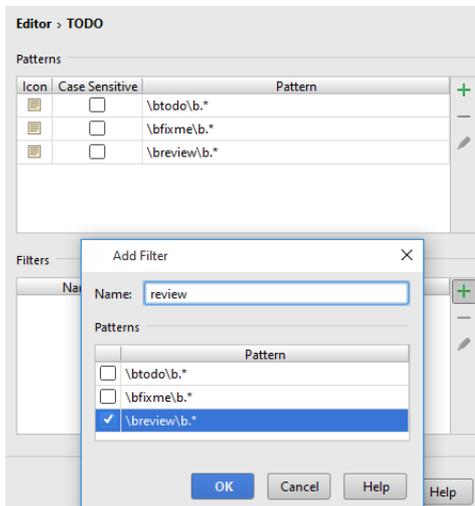
Open the Settings/Preferences dialog, and under the Editor section, click [TODO](#).

[Create pattern](#) `review`. To do that, click **+** in the Patterns section:



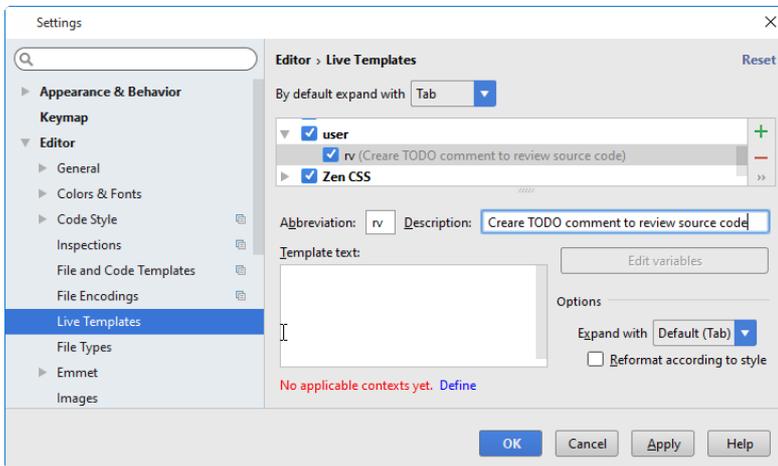
Define color in the Color Picker - in this case, it's pink.

Next, let's [create a filter](#). To do that, click **+** in the Filters section, and define the filter:



Creating live template and variables

Next, back in the Settings/Preferences dialog, under the Editor section, click [Live Templates](#).

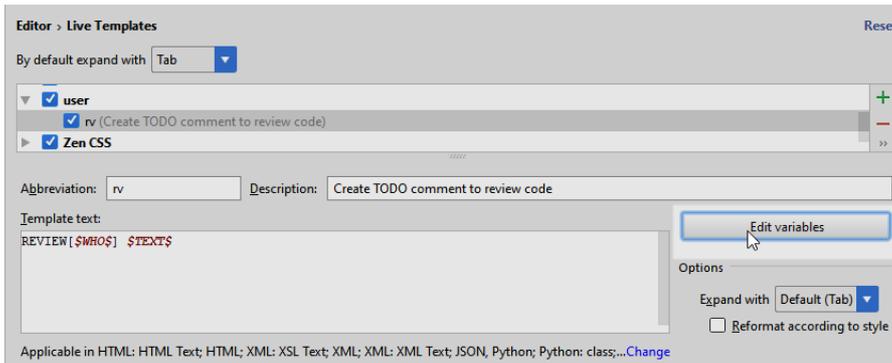


Note that the new template is added to the automatically created group `user`.

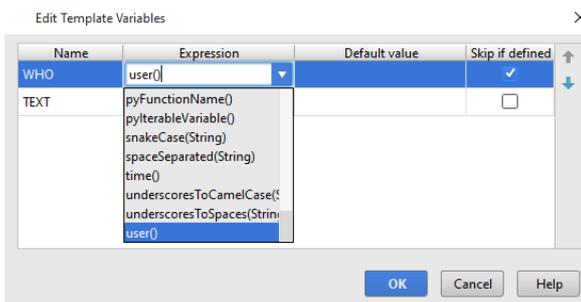
Next, pay attention to the red note at the bottom. It says that the new template lacks context, where it should apply. So let's click the link `Define`, and allow all possible contexts. And finally, let's define the body of the template itself: in the area `Template text`, type the following:

```
REVIEW[$WHO$] $TEXT$
```

We have two undefined variables here: `WHO` and `$TEXT$`. The variable `$TEXT$` will be used just as an input field, while the variable `WHO` should be filled in automatically. To define this variable, click the button `Edit variables`:



Next, in the `Edit Template Variables` dialog box, select an expression for the variable `WHO`:



Using the REVIEW items

Now let's make sure it works. Back in the editor, create a line comment (`Ctrl+Slash`), type `rv`, and press `TAB`:

```
# REVIEW[wombat] REVIEW[wombat]
```

Note that a right-gutter stripe next to the TODO comment is also added to the editor. As you see, the live template `rv` has automatically populated the user name, leaving us with the task of just entering some meaningful comment:

```
# REVIEW[wombat] please make sure that everything works
```

Now, when you opt to show REVIEW comments only, use the filter. To do that, click `▼` and select the filter `review` to show those TODO comments only, that have the keyword REVIEW.

TODO: Project Current File Scope Based Default Changelist

Found 2 TODO items in 2 files

- MyPurePythonProject (2 items in 2 files)
 - Solver.py
 - (16, 19) = REVIEW[wombat] please make sure that everything works
 - test_solver.py
 - (12, 3) = REVIEW[wombat]

Show All
review
Edit Filters

Using IPython/Jupyter Notebook with PyCharm

In this section:

- [Before you start](#)
- [Creating a IPython Notebook file](#)
- [Filling in and running the first cell](#)
- [Working with cells](#)
- [Adding](#)
- [Clipboard operations with the cells](#)
- [Running and stopping kernels](#)
- [Choosing style](#)
- [Writing formulae](#)

Before you start

Prior to executing the tasks of this tutorial, make sure that the following prerequisites are met:

- You have a Python project already [created](#). In this tutorial the project `C:/SampleProjects/py/IPythonNotebookExample` is used.
- In the [Project Interpreter](#) page of the Settings/Preferences dialog, you have:
 - [Created a virtual environment](#). For this tutorial a virtual environment based on Python 3.5.0 has been created.
 - [Installed the following package](#):
 - `ipython`

Note that PyCharm automatically installs the dependencies of these packages.

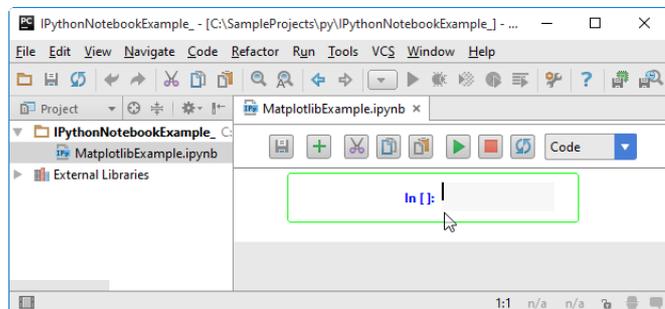
Note All the required packages will be installed automatically, if they are installed via `pip install ipython[notebook]`.

Creating a IPython Notebook file

In the [Project tool window](#), click `Alt+Insert`. Then, on the pop-up menu that appears, choose the option Jupyter Notebook and type the file name (here it is `MatplotlibExample.ipynb`).

The newly created file now shows up in the [Project tool window](#) and automatically [opens for editing](#).

By now, the new file is empty, but PyCharm recognizes it as a [notebook](#) file. As such, this file is marked with the icon  and features a toolbar, which is a complete replica of the real IPython Notebook toolbar:

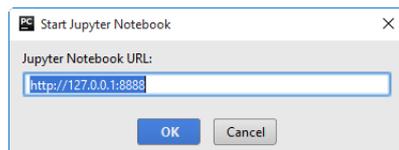


Filling in and running the first cell

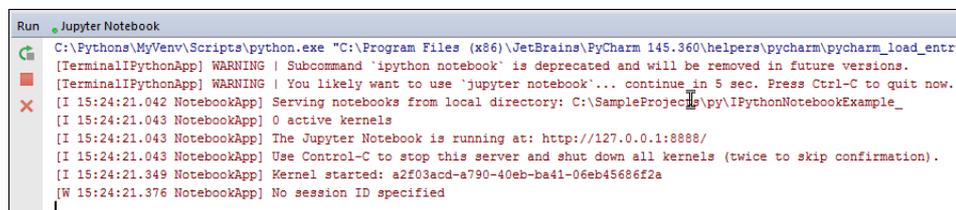
This is most easy. Just click the first cell and start typing. For example, in the very first cell type the following code to configure the `matplotlib` package:

```
%matplotlib inline
```

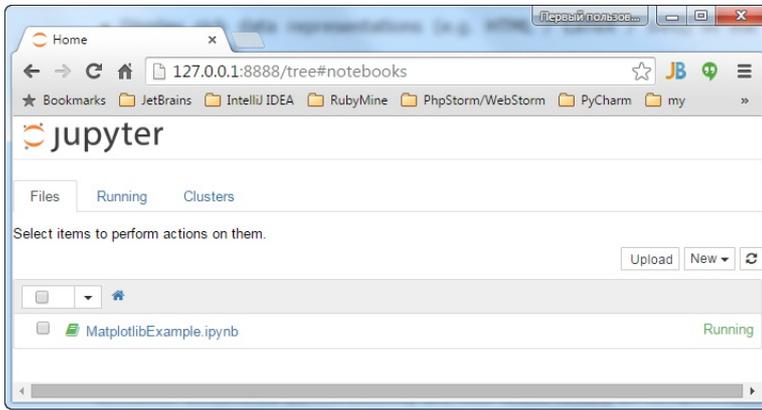
Next, click the  to run the cell (alternatively, you can press `Shift+Enter`). PyCharm shows a dialog box, where you have to specify the URL where the IPython Notebook server will run:



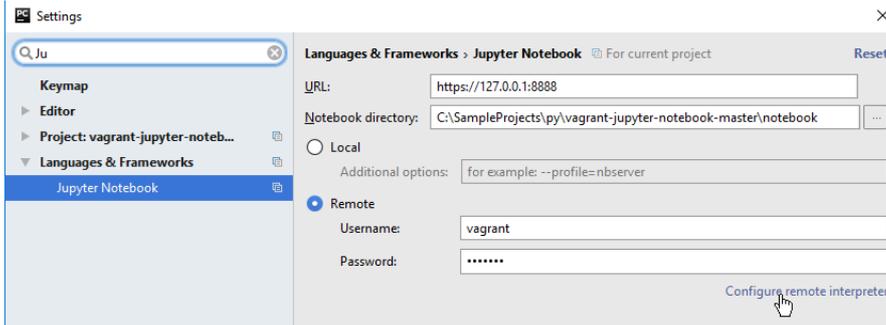
The console shows the server URL:



Follow this address:



Note that the default URL is specified in the IPython Notebook page of the Settings/Preferences dialog:



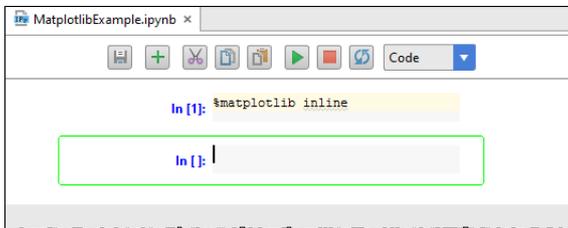
Actually, that's it... From now on you are ready to work with the notebook integration.

Working with cells

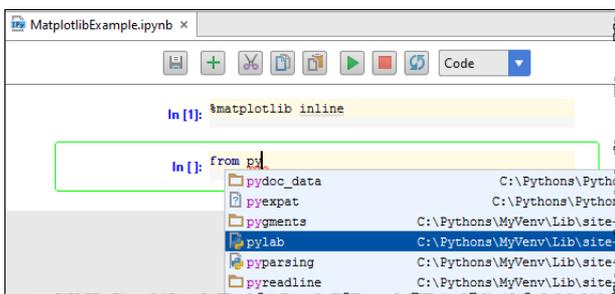
First of all, add the following import statement:

```
from pylab import *
```

This how it's done. As you might have noticed, while you've run the first cell, PyCharm has automatically created the next empty cell:



Start typing in this cell, and notice **code completion**:



Click  on the toolbar again to run this cell. Note that the cell produces no output, but the next empty cell is created automatically. In this new cell, enter the following code:

```
figure()
plot(x, y, 'r')
xlabel('x')
ylabel('y')
title('title')
show()
```

Run this cell. Oops! The attempt to run results in an error:

```
In [4]: figure()
plot(x, y, 'r')
xlabel('x')
ylabel('y')
title('title')
show()
```

```
Out[4]: -----NameError
Traceback (most recent call last):
  File "ipython-input-4-79b2b6e9f2b> in <module>:0
    1 figure()
----> 2 plot(x, y, 'r')
      3 xlabel('x')
      4 ylabel('y')
      5 title('title')
NameError: name 'x' is not defined
```

It seems that the variables should be defined first. To do that, add a new cell.

Adding

Since the new cell is added below the current one, click the cell with import statement - its frame becomes green. Then on the toolbar click **+** (or press

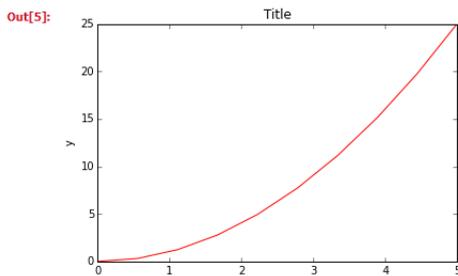
`Alt+Insert`).

In the created cell, type the following code that will define `x` and `y` variables:

```
x = linspace(0, 5, 10)
y = x ** 2
```

Run this cell, and then run the next one. This time it shows the expected output:

```
In [5]: figure()
plot(x, y, 'r')
xlabel('x')
ylabel('y')
title('Title')
show()
```



Clipboard operations with the cells

Look at the toolbar. There are `Ctrl+C`, `Ctrl+X`, and `Ctrl+V` buttons. If you click `Ctrl+X`, you thus delete the current cell, and take it into the clipboard.

Clicking `Ctrl+V` results in inserting the contents of the clipboard below the current cell. Finally, `Ctrl+C` just duplicates the current cell.

Try these buttons yourself.

Running and stopping kernels

As you've already learnt, the `Run` button is used to execute a cell.

If calculation of a certain cell takes too much time, you can always stop it. To do that, click `Stop` on the document toolbar.

Finally, you can rerun the kernel by clicking `Rerun` on the document toolbar.

The messages about all these actions show up in the console:

```
Run: Jupyter Notebook Jupyter Notebook
C:\Python3\MyVenv\Scripts\python.exe "C:\Program Files (x86)\JetBrains\PyCharm 145.380\helpers\pycharm\pyc
[TerminalIPythonApp] WARNING | Subcommand 'ipython notebook' is deprecated and will be removed in future v
[TerminalIPythonApp] WARNING | You likely want to use 'jupyter notebook'... continue in 5 sec. Press Ctrl-
[I 10:59:06.549 NotebookApp] Serving notebooks from local directory: C:\SampleProjects\py\IPythonNotebookE
[I 10:59:06.550 NotebookApp] 0 active kernels
[I 10:59:06.550 NotebookApp] The Jupyter Notebook is running at: http://127.0.0.1:8888/
[I 10:59:06.550 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip co
[I 10:59:07.072 NotebookApp] Kernel started: fba69f8e-2677-48eb-aeb8-5a3ba379328e
[W 10:59:07.079 NotebookApp] No session ID specified
[I 11:01:30.158 NotebookApp] Kernel restarted: fba69f8e-2677-48eb-aeb8-5a3ba379328e
[I 11:01:44.044 NotebookApp] Kernel restarted: fba69f8e-2677-48eb-aeb8-5a3ba379328e
```

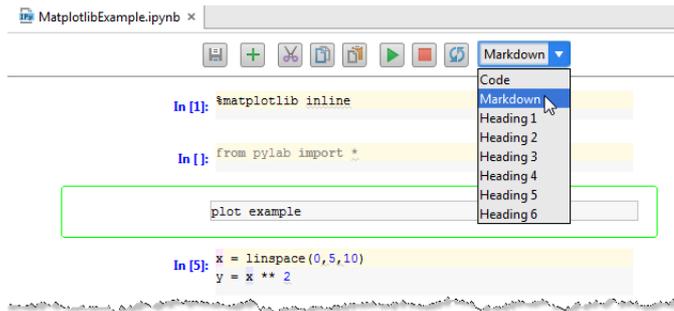
Choosing style

Look at the drop-down list to the right of the document toolbar. It allows you to choose presentation style of a cell. For example, the existing cells are presented as code.

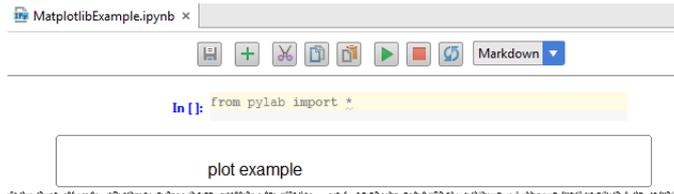
Click the cell with the import statement again, and then click `+`. The new cell appears below. By default, its style selector shows `Code`. In this cell, type the following text:

plot example

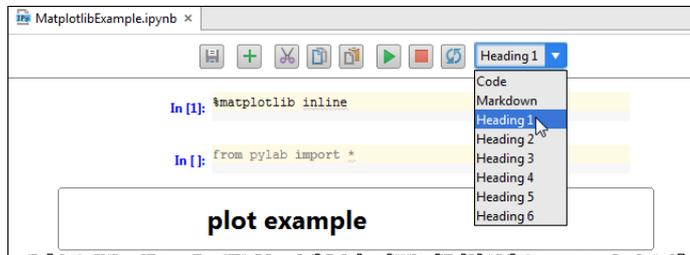
Next, click the down arrow, and choose Markdown from the list. The cell changes its view:



Now click  on the toolbar, and how the cell looks now:



Now you can just select the desired style from the drop-down list, and the view of the cell changes appropriately:

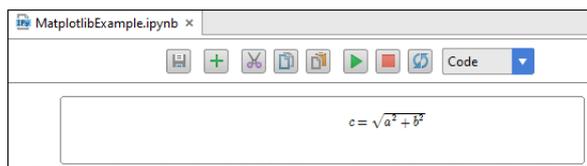


Writing formulae

Add a new cell. In this cell, choose Markdown from the style selector, and type the following text:

```
$$c = \sqrt{a^2 + b^2}$$
```

Click . The result is stunning:



As you see, PyCharm's IPython Notebook integration makes it possible to use [LaTeX notation](#) and render formulae, labels and text.

Next, explore the more complicated case. The expected result - the formula - should appear as the result of calculation. Add a cell and type the following code (taken from [SymPy: Open Source Symbolic Mathematics](#)):

```
from __future__ import division
from IPython.display import display

from sympy.interactive import printing
printing.init_printing(use_latex='mathjax')

import sympy as sym
from sympy import *
x, y, z = symbols("x y z")
k, m, n = symbols("k m n", integer=True)
f, g, h = map(Function, 'fgh')
```

Run this cell. It gives no output. Next, add another cell and type the following:

```
Rational(3,2)*pi + exp(I*x) / (x**2 + y)
```

Click  and enjoy:

MatplotlibExample.ipynb ×

Markdown

```
In [2]: from __future__ import division
from IPython.display import display

from sympy.interactive import printing
printing.init_printing(use_latex='mathjax')

import sympy as sym
from sympy import *
x, y, z = symbols("x y z")
k, m, n = symbols("k m n", integer=True)
f, g, h = map(Function, 'fgh')
```

```
In [3]: Rational(3,2)*pi + exp(I*x) / (x**2 + y)
```

Out[3]:

$$\frac{3\pi}{2} + \frac{e^{ix}}{x^2 + y}$$

Using the PyCharm Built-in SSH Terminal and Remote SSH External Tools

On this page:

- [What this tutorial is about](#)
- [Prerequisites](#)
- [Working with SSH client](#)
 - [Start connection](#)
 - [Provide connection information](#)
 - [Lo and behold!](#)
 - [What can we do in a SSH session?](#)
- [Working with remote SSH external tool](#)
 - [Configuring a SSH external tool](#)
 - [Launching SSH external tool](#)

What this tutorial is about

This tutorial describes how to make use of the PyCharm built-in SSH terminal and remote tools.

SSH basics are out of scope of this tutorial.

Prerequisites

Before you start, make sure that:

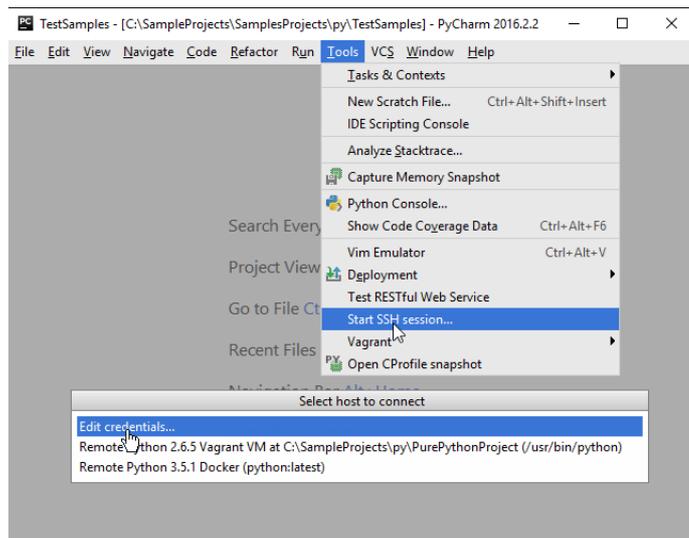
- You are working with PyCharm 3.0 or higher. This tutorial has been created with PyCharm 2016.2
- You have access to a SSH server.

Working with SSH client

Let's see how we can work with the PyCharm's built-in SSH client.

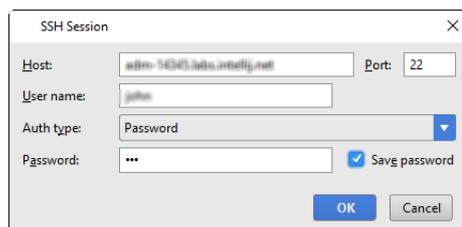
Start connection

On the main menu, choose Tools | Start SSH Session..., and then click Edit credentials:



Provide connection information

In the SSH Session dialog box, specify the connection information: host (local or remote), port, login name and password:



Lo and behold!

After clicking OK, the SSH session starts in the dedicated tab of the Terminal tool window:



What can we do in a SSH session?

As usual for an interactive session... run commands remotely, copy and paste, scroll through the history of commands using up and down arrow keys.

Working with remote SSH external tool

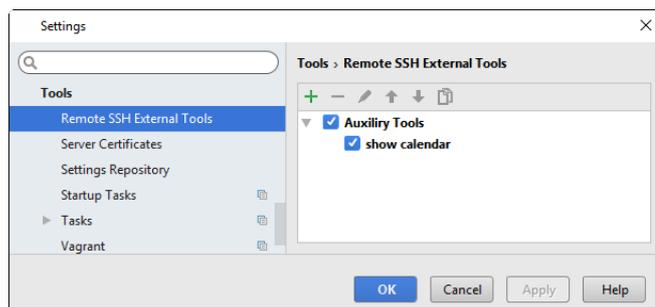
Let's define an external tool that will run a command over SSH, for example, show a calendar.

Configuring a SSH external tool

Open the Settings/Preferences dialog (⚙️ on the main toolbar), and under the Tools node, select the page [Remote SSH External Tools](#). Click + to create a new remote tool, and in the [Create Tool](#) dialog specify the new tool settings:

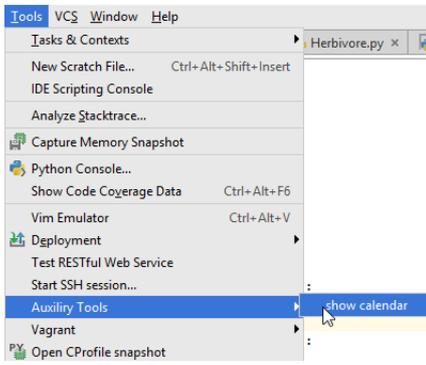
- The field Name helps specifying some visible name of the new tool. The next two fields are optional:
 - In the Description field type some meaningful description.
 - The field Group denotes a group with the specified name that will be created in the Tools menu, and the new SSH external tool will be placed under this group.
- In the Show in area, specify where do you want to see the new tool. In this case, select the check box Main Menu only.
- In the Connection settings area click the radio button Deployment server and choose Select server on every run. It means that every time you want to run this external tool, you will have to specify the connection settings.
- Finally, in the Tool settings area, specify the tool to be executed remotely. In the screenshot below we are running a bash command; parameters and working directory are optional. We can also make use of macros to inject the current command name.

Click OK to close the [Create Tool](#) dialog and return to the page [Remote SSH External Tools](#). We can see the new tool in the list:



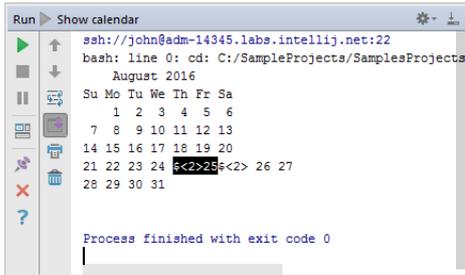
Launching SSH external tool

Once a tool has been set up, it will be shown in the menus selected earlier. In this case, this is the Tools menu that will display the newly created remote SSH external tool under the group, defined in the previous section:



Let's choose this command and see what happens. First of all, you are suggested to choose the server to connect to. Having chosen Edit credentials, specify the connection settings.

And finally, you see the calendar! Here it is:



Using Emacs as an External Editor in PyCharm

On this page:

- [What this tutorial is about](#)
- [Prerequisites](#)
- [Configuring Emacs as an external tool](#)
- [Opening current file in Emacs](#)
- [Assigning a keyboard shortcut](#)

What this tutorial is about

This short tutorial aims to walk you step by step through defining Emacs as an external editor for PyCharm.

Basics of [Emacs](#) are out of scope of this tutorial.

Prerequisites

Make sure that:

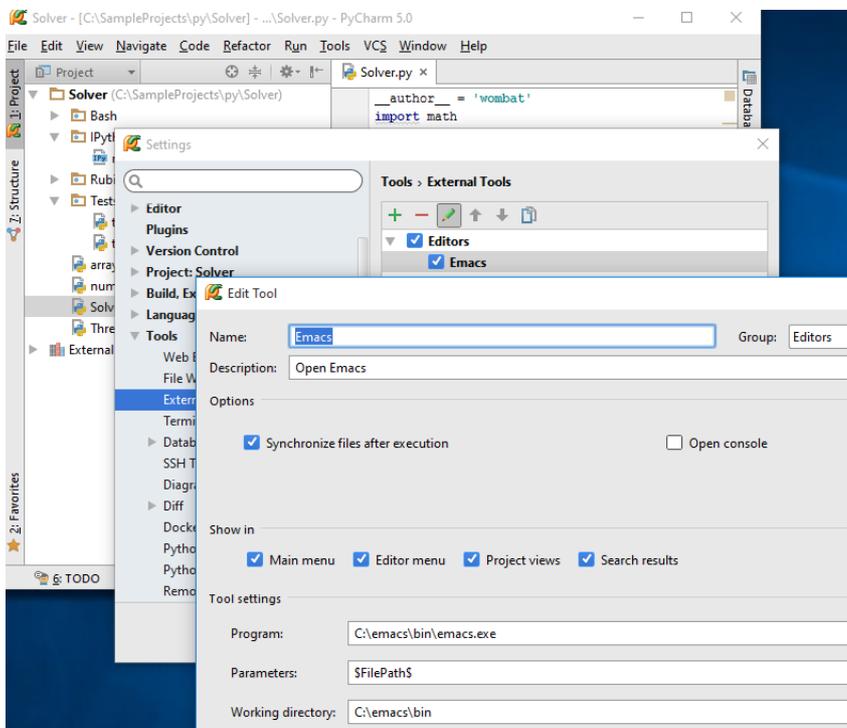
- You are working with PyCharm 2.7 or later. This tutorial is created with PyCharm version 5.0.x.
- Emacs is [downloaded and properly installed](#) on your computer.

Configuring Emacs as an external tool

Open Settings/Preferences dialog. To do that, you can, for example, choose File | Settings (on Windows and *nix) or PyCharm | Preferences (on Mac OS), or click  button on the main toolbar.

Then, under the Tools node, open the page [External Tools](#). On this page, you have to specify your Emacs installation as an external editor for the current file. This is how it's done...

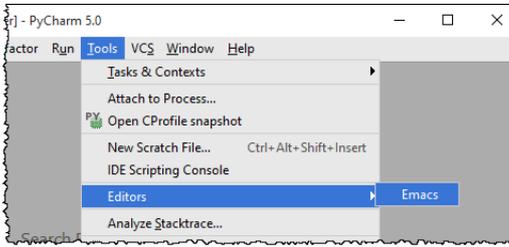
1. First, in the [External Tools](#) page, click . The [Create/Edit Tool](#) dialog box opens.
2. In this dialog, do the following:
 - Type the tool name (Emacs) and optional description (Open Emacs)
 - Specify the name of the group, under which Emacs will appear in the Tools menu. In this example, the group name is Editors. This step is optional - if you specify no group name, then Emacs will appear in the Tools menu as is.
 - Clear the check box Open console.
 - Specify Emacs binary file location. You can either type it manually, or click the ellipsis button and find the desired binary in your file system.
 - Since you want to open the current file in Emacs, pass the file path as a parameter to the program: in the Parameters field, type `$FilePath$`.
 - Finally, specify the working directory - in our example, this is `$ProjectFileDir$`
 - Click OK.



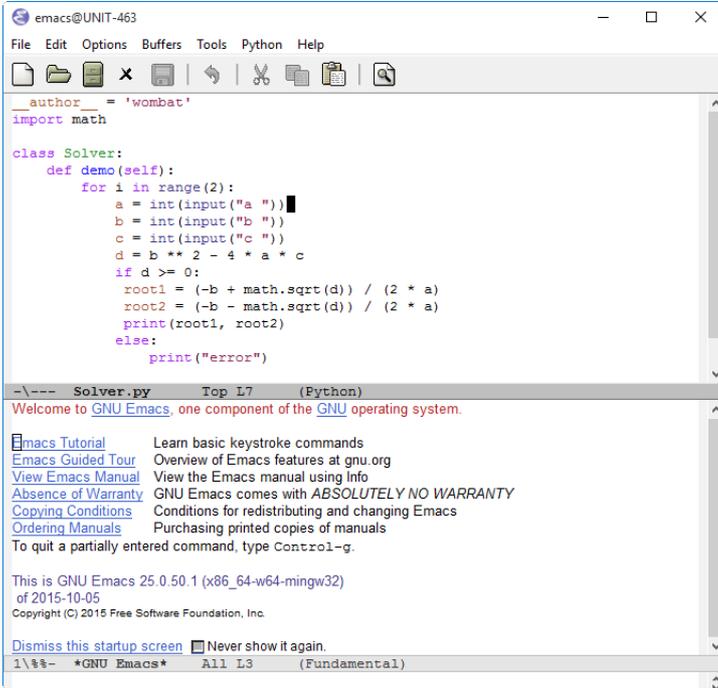
3. Apply changes and close the Settings/Preferences dialog.

Opening current file in Emacs

When you now look at the Tools menu, you will see the new node Editors. Pointing to this node reveals the Emacs command:

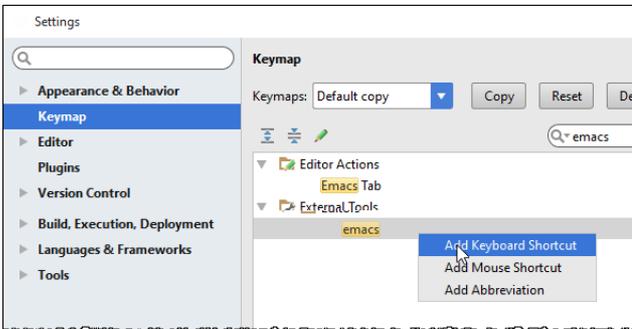


Open a file for editing. Next, on the Tools menu, choose Editors|Emacs - and see the current file in Emacs also:

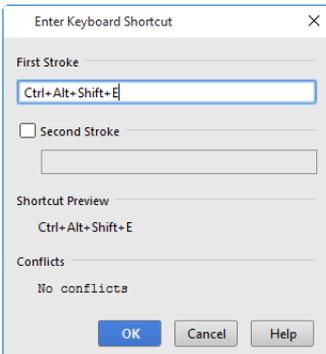


Assigning a keyboard shortcut

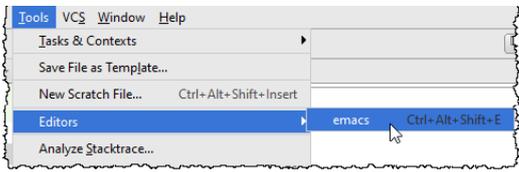
By the way, PyCharm makes it possible to assign a keyboard shortcut to this action: click to open Settings/Preferences dialog, open the [Keymap](#) page, find Emacs, and choose Add Keyboard Shortcut on the context menu:



[Enter Keyboard Shortcut Dialog](#) dialog box opens, where you have to specify, which shortcut you would like this action to be associated with. Let's, for example, use (Ctrl+Shift+Alt+E):



No conflicts are reported, so click OK, and see the new shortcut appearing in the list of actions and on the Tools | Editors | Emacs menu:



Using TextMate Bundles

In this section:

- [What this tutorial is about](#)
- [Prerequisites](#)
- [Importing bundles](#)
- [Extension conflicts](#)
- [Testing](#)

What this tutorial is about

Projects can contain file types unknown to PyCharm. While PyCharm comes with the built-in support for many programming and scripting languages, you might want to have syntax highlighting for the project-specific languages. For example, a project can contain a shell script, or Perl; a configuration file can exist in a project for the infrastructure automation purposes. If you want to have syntax highlighting for these cases, use the powerful PyCharm's integration with the text editor [TextMate](#).

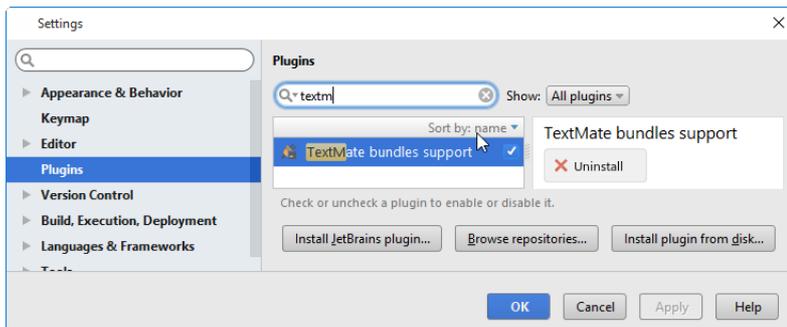
This tutorial aims to walk you step by step through configuring PyCharm to use the TextMate Bundles, and editing files with the registered extensions.

Learning TextMate is out of scope of this tutorial. For information about TextMate, refer to the [product documentation](#).

Prerequisites

Make sure that:

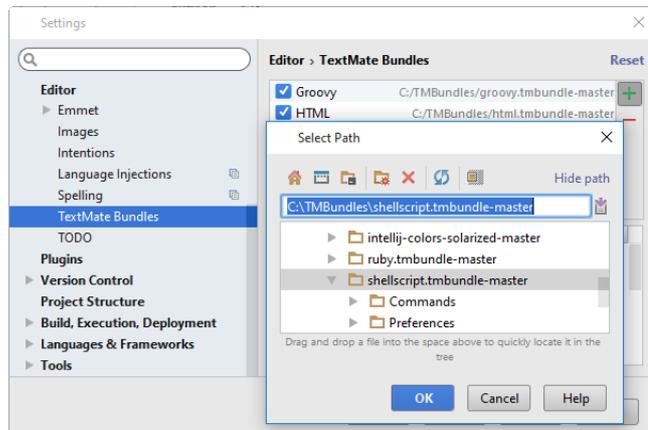
- You have already downloaded bundles you want to use. You can, for example, find the bundles you want to install on [GitHub](#) or [Subversion](#).
- You are working with PyCharm 2.7 (where TextMate Bundles has been supported) or higher. In this tutorial, PyCharm 2016.1 is used.
- Before you start working with TextMate Bundles, make sure that the TextMate bundle support plugin is enabled. The plugin is bundled with PyCharm and is activated by default. If the plugin is not activated, enable it on the [Plugins](#) page of the [Settings / Preferences Dialog](#) as described in [Enabling and Disabling Plugins](#).



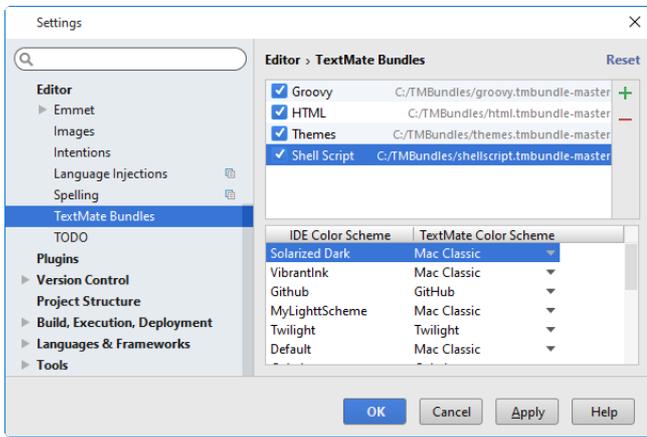
Importing bundles

Suppose you want PyCharm to highlight syntax of the Shell Script files. For this purpose, you have already downloaded the [Shell Script TextMate Bundle](#). It now resides on your hard disk, and you only have to import this bundle into PyCharm.

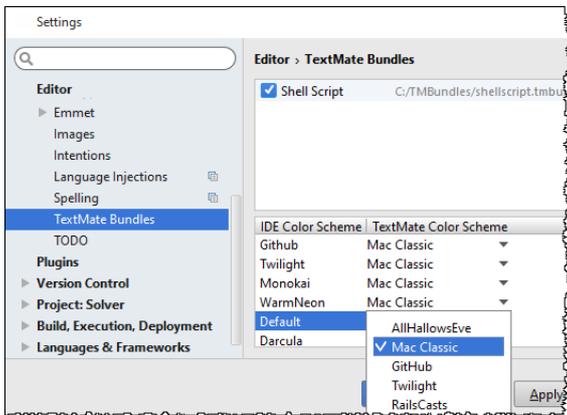
OK, off we go. On the main toolbar, click , and under the Editor node, click [TextMate Bundles](#). Then, in the TextMate Bundles area, click , and locate the desired bundle on your hard disk:



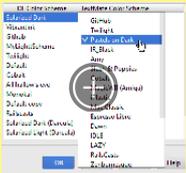
Click OK to apply changes. The Shell Script bundle appears in the list of recognized bundles, and its local path is visible to the right:



However, we have not yet defined which color scheme PyCharm will use to highlight Shell Script syntax. As you already know, PyCharm provides a number of color schemes, from the classic-looking ones to the fashionable dark schemes, like Darcula. Have a look at the color scheme mapping section in the lower part of the TextMate Bundles page. By default, the PyCharm's Default color scheme maps to the Mac Classic. If we want to use a different color scheme for our bundle, we can click the "Mac Classic" cell in the table of mappings, and select the desired scheme from the list of available ones. However, let's stick to the suggested scheme:

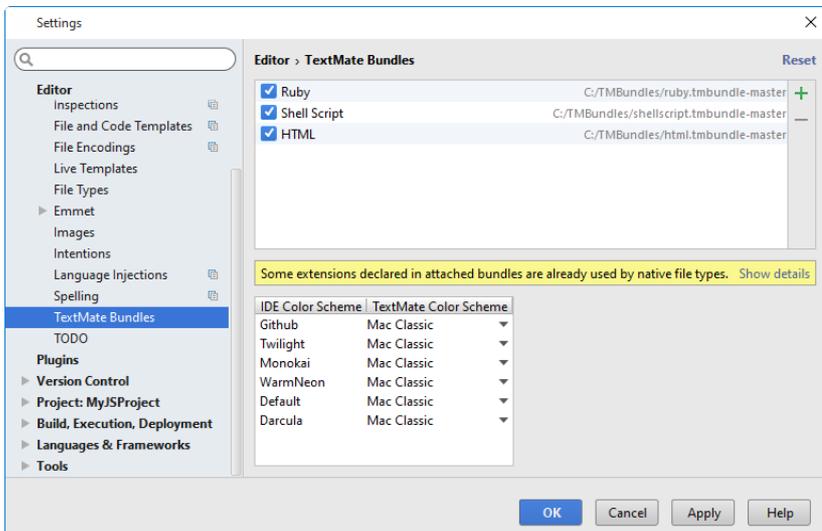


Tip If you want to use a custom TextMate color scheme, you can import a TextMate bundle with schemes, and it will become visible in the list of TextMate color schemes after clicking Apply.

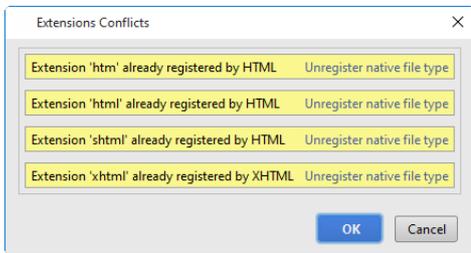


Extension conflicts

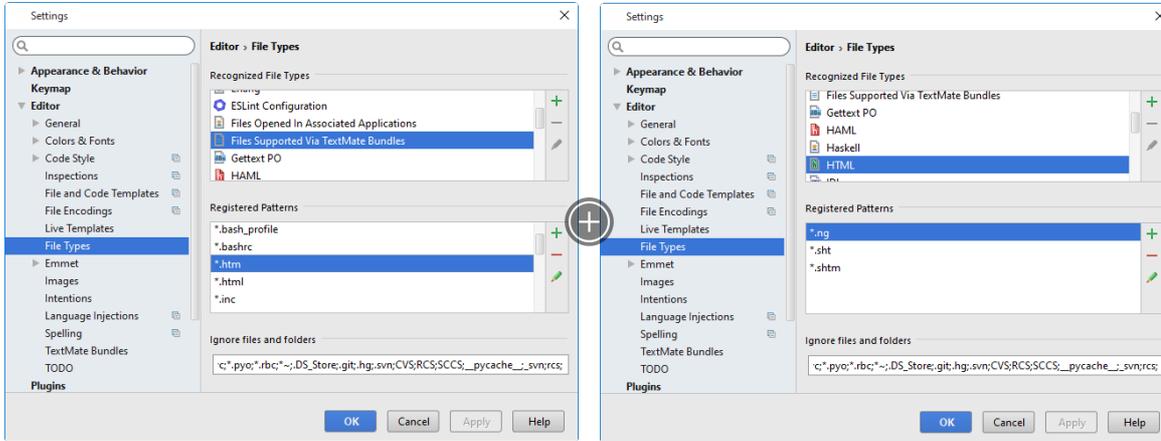
Suppose you've imported a bundle that runs into a conflict with the existing file types. The conflict is immediately reported:



Clicking the Show details link opens the dialog box that gives you the chance to unregister the required extensions from the native file types:



The node Files supported via TextMate Bundles now shows the new extensions (`.htm`, `.html`), and the node HTML lacks these extensions:



Testing

Once a TextMate bundle is added, PyCharm provides syntax highlighting for the file types registered with the bundle. Here's a sample script that uses the Shell Script TextMate bundle we've cloned earlier:

```
run.bash x
#!/bin/sh
echo "This script checks the existence of..."
echo "Checking..."
if [ -f var/log/messages ]
then
    echo "var/log/messages exist"
echo
echo "done"
fi
```

Using Vim Editor Emulation in PyCharm

On this page:

- [Before you start](#)
- [Downloading and installing IdeaVim plugin](#)
- [What happens to PyCharm's UI after restart?](#)
- [Configuring shortcuts](#)
- [Editing modes](#)

Before you start

Make sure that:

- You are working with PyCharm version 4.0.0 or higher. If you still do not have PyCharm, download it from [this page](#). To install PyCharm, follow the instructions, depending on your platform.

This tutorial is created with PyCharm version 2016.1.

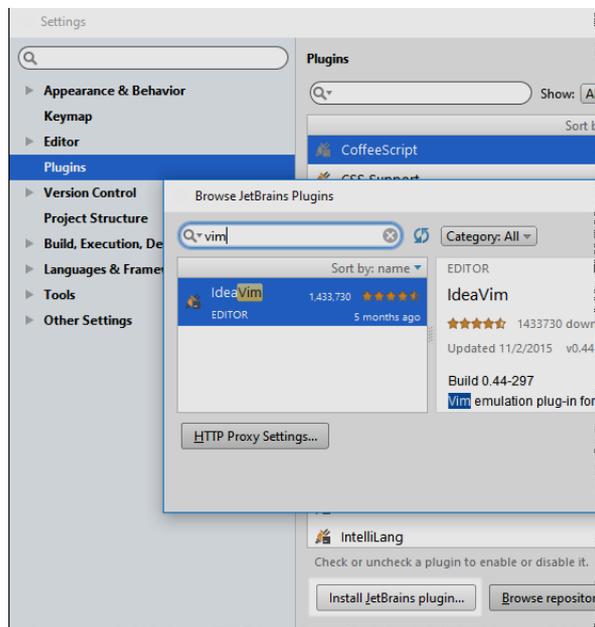
- You have at least one Python interpreter properly installed on your computer. You can download an interpreter from [this page](#).

Downloading and installing IdeaVim plugin

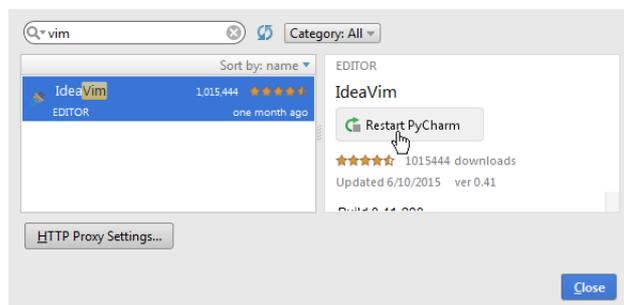
On the toolbar of the PyCharm main window, press `Ctrl+Alt+S` to open the [Settings/Preferences](#) dialog, and then click [Plugins](#).

You see the list of plugins currently installed on you computer. However, the IdeaVim plugin is not among them. Click the button Browse JetBrains plugins.

PyCharm shows the contents of the huge JetBrains repository... you can type the word "vim" in the search field to narrow down the list:

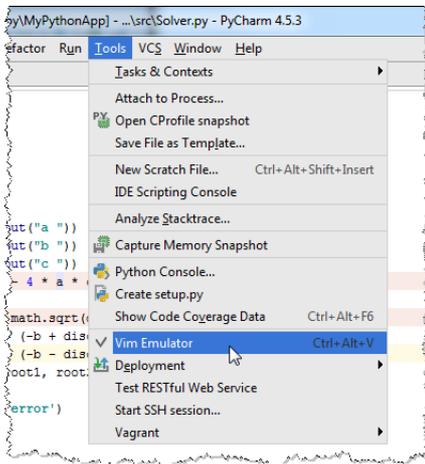


After installing the plugin, it actually becomes available after PyCharm restart.



What happens to PyCharm's UI after restart?

First, on the Tools menu, a check command Vim Emulator appears:



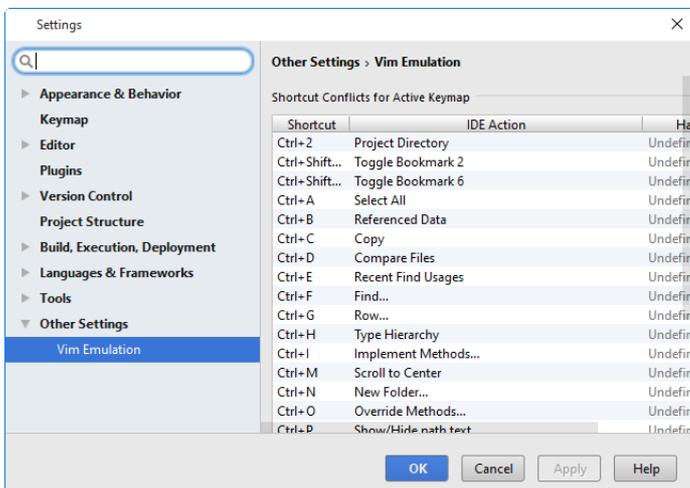
After PyCharm restart, this check command is selected. You can disable Vim by clearing this check command.

Second, in Settings/Preferences dialog, an additional node Other Settings appears, with the page Vim Emulation. This page appears after restart!

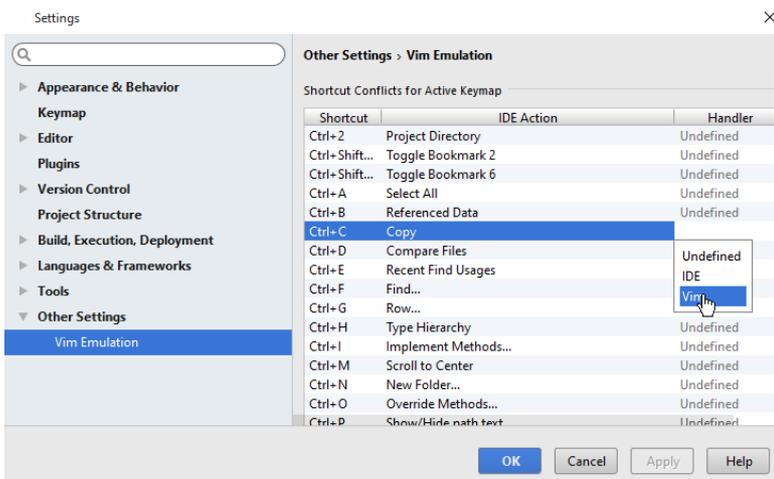
Configuring shortcuts

Both Vim and PyCharm are keyboard-centric. With **IdeaVim plugin**, it is quite possible that PyCharm's keymap runs into a conflict with the Vim keymap. That's why PyCharm allows you choosing which keyboard shortcut you prefer for a certain action. This is how it's done.

Open Settings/Preferences dialog, and under the node Other Settings, click Vim Emulation:

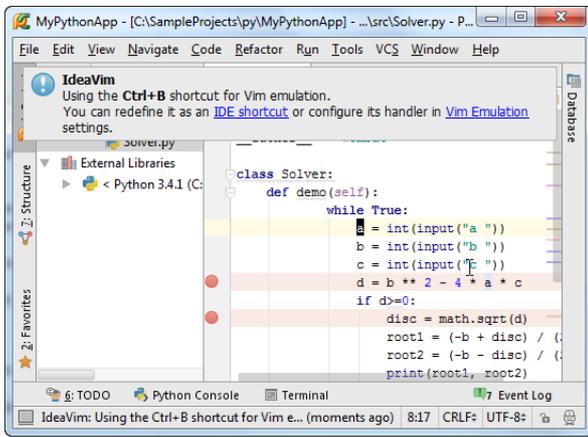


In the Shortcut column, select the shortcut you want to configure. Next, in the Handler column, click the corresponding cell, and see the drop-down list of three possible options (Undefined, Vim, IDE):



If you choose **IDE**, it means that the PyCharm's shortcut for this particular action is enabled. When you press, say, **Ctrl+Z**, PyCharm silently performs its action.

If you leave the handler undefined, then, on pressing the shortcut, say, **Ctrl+B**, PyCharm shows the following banner:



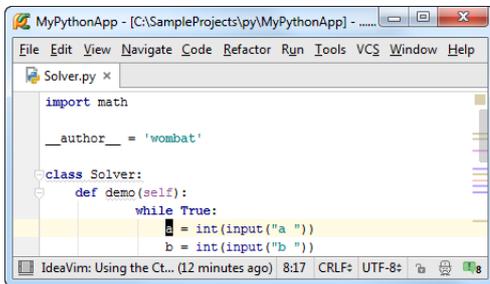
You can choose to redefine this shortcut as an IDE shortcut and thus accept the PyCharm's keymap. To do so, click the link [IDE shortcut](#).

If you click the link [Vim Emulation](#), then PyCharm will show the Vim Emulation page of the Settings/Preferences dialog.

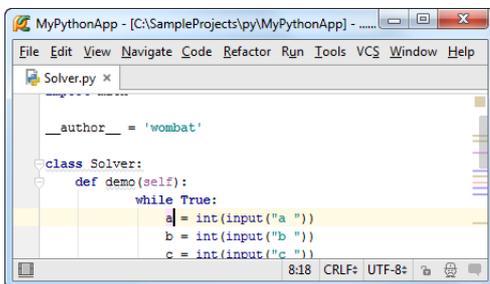
For the purposes of this tutorial, click the link [Vim Emulation](#). Then, when you press `Ctrl+B`, PyCharm will perform the Vim action for this keyboard shortcut.

Editing modes

OK, now that you have Vim enabled, you see that the cursor has changed its shape - now it is a block, which means that you are in the [Normal mode](#) :

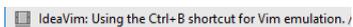


If you want to enter the [Insert mode](#), press `i`, and the cursor will turn into a line:



In this mode you can type new or change the existing code. Same way, you can enter the various Vim modes: for example, press `r` for the [Replace mode](#).

By the way, as soon as you enter Vim emulation, it is also reported in the Status bar.



To return to the Normal mode, press `Escape`.

Reference

This part contains miscellaneous information related to the UI of the dialogs, tool windows and views, keyboard shortcuts, syntax references etc.:

- [Essentials](#)
- [Dialogs](#)
- [Settings / Preferences Dialog](#)
- [Keyboard Shortcuts and Mouse Reference](#)
- [Tool Windows Reference](#)
- [Version Control Reference](#)
- [Diagram Reference](#)
- [Icons Reference](#)
- [Regular Expression Syntax Reference](#)
- [Copy and Paste Between PyCharm and Explorer/Finder](#)
- [Scope Language Syntax Reference](#)
- [Project and IDE Settings](#)
- [Directories Used by PyCharm to Store Settings, Caches, Plugins and Logs](#)
- [Synchronizing and Sharing Settings](#)
- [Networking in PyCharm](#)
- [Tuning PyCharm](#)
- [Index of Menu Items](#)
- [Color-Deficiency Adjustment](#)
- [Working with PyCharm Features from Command Line](#)

Essentials

PyCharm is an advanced IDE. To get the most out of its capabilities and features, you should be familiar with its concepts. Concepts describe the basic notions of the IDE.

In this part:

- [Project](#)
- [Content Root](#)
- [Scope](#)
- [Supported Languages](#)
- [Encoding](#)
- [Path Variables](#)

Project

On this page:

- [Basics](#)
- [Project files](#)
- [Project types](#)

Basics

Whatever you do in PyCharm, you do that in the context of a **project**. A project is an organizational unit that represents a complete software solution. It serves as a basis for coding assistance, bulk refactoring, coding style consistency, etc.

Project files

A project in PyCharm is represented in the **Directory Based Format**. A project directory is marked with  icon.

Such project directory contains the `.idea` directory, with the following files:

- `*.iml` file that describes the project structure.
- `workspace.xml` file that contains your workspace preferences.
- A number of `.xml` files. Each `.xml` file is responsible for its own set of settings, that can be recognized by its name: `projectCodeStyle.xml`, `encodings.xml`, `vcs.xml` etc.

Thus, for example, adding a new run/debug configuration and changing encoding will affect two different `.xml` files. This helps avoid merge conflicts when the project settings are stored in a version control system and modified by the different team members.

All the settings files in the `.idea` directory should be **put under version control** except the `workspace.xml`, which stores your local preferences. The `workspace.xml` file should be **marked as ignored by VCS**.

`.idea` directory is not visible in the Project view of the [Project tool window](#).

Project types

The directory structure of each project contains the `.idea` directory for the PyCharm-specific settings and the project file, and libraries.

PyCharm suggests the following types of projects:

- **Empty project** is intended for pure Python programming. The directory structure of such project contains the `.idea` directory for the PyCharm-specific settings and the project file, and libraries.

Create a plain Python project as described in the section [Creating Empty Project](#).

Warning! [Python](#) must be installed on your machine.

- **Django project**. This project type provides specific infrastructure of the [Django](#) applications, and all the necessary files and settings.

Create a Django application as described in the section [Creating Django Project](#).

Warning! [Python](#) must be installed on your machine.

- Working with Django applications requires a database. Using SQLite is preferred, since it is pre-configured. If you are using a different database engine, make sure it is installed and configured properly.

- **Google App Engine project**. This project type provides specific infrastructure of the Google App Engine application, and all the necessary files and settings.

Create project as described in the section [Creating Google App Engine Project](#).

Warning! [Python](#) and [Google App Engine SDK](#) must be installed on your machine.

- **Flask project**. This project type provides specific infrastructure of a Flask application, and all the necessary files and settings.

Create project as described in the section [Creating Flask Project](#).

- **Web2Py project**. This project type provides specific infrastructure of a [Web2Py](#) application, and all the necessary files and settings.

Create project as described in the section [Creating Web2Py Project](#).

- **Pyramid project**. This project type provides specific infrastructure of a [Pyramid](#) application, and all the necessary files and settings.

Create project as described in the section [Creating Pyramid Project](#).

- **Client-side projects** [HTML5 Boilerplate](#), [Twitter Bootstrap](#), and [Foundation](#).

For the client-side applications, PyCharm creates specific infrastructure, with the required files and directories.

Create project as described in the section [Creating Projects from Scratch in PyCharm](#).

Content Root

On this page:

- [Basics](#)
- [Content root types](#)

Basics

In the terms of PyCharm, **content** is a collection of files with which you are currently working, possibly organized in a hierarchy of subfolders. The folder which is the highest in this hierarchy is called **content root folder** or **content root** (shown as ) for short. A project has at least one content root folder, by default it is the project folder itself.

Having several content roots enables you to work with files from several directories that do not have a common immediate parent. This is helpful when you use some static contents, for example, icons. You can just save them all in a folder and then specify this folder as an extra content root in several projects.

Content root types

By default, all the files in a content root folder are involved in indexing, searching, parsing, code completion, etc. To change this status, folders within a content root can be assigned to the following categories:

- Regular **content roots**, created as described in the section [Configuring Content Roots](#). These roots are marked .

A content root is a folder that contains the files that make up your project.

- **Source roots** (or source folders; shown as .

These roots contain the actual source files and resources. PyCharm uses the source roots as the starting point for resolving imports.

The files under the source roots are interpreted according to their type. PyCharm can parse, inspect, index, and compile the contents of these roots.

- **Resource roots** (or resource folders; marked as .

These roots are intended for resource files in your application (images, style sheets, etc.) By assigning a folder to this category, you tell PyCharm that files in it and in its subfolders can be referenced relative to this folder instead of specifying full paths to them.

- **Excluded roots** (shown as ) are ones that PyCharm "almost ignores".

These roots contain files and folders ignored by PyCharm when indexing, searching, parsing, watching etc.

Excluded roots are not visible to PyCharm. Usually, one would like to exclude temporary build folders, generated output, logs, and other project output.

Excluding the unnecessary paths is a good way to significantly improve performance.

- **Template roots** marked as  contain templates for the various web projects.

Scope

On this page:

- [Basics](#)
- [Types of scopes](#)
- [Defining scopes](#)
- [Scopes coloring](#)
- [Predefined scopes](#)

Basics

A **scope** is a subset of files, packages and/or directories in your project, to which you can limit the application of specific operations, e.g. [search](#), [code inspection](#), etc. Besides, you can configure [coloring](#) for each scope to see at once what sort of file you are dealing with.

Scopes get more helpful as your project grows larger. There are a lot of [predefined scopes](#) that cover basic cases. Additionally, it is possible to [add custom scopes](#) to your project. For example, you can create custom scopes for tests or for the files you are responsible for in your team.

Types of scopes

Scopes can be either shared or local:

- **Shared scopes** are accessible for the team members via VCS and are stored at the project level. In the `scopes` directory under `.idea`, as a file with the extension `.xml`, i.e. `.idea/scopes/<scope_name>.xml`.
- **Local scopes** are intended for personal use only and are stored in your workspace in the file `workspace.xml` under `.idea`.

If necessary, you can share a local scope, make a shared scope local, or create a copy of the scope. For more information, see [Configuring Scopes and File Colors](#).

Defining scopes

PyCharm provides a special language that enables you to flexibly define the sets of entities included in a scope. See [Scope Language Syntax Reference](#) for details.

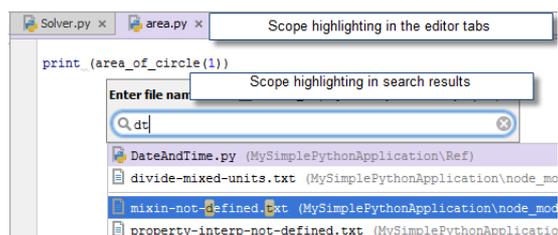
To create and edit **scopes**, use the [Scopes](#) page of the Settings dialog. **Scopes** are defined in the following modes:

- Manually, by specifying file masks according to the [scope language syntax](#) in the Pattern text box.
 - By selecting files and folders and clicking the buttons Include, Include Recursively, Exclude, and Exclude Recursively. Based on the inclusion/exclusion, PyCharm creates an expression and displays it in the Pattern.
- Refer to the section [Configuring Scopes and File Colors](#).

You can explore the available scopes in the [Project tool window](#).

Scopes coloring

Files belonging to different scopes can be highlighted in different colors throughout the PyCharm's user interface: in [navigation lists](#), in the editor tabs, in the [Project window](#). This allows much faster and easier navigation in large projects.



If some file is included into several scopes, the order of the scopes becomes important: PyCharm uses the color of the uppermost scope (shown in the [Scopes](#) settings page) to highlight such file. Of course, you can change the order of the scopes, and thus the resulted highlighting.

For detailed instructions on how to configure the scope order and scope-color associations, see [Configuring Scopes and File Colors](#).

Predefined scopes

PyCharm provides a number of predefined scopes, for example:

- Project Files. This scope includes all the files within the project content roots (see [Content Root](#) and [Configuring Content Roots](#)). Libraries and SDKs are, generally, are not included in this scope.
- Problems. This scope includes the files within the project content roots in which syntactic errors are found.
- Project and Libraries. This scope includes all the files within the project content roots, and also all libraries and SDKs. In the [Project tool window](#), this scope corresponds to the scope view All.
- Project Production Files. This scope is similar to the [Project Files scope](#). The difference is that the test source roots are not included. In the [Project tool window](#), this scope corresponds to the scope view Production.
- Project Test Files. This scope is limited to the project test source roots. In the [Project tool window](#), this scope corresponds to the scope view Tests.
- Non-Project Files. This scope is available only as a view in the [Project tool window](#). It is limited to libraries and SDKs.
- Changed Files. This scope corresponds to all changed files, that is, ones associated with all existing [changelists](#).
- Default. This scope corresponds to the files associated with the [changelist](#) `Default`.
- Favorite '`<name>`'. This scope corresponds to a list of favorite items with the specified name. See [Managing Your Project Favorites](#).

- Open files. This scope corresponds to the files opened in PyCharm editor.
- Current file. This scope corresponds to the file currently active in PyCharm editor.
- Selected files. This scope corresponds to the files currently selected in PyCharm (e.g. in the [Project tool window](#)).

Predefined scopes cannot be edited.

Supported Languages

In this section:

- [Supported languages](#)
- [Coding assistance](#)

Supported languages

Development of a modern application involves using multiple languages, that is why PyCharm is an IDE for polyglot programming. With the deep understanding of all the subtleties of the source code structure and syntax, PyCharm extends its support to:

- [Python](#)
 - Python should be [downloaded and installed](#) on your computer.
 - PyCharm supports Python versions 2.4 to 3.6.
 - PyCharm supports [wxPython](#), [PyQt4](#) and [PyGTK](#).
- [JavaScript](#). Refer to the section [JavaScript-Specific Guidelines](#).
- [CoffeeScript](#). Refer to the section [CoffeeScript Support](#).
- Markup languages and style sheets:
 - [XML](#)
 - [HTML/ XHTML](#)
 - [YAML](#)
 - [CSS](#)
 - [Sass/ Less](#)
 - [Stylus](#)

Refer to [Markup Languages and Style Sheets](#).

PyCharm supports the following frameworks, tools, and template languages:

- [Django](#) up to the version 1.3.
- [Google App Engine](#).
- [Flask](#).
- [Buildout](#).
PyCharm detects `buildout` support in the existing projects, and makes it possible to [open](#) them. However, you have to provide the buildout environment in advance.
- [Web2Py](#). Refer to the section [Web2Py](#) for details.
- [Pyramid](#). Refer to the section [Pyramid](#) for details.
- [Maya](#). This support includes remote debugging and launching scripts in Maya.
- Template languages:
 - [Mako](#)
 - [Jinja2](#)
 - [Chameleon](#)

Refer to the section [Templates](#).

- [SQLAlchemy](#). This support includes:
 - Code insight
 - Possibility to view database structure in a diagram. Refer to the section [Working with Diagrams](#).
 - Code completion and resolve.

Coding assistance

Coding assistance in PyCharm includes:

- Syntax and error highlighting. The color attributes are configurable in the [Colors and Fonts | <language>](#) pages of the Settings/Preferences dialog.
- [File templates](#) for the supported languages that enable creating stub classes, scripts etc.
- [Live templates](#) for creating complicated code constructs.
- [Code completion](#).
- [Code generation](#).
- [Code folding, formatting](#), and highlighting.
- [Intention actions and quick fixes](#).
- [Ability to view code hierarchy](#).
- [Quick access to the API documentation](#).
- [Using macros in the editor](#).
- [Advanced search and replace facilities](#).
- [Advanced means of navigation](#).
- [Refactoring](#).
- Code completion, syntax and error highlighting for buildout parts.
- Navigation to buildout parts within `buildout.cfg` file.
- The [embedded local terminal](#) where you can execute commands without leaving PyCharm.

Besides editing assistance, PyCharm enables [debugging](#) for JavaScript, and Python and Django applications.

Warning! Debugging for JavaScript applications is supported only in the [Chrome](#) browser.

Encoding

Different types of files use different ways to define encoding. PyCharm recognizes encoding of files based on their contents.

Encoding has influence on the way PyCharm reads or writes files. If a file has been modified but not yet saved, any changes in encoding affect file writing; if a file has not been modified, then reading is affected. PyCharm suggests specific ways to change encoding of a file according to its type, using [File Encodings](#) Settings page, the [Status bar](#), or [the editor](#).

Note Encoding applies to directories and individual files. The encoding information saved in a file overrides the default encoding; encoding of a file or subdirectory overrides encoding settings on the higher levels.

Encoding can be changed in

File encoding is specified within the file, for example, in XML.	If a file contains explicit encoding declaration, you can change it in the Editor . In this case PyCharm provides code completion.
File encoding is defined by BOM.	In this case, you can't change encoding with which PyCharm reads the file, but it is still possible to change encoding for writing such file.
UTF characters are detected in the file contents.	PyCharm provides an option that automatically changes file encoding to UTF , if the file contents can be reasonably interpreted as UTF. This option only works for reading; a file can be saved with any encoding.
Encoding cannot be found out from the file content.	In this case, the default encoding is the one defined by the IDE encoding in the File Encodings page of the Settings dialog. You can change it for multiple files and directories , or for a single file .

Path Variables

To avoid complications when moving [projects](#) from one computer to another, PyCharm provides path variables. (A path variable, obviously, is a variable whose value is an absolute path to a directory or file.)

Path variables are particularly useful when working with third-party [libraries](#) stored outside the project directory.

To illustrate, let's assume there is a project shared among a team of developers (e.g., through version control), and there is a library in this project defined at the project level.

Everything goes fine if the library is located under the project content root. This, however, is rarely the case.

A more typical situation is when the library location is external to the projects. Such a library is referenced by its absolute path and there's no guarantee that this path is the same on every one of the computers used by the team.

The obvious solution is to define a path variable for the library location. In such a case, the library path may be set individually on each of the computers.

To summarize, you should define path variables under the following set of conditions:

- There are third-party libraries in your project defined at the project level.
- These libraries are stored outside the project content root.
- There is a need to work with the project on more than one computer (e.g., share the project among the development team members through version control).

Ignored variables

If when opening a project, PyCharm detects unresolved path variables, it asks you to define proper values for them. If for some reason you don't want to do that (e.g., you are not going to use the corresponding library or libraries), you have an option of adding the corresponding variables to a list of ignored variables.

There may also be other cases when the list of ignored variables is useful.

At the internal level, path variables are represented by strings in which the name of a variable is enclosed between a pair of dollar sign characters, for example, `$MY_PATH_VARIABLE$`.

Such a pattern, in principle, may occur in your project without the meaning of a path variable.

To tell PyCharm that a string that starts and ends with a dollar sign character (e.g., `$SOME_STRING$`), actually, is not a path variable, you should add such a string (e.g., `SOME_STRING`) to the list of ignored variables.

Path variables and the list of ignored variables are configured on the [Path Variables](#) page of the Settings dialog.

Dialogs

This part contains descriptions of the following dialogs:

- [Breakpoints](#)
- [Bookmarks Dialog](#)
- [Color Picker](#)
- [Confirm Drop Dialog](#)
- [Create Table and Modify Table Dialogs](#)
- [Create Test Dialog](#)
- [CSV Formats Dialog](#)
- [Differences viewers](#)
- [Docker Registry Dialog](#)
- [Edit as Table: <file_name> Format Dialog](#)
- [Edit Log Files Aliases Dialog](#)
- [Edit Macros Dialog](#)
- [Edit Project Path Mappings Dialog](#)
- [Evaluate Expression](#)
- [Export Test Results](#)
- [Export to HTML](#)
- [Extract Variable Dialog for Sass](#)
- [File Cache Conflict](#)
- [Find Usages](#)
- [Generate Instance Document from Schema Dialog](#)
- [Import File Dialog](#)
- [Import File dialog when called from the table editor](#)
- [Import Table dialog](#)
- [Login to IntelliJ Configuration Server Dialog](#)
- [Map External Resource Dialog](#)
- [Non-Project Files Protection Dialog](#)
- [Open Task Dialog](#)
- [Optimize Imports Dialog](#)
- [Print](#)
- [Productivity Guide](#)
- [PSI Viewer](#)
- [Pull Image Dialog](#)
- [Push Image Dialog](#)
- [PyCharm License Activation Dialog](#)
- [Recent Changes Dialog](#)
- [Refactoring Dialogs](#)
- [Reformat File Dialog](#)
- [Register New File Type Association Dialog](#)
- [Run/Debug Configurations](#)
- [Save File as Template Dialog](#)
- [Select Path Dialog](#)
- [Specify Code Cleanup Scope Dialog](#)
- [Specify Code Duplication Analysis Scope](#)
- [Specify Inspection Scope Dialog](#)
- [Structural Search and Replace Dialogs](#)

Breakpoints

Run | View Breakpoints

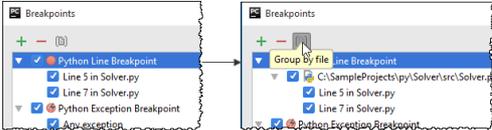
Ctrl+Shift+F8



In this section:

- [Toolbar](#)
- [Breakpoint options](#)
- [Context menu commands](#)
- [Speed search of a breakpoint](#)

Toolbar

Item	Tooltip and shortcut	Description
	Add Breakpoint Alt+Insert	Click to show the list of available breakpoint types . Select the desired type to create a new breakpoint.
	Remove Breakpoint	Click this button to remove selected breakpoints.
	Group by File	Click this button to display breakpoints under their respective files, rather than under their types: 

Breakpoint options

The controls of this part of the Breakpoints dialog depend on the type of the selected breakpoint.

OptionDescriptionTypes of breakpoints

Suspend	<p>Select this check box to enable suspend policy for a breakpoint.</p> <p>Select one of the radio buttons to specify the way the running of the program is paused when a breakpoint is reached. If you work with JavaScript breakpoints, you only need to specify whether you want to suspend program execution when the breakpoint is hit.</p> <p>If the check box is not selected, no threads are suspended.</p> <p>SuspendDescription policy</p> <table border="1"><tbody><tr><td>All</td><td>When a breakpoint is hit, all threads are suspended.</td></tr><tr><td>Thread</td><td>When a breakpoint is hit, the thread where the breakpoint is hit, is suspended.</td></tr><tr><td>Make default</td><td>Click this button if you want the suspend policy specified for the breakpoint in question to be used as the default one for the subsequently created breakpoints.</td></tr></tbody></table> <p> This button only appears, when Thread option is selected.</p>	All	When a breakpoint is hit, all threads are suspended.	Thread	When a breakpoint is hit, the thread where the breakpoint is hit, is suspended.	Make default	Click this button if you want the suspend policy specified for the breakpoint in question to be used as the default one for the subsequently created breakpoints.	Python line breakpoints
All	When a breakpoint is hit, all threads are suspended.							
Thread	When a breakpoint is hit, the thread where the breakpoint is hit, is suspended.							
Make default	Click this button if you want the suspend policy specified for the breakpoint in question to be used as the default one for the subsequently created breakpoints.							
Condition	<p>Select this check box and specify a condition for hitting a breakpoint in the text field. A condition is a Boolean expression.</p> <p>This expression should be valid at the line where the breakpoint is set, and is evaluated every time the breakpoint is reached. If the evaluation result is <code>true</code>, user-selected actions are performed.</p> <p>If the result is <code>false</code>, the breakpoint does not produce any effect. If the Debugger cannot evaluate the expression, it displays the Condition evaluation error message. You can select whether you would like to stop at this breakpoint or ignore it.</p> <p>To the right of the Condition field, there is the button (Shift+Enter) that opens the multiline editor.</p>	All types						
Log message to console	<p>Select this check box if you want a log message to be displayed in the console output when the breakpoint is hit.</p>	All types						
Evaluate and log	<p>Select this check box if you wish to evaluate a certain expression at this breakpoint and to export result to the console output.</p> <p>To the right of this field, there is the button (Shift+Enter) that opens the multiline editor.</p> <p> If the expression to be evaluated is incorrect when a particular breakpoint is reached, the console output displays an error message:</p> <pre>Unable to evaluate expression <your_expression></pre>	Line breakpoints						

Remove once hit	Select this check box, if the you want the breakpoint to be deleted after hitting it.	All types
Disabled until selected breakpoint is hit	From the drop-down list, select the breakpoint in question. The option None corresponds to the always enabled breakpoint. Besides that, you can also choose the behavior of this breakpoint, when the selected one is hit: – Disable again – Leave enabled	All types
Activation policy		
On termination	The Debugger stops when the process terminates with this exception.	Python exception breakpoints
On raise	If this option is selected, the Debugger stops on throwing an exception. So doing, the Debugger stops only on the first place where the exception has been thrown.	Python exception breakpoints
Ignore library files	If this check box is selected, the debugger does not stop at the exceptions thrown inside libraries. If this check box is not selected, the debugger stops at the location in a library file, where the exception is thrown.	Python exception breakpoints

Context menu commands

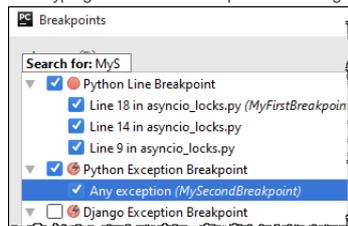
CommandDescription

Move to group	Point to this command to move the selected breakpoint to a new group , to one of the existing groups(<code><group name></code>), or out of a group (<code><no group></code>).
Edit description	Choose this command to enter or change description of a breakpoint .

Speed search of a breakpoint

To find a particular breakpoint

– Start typing address or description of the target breakpoint:



PyCharm highlights the line with the matching address or description.

Bookmarks Dialog

Navigate | Bookmarks | Show Bookmarks

Shift+F11

Use this dialog to navigate between bookmarks, and manage the collection of anonymous bookmarks and bookmarks with mnemonics in a project.

Toolbar options

ItemShortcutDescription

	Ctrl+Enter	Click to add/edit description for the selected bookmark.
	Alt+Delete	Click to delete the selected bookmark.
	Alt+Down	Use these buttons to reorder bookmarks.
	Alt+Up	

The left pane of the Bookmarks dialog displays the list of bookmarks created with the brief description, if any. The order of bookmarks in the collection defines the order in which Navigate | Bookmarks | Next/Previous Bookmark command visits the bookmarks. The right pane displays the preview of the file where the selected bookmark is toggled.

To close the dialog box, press `Escape`.

Color Picker

Settings | Editor | Colors and Fonts

Ctrl+Shift+A

Use this dialog to find the exact RGB, HSB and hex values of any color component.

ItemDescription

	Click this eyedropper to navigate to the object and obtain its color RGB or HSB as well as its color hex values.
Color mixer	Use this area to select a color.
Stack of colors	This area displays colors that were last selected.
Brightness	Move the slider upwards to make to colors in the Color mixer area brighter; move the slider down to make colors darker.
Choose	Click this button to select a color from the Color Mixer area, or the Stack of colors.
Cancel	Click this button to omit saving the color information.

Confirm Drop Dialog

From the [Database tool window](#):

Edit | Drop, Drop from the context menu, or  .

Click OK to remove the selected item or items, or click  to copy the SQL statement or statements into the [database console](#).

ItemDescription

SQL Preview The statement or statements to be run to remove the selected item or items. If necessary, you can edit the statements right in this pane. The statements are executed when you click OK.

 Copy the statements into the corresponding database console and close the dialog.

Create Table and Modify Table Dialogs

- [Upper part: GUI](#)
- [SQL Script](#)
- [Options](#)

Upper part: GUI

The upper part of the dialog provides a GUI for defining the table structure, and associated constraints and indexes.

SQL Script

The pane under SQL Script shows the statement or statements to be run to achieve the result you have specified using the GUI.

You can use this pane just as a preview. You can also edit the statement or statements right there.

Options

Item	Description
Execute in database	<p>Change data structure in your database by running the statement or statements.</p> <p>Execute. Execute the statements right away. If associated changes in database consoles are possible, the corresponding preview is shown. See Previewing changes.</p> <p>Preview. Preview potential associated changes prior to executing the statements.</p>
Replace existing DDL	<p>Replace the definition (usually, a <code>CREATE</code> statement) in the corresponding database console or SQL file with the statements shown under SQL Script.</p> <p>Replace. Perform the replacement right away. If associated changes are possible, the corresponding preview is shown.</p> <p>Preview. Preview potential associated changes prior to performing the replacement.</p>
Open in editor	<p>Copy the statements into the corresponding database console or SQL file.</p> <ul style="list-style-type: none">- Modify existing objects. In the generated set of statements, <code>ALTER</code> statements are preferred to <code>CREATE</code> statements.- Create modified objects. The changes you have made using the GUI are translated into a minimal set of <code>CREATE</code> and other statements.- Create all objects. The generated set always includes the <code>CREATE TABLE</code> statement, as if the whole table were created anew. <p>To Editor. Switch to the corresponding editor tab right away. If associated changes are possible, the corresponding preview is shown prior to copying the statements.</p> <p>Preview. Preview potential associated changes.</p>

Create Test Dialog

Navigate | Test - Create New Test

Ctrl+Shift+T

Use this dialog box to generate a test class for a class or method, if such test doesn't yet exist.

ItemDescription

Target directory Click the browse button  to open the directory chooser dialog box, and select the target directory, where the desired test file will be generated.

Test file name In this text field, specify the name of a file where test class will be generated, or accept the default name.

Test class name In this text field, specify the name of the test class, or accept the default name.

Test method This list shows all methods of a class in question (if the dialog has been invoked from the class declaration), or the name of the only method (if the dialog has been invoked from within this method).
Select the check box next to the method names to include these test methods in the generated test class.

CSV Formats Dialog

To access this dialog:

- From any of the table views:

Right-click the table and select Data Extractor | Configure CSV Formats.

- From the [Database tool window](#):

Right-click the table or view of interest, select Dump Data to File | Configure CSV Formats.

This dialog lets you specify the settings for converting table data into delimiter-separated values formats (e.g. CSV, TSV) and vice versa.

When working on the conversion settings, use the preview in the right-hand part of the dialog.

ItemDescription

Formats	The list of the available delimiter-separated values formats is shown. Each format is a named set of corresponding conversion settings. Select the format whose settings you want to view or edit. Use + , - , ↑ and ↓ to create, delete and reorder the formats; 📄 to create a copy of the selected format.
Value separator	Select or type the character for separating individual values.
Row separator	Select or type the character for separating rows.
Null value text	The text to be used as a value if a cell contains <code>null</code> (an unknown value).
Add row prefix/suffix	Row prefix and suffix are character sequences which in addition to the row separator indicate the beginning and end of a row. If necessary, click the link and specify the row prefix and suffix in the fields that appear.
Quotation	Each line in the area under Quotation is a quotation pattern (see Quote values). A quotation pattern includes: <ul style="list-style-type: none">- The left quotation character, the one inserted before a value.- The right quotation character, the one inserted after a value; usually, the same as the left quotation character.- An escape method or character for the cases when the quotation character is part of a value. E.g. Escape: duplicate means that if a quotation character occurs within a value, it is doubled. (You can specify your own escape character instead.) <p>If there is more than one pattern, the first of the patterns is used.</p> <p>Use +, -, ↑ and ↓ to create, delete and reorder the patterns.</p> <p>To start editing an existing pattern, just click the pattern of interest.</p>
Quote values	Specify in which cases the values should be quoted (i.e. enclosed within quotation characters). <ul style="list-style-type: none">- When needed. A value is quoted only if it contains the value and/or the row separator.- Always. Any value is quoted in its text representation.
Trim whitespaces	If this check box is not selected, the Unicode whitespace characters that precede and follow the value separators are treated as parts of the corresponding values. If this check box is selected, the corresponding whitespace characters are ignored or removed.
First row is header	If this check box is selected, the first row is treated as containing column names. The settings that appear under Header Format have the same meanings as the ones above but are applied to the first row.
First column is header	If this check box is selected, the first column is treated as containing row names.

Differences viewers

- [Differences Viewer for Files](#)
- [Differences Viewer for Folders](#)
- [Differences Viewer for Table Structures](#)
- [Differences Viewer for Tables](#)

Differences Viewer for Files

Project tool window | context menu of a file | Compare File with Editor

Project tool window | context menu of a file | Compare Files

Version Control tool window | Local Changes tab | 

Version Control tool window | context menu of a folder or file | Show Diff

In this section:

- [Basics](#)
- [Diff & Merge viewer](#)
- [Keyboard shortcuts](#)
- [Context menu commands](#)

Basics

This dialog is displayed every time you compare two files, or compare two versions of a file (local changes or changes between local files and their revisions in a remote repository). You can compare files of any types, including binaries and `.jar` files.

Note that you can open the differences viewer without running PyCharm. To do this, execute the following command:

```
<path to PyCharm executable file> diff <path_1> <path_2>
```

where `path_1` and `path_2` are paths to the files you want to compare.

The differences viewer provides a powerful editor that enables code completion, live templates, etc.

Diff & Merge viewer

ItemTooltip Description and Shortcut

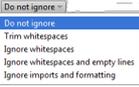
	Previous Difference / Next Difference  	Use these buttons to jump to the next/previous difference. When the last/first difference is hit, PyCharm suggests to click the arrow buttons  /  once more and compare other files, depending on the Go to the next file after reaching last change option in the Differences Viewer settings . This behavior is supported only when the Differences Viewer is invoked from the Version Control tool window.
---	--	---

 	Compare Previous/Next File  	Click these buttons to compare the local copy of the previous/next file with its update from the server. Note These controls are only available if more than one file has been modified locally.
--	---	--

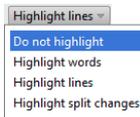
	Go To Changed File 	Click this button to display all changed files in a current change set (and navigate to them).
--	---	--

	Jump to Source 	Click this button to open the selected file in the active pane in the editor. The caret will be placed in the same position as in the Differences Viewer.
--	---	---

Viewer type		Use this drop-down list to choose the desired viewer type. The side-by-side viewer has two panels; the unified viewer has one panel only. Both types of viewers enable you to - Edit code. Note that one can change text only in the right-hand part of the default viewer, or, in case of the unified viewer, in the lower ("after") line, i.e. in your local version of the file. - Perform the Apply/Append/Revert actions.
-------------	---	---

Whitespase		Use this drop-down list to define how the differences viewer should treat white spaces in the text. - Do not ignore: white spaces are important, and all differences are highlighted. This option is selected by default. - Trim whitespaces: (" <code>\t</code> ", " ") , if they appear in the end and in the beginning of a line. - If two lines differ in trailing whitespaces only, these lines are considered equal. - If two lines are different, such trailing whitespaces are not highlighted in the By word mode. - Ignore whitespaces: white spaces are not important, regardless of their location in the source code. - Ignore whitespaces and empty lines: the following entities are ignored: - all whitespaces (as in the 'Ignore whitespaces' option) - all added or removed lines consisting of whitespaces only - all changes consisting of splitting or joining lines without changes to non-whitespace parts. For example, changing <code>a b c</code> to <code>a \n b c</code> is not highlighted in this mode.
------------	---	---

Highlighting mode	Select the way differences granularity is highlighted. The available options are: - Highlight words : the modified words are highlighted
-------------------	--



- Highlight lines: the modified lines are highlighted
- Highlight split changes: if this option is selected, big changes are split into smaller 'atomic' changes.
For example, `A \n B` vs. `A X \n B X` will be treated as two changes instead of one.
- Do not highlight: if this option is selected, the differences are not highlighted at all. This option is intended for significantly modified files, where highlighting only introduces additional difficulties.

	Collapse unchanged fragments	Click this button to collapse all unchanged fragments in both files. The amount of non-collapsible unchanged lines is configurable in the Diff & Merge settings page.
	Synchronize scrolling	Click this button to simultaneously scroll both differences panes; if this button is released, each of the panes can be scrolled independently.
	Editor settings	Click this button to invoke the list of available settings. Select or clear this options to show or hide whitespaces, line numbers and indent guides, to use or disable the use of soft wraps, and to set the highlighting level. These commands are also available from the context menu of the differences viewer gutter.

Include into commit `Alt+I`

This checkbox only appears if you invoke the Differences Viewer from the [Commit Changes dialog](#) with multiple changed files (all of which are deselected), and you explore the differences between them and hit the last difference in a file. Select this checkbox if you want to include the file you've reviewed into the commit.

Move to Another Changelist `F6`

This button only appears if you invoke the Differences Viewer from the [Commit Changes dialog](#) with multiple changed files (all of which are deselected), and you explore the differences between them and hit the last difference in a file. Click this icon to [move](#) the file you've reviewed to another changelist.

Show diff in external tool

Click this button to invoke an external differences viewer, specified in the [External Diff Tools](#) settings page. This button only appears on the toolbar when the Use external diff tool option is enabled in the [External Diff Tools](#) settings page.

Help

Click this button to show the corresponding help page.

`F1`

`Ctrl+Tab`

Use this keyboard shortcut to switch between the panes of the Differences viewer. The active pane has the cursor.

Use these [chevron](#) buttons to apply differences between panes (in case of the side-by-side viewer) or between lines (in case of the unified viewer). The chevron buttons can change their behavior:

- Click to apply changes. This behavior is the default one.
- Press `Ctrl` to change to or and append changes.

Merge actions

Compare Left and Middle/Middle and Right/Left and Right Contents

Click these buttons to compare left/middle/right parts in a new window.

Apply All Non-Conflicting Changes

Click this button to apply all non-conflicting changes. You can also make this behavior automatic, by selecting the check box [Automatically apply non-conflicting changes](#) in the [Diff & Merge](#) page of the [Settings/Preferences dialog](#).

Apply Non-Conflicting Changes from the Left/Right Side

Click these buttons to merge non-conflicting changes from the left/right parts of the dialog.

N/A **Annotate**

This option is only available from the context menu of the gutter.

Use this option to explore who introduced which changes to the repository version of the file in question, and when. The annotations view lets you see detailed information for each line of code, such as the version from which this line originated, the ID of the user who committed this line, and the commit date.

You can [configure the amount of information displayed in the annotations pane](#).

For more details on annotations, refer to [Viewing Changes Information](#)

Keyboard shortcuts

Keyboard shortcut **Description**

<code>Ctrl+Shift+D</code>	Use this keyboard shortcut to show the popup menu of the most commonly user diff commands.
<code>Ctrl+Tab</code>	Use this keyboard shortcut to switch between the left and the right panes.
<code>Ctrl+Shift+Tab</code>	Use this keyboard shortcut to select the position obtained by <code>Ctrl+Tab</code> in the opposite pane.
<code>Ctrl+Z</code> / <code>Ctrl+Shift+Z</code>	Use this keyboard shortcut to undo/redo a merge operation. Conflicts will be kept in sync with the text.

Context menu commands

This context menu is available in the middle of the editor:

ItemDescription

Show Whitespaces	Select this check command to show whitespaces as the dots in the Differences Viewer .
Show Line Numbers	Select this check command to show line numbers in the Differences Viewer.
Show Indent Guides	Select this check command to have PyCharm display vertical lines in the Differences Viewer to indicate positions of indents.
Use Soft Wraps	Select this check command to have PyCharm wrap the lines of code, when the dialog box is resized.
Highlighting level	Use this menu item to select the highlighting level in the Differences Viewer. To learn more about the level of highlighting, refer to the description of the Status Bar .

Annotate Select this check command to [annotate](#) the changes.

Tip This command is available only for the files under version control.

This context menu is available in both editors:

ItemDescription

Accept/Append	Select these commands to accept or append the lines shown in the Differences Viewer.
Compare with Clipboard	Select this command to compare the file in the respective pane of the Differences Viewer with the contents of the the Clipboard .
Annotate	Select this check command to annotate the changes.

Tip This command is available only for the files under version control.

This context menu is available in the right-hand strip of the Differences Viewer:

ItemDescription

Go to high-priority problems only/Go to next problem	Click one of these radio-buttons to define the way of navigating between the encountered problems.
Customize highlighting level	Click this command to show the slider to change the highlighting level in the Differences Viewer. To learn more about the changing the highlighting level, refer to the sections Status Bar and Changing Highlighting Level for the Current File .
Show code lens on scrollbar hover	Select this check command to switch the Differences Viewer to the lens mode .

Differences Viewer for Folders

Project tool window | context menu of a folder | Compare Directory with

Project tool window | context menu of two selected folders | Compare Directories

Project tool window | context menu of a folder | Sync with Deployed to

Remote Host tool window | context menu of a folder | Sync with local

Database tool window | context menu of two selected items | Compare

This window is displayed every time you explore differences between:

- [Two local directories](#) (local history for folders, recent changes, or version control).
- [A remote folder and its local version.](#)

In this dialog box, explore the detected differences and synchronize the compared items.

Tip You can also open the difference viewer without running PyCharm. This is done through the following command: `<path to PyCharm executable file> diff <path_1> <path_2>` where `path_1` and `path_2` are paths to the folders in question.

In this section:

- [Toolbar](#)
- [List of items](#)
- [Differences pane](#)
- [Diff viewer](#)
- [Context menu](#)

Toolbar

Item	Tooltip and shortcut	Description	Available for
	F7 Shift+F7	Use these buttons to jump to the next/previous difference. When the last/first difference is hit, PyCharm suggests to press F7 / Shift+F7 once more and compare other files.	Version control
Tip This feature becomes available only when the Differences Viewer is invoked from the Version Control tool window.			
		Click this button to open file in the editor's active tab. The caret will be placed in the same position as in the Differences Viewer.	All
	F4	Click this button to invoke an external differences viewer, specified in the External Diff Tools page. This button only appears on the toolbar, when the check box Use external diff tool is selected in the External Diff Tools page.	All
		Click this button to show reference page.	All
	F1		
		Click this button to refresh the contents of the differences viewer.	All
	F5		
		Press this toggle button to have PyCharm show the items, that are present in the first of the compared directories or database objects and are missing in the second one.	All
		Press this toggle button to have PyCharm show items that are present in both folders or database objects but differ in contents, or timestamp, or size.	All
		Press this toggle button to have PyCharm show the items that are present in both directories or objects and have the same contents, or timestamp, or size, depending on the parameter for comparison specified in the Compare by drop-down list.	All
		Press this toggle button to have PyCharm show the items, that are present in the second of the compared directories or database objects and are missing in the first one.	All
Compare by		In this drop-down list, select the parameter to be used for comparison. The available options are: - Contents - Size - Time stamp	Local folders Local-remote folders

	Synchronize Selected	Click this button to have PyCharm apply the specified action to the selected pair of items. Actions to be performed are shown in the * field.	All
			
	Synchronize All	Click this button to have PyCharm apply the specified action to all the pairs of items in the list. Actions to be performed are shown in the * field.	All
			
	Hide excluded files	Click this button to suppress showing files excluded from synchronization .	Local-remote folders
Filter		Type the filtering string (for example, file or table name). Use * wildcard to replace any number of arbitrary characters. Note that filter applies on pressing  .	All
Path		These fields show the paths to the folders being compared. To change a directory, click the Browse button  and specify another directory in the dialog that opens .	Local folders Local-remote folders

List of items

The list shows the items from the compared objects that meet the comparison criterion specified in the [Compare by](#) drop-down list and the filtering criteria specified through the [toolbar buttons](#).

Item	Description	Available for
Name	These read-only fields show the names of files under the object specified in the Path fields.	All
Size	These read-only fields show the sizes of files under the folders being compared.	Local folders Local-remote folders
Date	These read-only fields show the timestamps of files under the folders being compared.	Local folders Local-remote folders

* The icon in this field indicates the action that will be applied to the pair of items in the current line upon clicking the Synchronize Selected  or Synchronize All  toolbar button. To change the currently selected action, click the icon.

IconAction

	Copy the item in the left side to the right side, possibly overwriting the contents of the corresponding target item, if it already exists.
	Copy the item in the right side to the left side, possibly overwriting the contents of the corresponding target item, if it already exists.
	The items are treated identical with regard to the selected criterion of comparison. No action will be performed by default.
	The items differ with regard to the selected criterion of comparison. No action will be performed by default. Explore the differences in the Differences Pane and change the intended action by clicking the icon.
	The item is present only in one of the folders and will be removed.

Differences pane

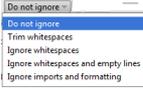
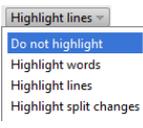
The differences pane is displayed only for files with the same names, which exist on both sides. For the files and DB objects that exist on one side only, the contents of the selected file/DB object is displayed.

If the files have read-only status, they are not editable in the differences pane.

Diff viewer

ItemTooltip Description and Shortcut

	Previous Difference / Next Difference  	Use these buttons to jump to the next/previous difference. When the last/first difference is hit, PyCharm suggests to click the arrow buttons  /  once more and compare other files, depending on the Go to the next file after reaching last change option in the Differences Viewer settings .
		This behavior is supported only when the Differences Viewer is invoked from the Version Control tool window.

 	<p>Compare Previous/Next File</p> <p>Alt+Left Alt+Right</p>	<p>Click these buttons to compare the local copy of the previous/next file with its update from the server.</p> <p>Note These controls are only available if more than one file has been modified locally.</p>
	<p>Go To Changed File</p> <p>Ctrl+N</p>	<p>Click this button to display all changed files in a current change set (and navigate to them).</p>
	<p>Jump to Source</p> <p>F4</p>	<p>Click this button to open the selected file in the active pane in the editor. The caret will be placed in the same position as in the Differences Viewer.</p>
<p>Viewer type</p>		<p>Use this drop-down list to choose the desired viewer type. The side-by-side viewer has two panels; the unified viewer has one panel only. Both types of viewers enable you to</p> <ul style="list-style-type: none"> – Edit code. Note that one can change text <i>only</i> in the right-hand part of the default viewer, or, in case of the unified viewer, in the lower ("after") line, i.e. in your local version of the file. – Perform the Apply/Append/Revert actions.
<p>Whitespace</p>		<p>Use this drop-down list to define how the differences viewer should treat white spaces in the text.</p> <ul style="list-style-type: none"> – Do not ignore: white spaces are important, and all differences are highlighted. This option is selected by default. – Trim whitespaces: (" \t", " ") , if they appear in the end and in the beginning of a line. <ul style="list-style-type: none"> – If two lines differ in trailing whitespaces only, these lines are considered equal. – If two lines are different, such trailing whitespaces are not highlighted in the By word mode. – Ignore whitespaces: white spaces are not important, regardless of their location in the source code. – Ignore whitespaces and empty lines: the following entities are ignored: <ul style="list-style-type: none"> – all whitespaces (as in the 'Ignore whitespaces' option) – all added or removed lines consisting of whitespaces only – all changes consisting of splitting or joining lines without changes to non-whitespace parts. <p>For example, changing <code>a b c</code> to <code>a \n b c</code> is not highlighted in this mode.</p>
<p>Highlighting mode</p>		<p>Select the way differences granularity is highlighted.</p> <p>The available options are:</p> <ul style="list-style-type: none"> – Highlight words: the modified words are highlighted – Highlight lines: the modified lines are highlighted – Highlight split changes: if this option is selected, big changes are split into smaller 'atomic' changes. <p>For example, <code>A \n B</code> vs. <code>A X \n B X</code> will be treated as two changes instead of one.</p> <ul style="list-style-type: none"> – Do not highlight: if this option is selected, the differences are not highlighted at all. This option is intended for significantly modified files, where highlighting only introduces additional difficulties.
	<p>Collapse unchanged fragments</p>	<p>Click this button to collapse all unchanged fragments in both files. The amount of non-collapsible unchanged lines is configurable in the Diff & Merge settings page.</p>
	<p>Synchronize scrolling</p>	<p>Click this button to simultaneously scroll both differences panes; if this button is released, each of the panes can be scrolled independently.</p>
	<p>Editor settings</p>	<p>Click this button to invoke the list of available settings. Select or clear this options to show or hide whitespaces, line numbers and indent guides, to use or disable the use of soft wraps, and to set the highlighting level.</p> <p>These commands are also available from the context menu of the differences viewer gutter.</p>
<p>Include into commit</p> <p>Alt+I</p>	<p>Include into commit</p> <p>Alt+I</p>	<p>This checkbox only appears if you invoke the Differences Viewer from the Commit Changes dialog with multiple changed files (all of which are deselected), and you explore the differences between them and hit the last difference in a file.</p> <p>Select this checkbox if you want to include the file you've reviewed into the commit.</p>
	<p>Move to Another Changelist</p> <p>F6</p>	<p>This button only appears if you invoke the Differences Viewer from the Commit Changes dialog with multiple changed files (all of which are deselected), and you explore the differences between them and hit the last difference in a file.</p> <p>Click this icon to move the file you've reviewed to another changelist.</p>
	<p>Show diff in external tool</p>	<p>Click this button to invoke an external differences viewer, specified in the External Diff Tools settings page.</p> <p>This button only appears on the toolbar when the Use external diff tool option is enabled in the External Diff Tools settings page.</p>
<p>Help</p> <p>F1</p> <p>Ctrl+Tab</p>	<p>Help</p> <p>F1</p> <p>Ctrl+Tab</p>	<p>Click this button to show the corresponding help page.</p> <p>Use this keyboard shortcut to switch between the panes of the Differences viewer. The active pane has the cursor.</p>
		<p>Use these chevron buttons to apply differences between panes (in case of the side-by-side viewer) or between lines (in case of the unified viewer).</p>

The chevron buttons can change their behavior:

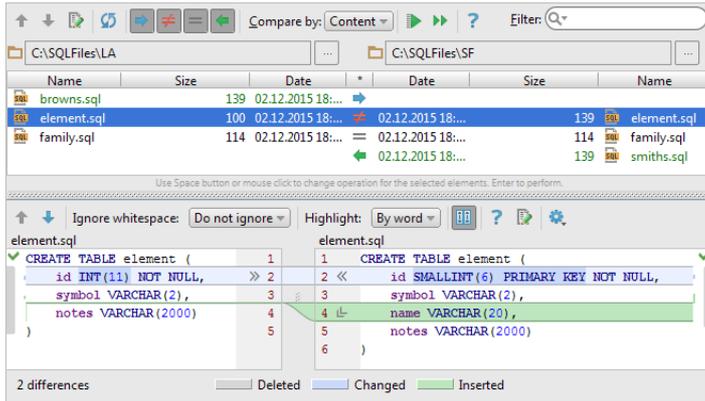
- Click **↔** to apply changes. This behavior is the default one.
- Press **Ctrl** to change **↔** to **↵** or **⇆** and append changes.

Context menu

This menu appears on right-clicking an entry in the list of items. The commands listed below define actions to be taken for the selected entry and set the action icons in the * column.

ItemIconDescription

Set Copy to Right/Left	↔ This command sets the specified icons in the * column to copy a file missing from one of the directories. If a file exists on one side, it will be copied; if it doesn't exist, then the file on the other side will not be deleted.
Set Delete	✖ This command sets the specified icons in the * column to delete file.
Set Do Nothing	Choose this command to remove an action icon.
Set Default	Choose this command to set the default action for the entry.
Warn When Delete	Select this option to display a warning when trying to delete a file that is located only in one of the two directories during their merge.



File comparison statuses and intended operations are shown in the column marked with an asterisk (*). To assign or change an operation, use the context menu associated with the corresponding cell. Alternatively, click the cell or press **Space** one or more times.

Applying operations:

- **Enter** applies the operations to selected files.
- **Ctrl+Enter** applies the operations to all the files.

IconDescription

↔	The file exists only in the left-hand folder. The intended operation is to copy the file to the right-hand folder. If a file exists in both folders and you apply this operation, the file in the right-hand folder is overwritten.
≠	For the selected comparison criterion, the files are not identical. No operation is assumed. Study the file differences in the lower part of the view. You can choose to overwrite one of the files by assigning and applying the corresponding operation. You can as well modify the file contents. This may be done by typing or by using the following buttons and context menu commands: <ul style="list-style-type: none"> - ↔ or ↵ or Replace. Replace the fragment with the one from the other pane. - ↵ or ⇆ or Insert. Insert the fragment into the other pane. - Remove. Remove the corresponding fragment. To undo the changes, use Ctrl+Z .
=	For the selected comparison criterion, the files are identical. No operation is assumed.
↵	The file exists only in the right-hand folder. The intended operation is to copy the file to the left-hand folder. If a file exists in both folders and you apply this operation, the file in the left-hand folder is overwritten.
✖	Delete the file. The operation is not available for files that exist in both folders.

Differences Viewer for Table Structures

test/element | chemistry/element

Name	*	Name
id		id
notes		name
symbol		primary
		symbol

Use Space button or mouse click to change operation for the selected elements. Enter to perform.

Ignore whitespace: Do not ignore | Highlight: By word

id (Read-only) | id (Read-only)

```
id int(11) NOT NULL | id smallint(6) NOT NULL
```

1 difference | Deleted | Changed | Inserted

Comparison statuses are shown in the column marked with an asterisk (*).

IconDescription

	The column, constraint or index exists only in the left-hand table.
	The items exist in both tables but their definitions are different. You can study the differences in the lower part of the view.
	The items are identical.
	The item exists only in the right-hand table.

Generating ALTER TABLE statements

You can generate a set of `ALTER TABLE` statements for making data definitions in the left-hand and the right-hand parts identical. Use one of the following buttons in the upper part of the view:

- Migrate Left. The statements for the left-hand table or tables are generated.
- Migrate Right. This buttons does the same but for the right-hand table or tables.

The `ALTER TABLE` statements are generated for the items marked , and . If you don't want to generate the statements for some of those items, right click the * cell and select Set Do Nothing for all such items.

Differences Viewer for Tables

The primary purpose of the differences viewer for tables is to show the differences and similarities of data.

test.browns (Read-only)			test.smiths (Read-only)		
name	relation		member_id	name	relation
Emily	mother	1	1 1	Chloe	mother
Harry	father	2	2 2	Harry	father
Dylan	brother	3			

Docker Registry Dialog

For this dialog to be available, the Docker integration plugin must be installed and enabled.

Specify your [Docker](#) image repository user account settings.

ItemDescription

Name	The name for this set of settings.
Address	The image repository service URL, e.g. – registry.hub.docker.com for Docker Hub – quay.io for Quay
Username	The user name for your user account.
Password	Your password.
Email	The email address that you specified when creating your user account.
Server	The associated Docker configuration (used to connect to the service to check that your user account settings are correct).

Edit as Table: &file_name> Format Dialog

If a text file containing delimiter-separated values (e.g. CSV, TSV) is open in the editor, this dialog opens when you select the Edit as Table command.

Specify how your delimiter-separated values should be converted into table format.

When working on the conversion settings, use the table preview in the right-hand part of the dialog.

ItemDescription

Formats	Select your file format and check the table preview. If you haven't achieved the desired result yet, adjust the settings.  If you have changed the settings and want to save the changes, click this icon and select one of the following: <ul style="list-style-type: none">– Save Changes. The settings are saved "under the same name", without creating a new format. (A format, in fact, is a named set of settings.)– Save As. The settings are saved "under a different name": a new format is created and you can specify the name for that new format.
Value separator	Select or type the character used for separating individual values.
Row separator	Select or type the character that should be treated as a row separator.
Null value text	The text to be used as a value if a cell contains <code>null</code> (an unknown value).
Add row prefix/suffix	Row prefix and suffix are character sequences which in addition to the row separator indicate the beginning and end of a row. If necessary, click the link and specify the row prefix and suffix in the fields that appear.
Quotation	Each line in the area under Quotation is a quotation pattern (see Quote values). A quotation pattern includes: <ul style="list-style-type: none">– The left quotation character, the one inserted before a value.– The right quotation character, the one inserted after a value; usually, the same as the left quotation character.– An escape method or character for the cases when the quotation character is part of a value. E.g. Escape: duplicate means that if a quotation character occurs within a value, it is doubled. (You can specify your own escape character instead.) <p>If there is more than one pattern, the first of the patterns is used.</p> <p>Use , ,  and  to create, delete and reorder the patterns.</p> <p>To start editing an existing pattern, just click the pattern of interest.</p>
Quote values	Specify in which cases the values should be quoted (i.e. enclosed within quotation characters). <ul style="list-style-type: none">– When needed. A value is quoted only if it contains the value and/or the row separator.– Always. Any value is quoted in its text representation.
Trim whitespaces	If this check box is not selected, the Unicode whitespace characters that precede and follow the value separators are treated as parts of the corresponding values. If this check box is selected, the corresponding whitespace characters are ignored or removed.
First row is header	If this check box is selected, the first row is treated as containing column names. The settings that appear under Header Format have the same meanings as the ones above but are applied to the first row.
First column is header	If this check box is selected, the first column is treated as containing row names.

Edit Log Files Aliases Dialog

The dialog opens when you click the [+](#) or the [🔍](#) buttons in the Logs tab of the [Run/Debug Configuration dialogs](#).

Use this dialog to specify a file or a group of files that you want to be displayed on dedicated tabs of the [Run](#) or [Debug tool window](#).

ItemDescription

Alias	In this text box, type the alias for the log entry. This alias will be displayed in the Logs tab of the Run/Debug Configuration: Application dialog box and in the header of the dedicated tab of the Run or Debug tool window.
-------	---

Log File Location	In this text box, specify the log files to display during running or debugging. Do one of the following: <ul style="list-style-type: none">– Specify the full path to a specific file. Type the path manually or click the Browse button  and choose the file in the dialog that opens.– Specify the base directory and add an Ant pattern that defines the fileset to be displayed.
-------------------	--

Show All Files Coverable by Pattern	Select this check box to have PyCharm open a separate dedicated tab for each log file that matches the specified pattern.
-------------------------------------	---

Please note the following:

- If no alias is specified, the dedicated tab header shows the path to each log file.
- If a pattern covers more than one file, the tab header shows the name of the file instead of the log entry alias, even if an alias is specified.

Edit Macros Dialog

Edit | Macros | Edit Macros

Use this dialog to change or delete the existing macros. Note that the dialog is not available, when there are no macros.

ItemDescription

Existing macros list The left-hand pane of the dialog shows the list of existing macros.

 Click this button to delete a recorded macro.

 Click this button to change name of a recorded macro in the Rename Macro dialog.

Actions list The right-hand pane shows the list of actions recorded for each macro. The list of actions for each macro is editable - you can remove unnecessary actions.

 Click this button to delete an action from the macro.

Edit Project Path Mappings Dialog

The dialog box opens when you click  next to the Path Mappings field in various cases (for example, in the [Run/Debug Configuration: Node JS](#) dialog , or when you [configure a remote interpreter](#)).

If in the current run/debug configuration you use an interpreter accessible through SFTP connection or located on a Vagrant instance, the mappings are automatically retrieved from the corresponding deployment configuration or `Vagrantfile` and listed in the dialog box. The mappings are read-only.

- To add a custom mapping, click **+** and specify the path in the project and the corresponding path on the remote runtime environment in the Local Path and Remote Path fields respectively. Type the paths manually or click  and select the relevant files or folders in the dialog box that opens.
- To remove a custom mapping, select it in the list and click **-**.

Evaluate Expression

Run | Evaluate Expression

Editor Context Menu | Evaluate Expression

Alt+F8



Use this dialog box to calculate values of expressions or code fragments during the debugging session.

ItemDescription

Expression	Use this field to edit the expression to be evaluated. If an expression is selected in the editor, this field displays selection. Tip This field is available in the Expression Mode.
Statements to Evaluate	Type the group of statements to be evaluated. If a code fragment is selected in the editor, this field displays selection. Tip This field is available in the Code Fragment Mode.
Result	Here the results are displayed.
Evaluate	Click this button to evaluate the current expression or code fragment.
Close	Click this button to close the dialog box.
Code Fragment Mode	Click this button to toggle to the Code Fragment Mode.
Expression Mode	Click this button to toggle to the Expression Mode.

Export Test Results

Run tool window | Test Runner



The dialog box opens when you click the Export Test Results button  in the [Test Runner tab](#) of the [Run tool window](#).

In this dialog box, choose the format in which you want the test output saved and the file to save the test results in.

ItemDescription

Export format	In this area, choose the desired output format. The available options are: <ul style="list-style-type: none">- HTML- XML- Custom, apply XSL template: choose this option to have the results presented according to your own code style. Select the relevant <code>*.xsl</code> code style definition file in the file chooser.
Output	In this section, specify the target file name (File name), and directory (Folder).
Open exported file in browser	Select this check box to automatically open the above defined file with the test results in the default browser.

Export to HTML

File | Export to HTML

Use this dialog to save selected files in HTML format.

ItemDescription

File <name>	Click this radio button to print the file currently selected in the Project view, or open in the editor.
-------------	--

Selected text	Click this radio button to print the text selected in the editor.
---------------	---

All files in the directory	Click this radio button to print all files in the current directory.
----------------------------	--

Include subdirectories	Select this check box to print the files in the subdirectories of the current directory.
------------------------	--

Output directory	Specify fully qualified path to the directory, where the resulting HTML file will be stored.
------------------	--

Show line numbers	Select this check box to include line numbers in the resulting HTML file.
-------------------	---

Generate hyperlinks to classes	Select this check box to replace class names with the hyperlinks to the respective classes.
--------------------------------	---

Open generated HTML in browser	Select this check box to show HTML file in the default browser after export.
--------------------------------	--

Extract Variable Dialog for Sass

Refactor | Extract Variable

Ctrl+Alt+V

Use this dialog box to replace a Sass expression with a variable.

ItemDescription

Name	Specify the name for the new variable.
Place for declaration	Select the place in the source code, where the new variable will be declared. The declaration can be global (a variable is declared in the beginning of a Sass file and is available throughout the whole file), or local (a variable is declared immediately before use, and is available in the current block only).
Replace all occurrences	Select this option to replace all the occurrences of the selected expression (if more than one occurrence of the selected expression is found).

File Cache Conflict

An external process has changed a file, opened and unsaved in PyCharm, which results in two conflicting versions of a file. Resolve this conflict using the following options.

Item	Description
Load FS Changes	Click this button to load the file version produced outside of PyCharm, and overwrite your local changes.
Keep Memory Changes	Click this button to preserve the version produced in PyCharm and stored in cache.
Show Difference	Click this button to invoke the differences viewer that shows the version in the file system to the left, and PyCharm version to the right. You can merge differences as required and load the desired version to PyCharm. By default, the cache version is saved.

Find and Replace in Path

Edit | Find | Find in Path or Replace in Path

`Ctrl+Shift+F` or `Ctrl+Shift+R`

In the [Project tool window](#) and [Navigation Bar](#): Find in Path or Replace in Path from the context menu for a directory.

Specify what you want to find and where. In the Replace in Path window, also specify the replacement text or pattern.

Use `Ctrl+Shift+F` and `Ctrl+Shift+R` to switch between the find and replace modes.

Search pattern and replacement text options

ItemDescription

Match case	Select this check box to have PyCharm distinguish between upper and lowercase letters while searching.
Preserve case	<p>If you select this check box PyCharm retains the case of the first letter and the case of the initial string in general. For example, <i>MyTest</i> will be replaced with <i>Yourtest</i> if you specify <i>yourtest</i> as the replacement.</p> <p>This check box is disabled, if the Case sensitive or Regular expressions check box is selected.</p> <p>This field is available only in the Replace in Path dialog.</p>
Words	<p>Select this check box to have PyCharm search for whole words or their parts, (character strings separated with spaces, tabs, punctuation, or special characters).</p> <p>This check box is disabled, if the Regex check box is selected.</p>
Regex	<p>Select this check box if the specified search pattern should be treated as a regular expression.</p> <p>Tip Click the ? mark next to the check box to view reference on regular expressions syntax.</p>
	<p>Use this drop-down list to confine the search to a certain context, for example:</p> <ul style="list-style-type: none">– anywhere - select this option to search everywhere.– in comments - select this option to confine search to comments, ignoring the other occurrences.– In string literals - select this option to confine search to string literals, ignoring the other occurrences– Except... - select one of the exception options to perform search avoiding comments, string literals or both.
File mask	<p>Select this check box to narrow down the search scope through file masks. In the drop-down list, select the desired mask or specify a new one using wildcards.</p> <ul style="list-style-type: none">– Wildcards can include:<ul style="list-style-type: none">– * to substitute a set of any characters,– ? to substitute a single character,– ! to exclude files. Mind that ! should go first in a particular file name pattern, for example, <code>!*.*gant</code> <p>You can specify multiple file masks, delimited with commas (for example, <code>*.xml,a?.sql,!*.html</code>).</p> <p>Note also, that negated pattern (for example, <code>!*.*min.js</code>) has implicit inclusion pattern <code>*</code>. This allows avoiding such constructs as <code>*</code>, <code>!*.*min.js</code> for every file except minified javascript).</p> <p>If text to find is not entered, and this check box is selected, then PyCharm find all files matching the specified mask, regardless of their contents.</p>
Search field	<p>In this field, specify the search pattern. Type the text manually or select one of the previously specified patterns from the drop-down list.</p> <ul style="list-style-type: none">– If you specify the search pattern through a regular expression, use the <code>\$n</code> format in back references (to refer to a previously found and saved pattern).– This field can be left empty. If there is no text to find, but the File mask check box is selected, then search results include only the files matching the specified mask. <p>Click  icon to see the list of recent search entries.</p>
Replace field	<p>In this field, specify the replacement text. Type the text manually or click  icon to select one of the previously specified replace entries from the drop-down list.</p> <ul style="list-style-type: none">– If you specify the replacement text through a regular expression, use the <code>\$n</code> format in back references.– To use a backslash character <code>\</code> in a regular expression, escape the meaningful backslashes by inserting three extra backslashes before them: <code>\\ \\</code>. <p>This field is available only in the Replace in Path dialog.</p>
All projects	Select this option to search through all the open projects.
In Project	Select this option to search through the entire project.
Module	Select this option to search through a module within the project. PyCharm displays a field with the name of the current module. If you have more than one module you can switch to another module using the drop-down list.
Directory	<p>Select this option to perform search within the specified directory. By default, the text area already contains the directory name where a file currently opened in the editor is located (if you call the dialog from the editor), or where a file selected in the tool window is located (if you call the dialog from the tool window), or the directory name selected in the tool window.</p> <p>Pressing the ellipsis button opens the Select Path dialog, where you can select the necessary directory.</p>
	This icon is only available for the directory search. Select it to set the search to be performed in the chosen directory and its subdirectories.
Scope	<p>Select this option to search in a scope.</p> <p>You can choose one of the scopes from the drop-down list, or click the ellipsis button, and define a new scope in the Scopes dialog.</p>

Preview area

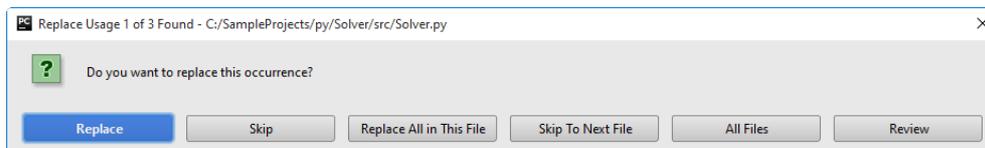
Use this area to check the preview of your search target.

You can press **Up** or **Down** keys to navigate between entries in the preview area without leaving the Search field field. You can press **F4** to get to the selected entry in the editor. You can also edit your entry right in the preview area. PyCharm opens an editor for each search result so you can edit the result without leaving the Find in Path or Replace in Path window. Press **F3** to skip to the next matched entry in the preview editor. You can also press **Ctrl+F** to search through the current file or **Ctrl+R** to replace text in the current file.

ItemDescription

- Click the down arrow to reveal the result presentation options.
 - Skip results tab with one usage - Select this check box to be navigated directly to the found string in the editor, when only one usage is found. The check box is available only in the Find in Path dialog box.
 - Open in new tab - If selected, sets the search results to be displayed in a separate tab.

- Open in Find Window
 - When you click this button in the Find in Path dialog box, PyCharm displays the encountered occurrences of the search string in the [Find tool window](#), selects the first occurrence and opens the file with this occurrence in the editor and moves the focus to it.
 - When you click this button in the Replace in Path dialog box, PyCharm displays the encountered occurrences of the search string in the [Find tool window](#), selects the first occurrence and opens the file with this occurrence in the editor and moves the focus to it. At the same time, PyCharm opens the Replace Usage dialog box, with the full path to the encountered occurrence in the title bar:



Do one of the following:

- To have the selected occurrence replaced, click **Replace**.
- To preserve the selected occurrence and move to the next one, click **Skip**.
- To have all the occurrences of the search string in the currently active tab replaced, click **Replace All in This File**.
- To preserve the occurrences of the search string in the currently active tab (any) and move to the next file, click **Skip to Next File**.
- To have all the detected occurrences replaced, click **All Files**.
- To switch to the manual mode, click **Preview**. The Replace Usage dialog box closes and the focus moves to the Find tool window. Do one of the following:
 - Browse through the list of detected occurrences, select the ones you want to replace and then click **Replace Selected**.
 - To have all the occurrences changed click **Replace All**.

Find Usages

Edit | Find | Find Usages Settings

Alt+F7

The dialog also opens when you click  in the Show Usages pop-up window which lists all the occurrences of the symbol at caret.

Use this dialog to configure the search procedure and scope when looking for occurrences of the following data:

- Fields, variables and parameters
- Classes, tags, attributes, and references in the HTML, XML, and CSS files
- Symbols at caret.

The search results are displayed in the [Find tool window](#).

ItemDescription

Skip results tab with one usage	Select this check box to be navigated directly to the found usage without the Find tool window displayed, when only one usage is found.
Scope	Specify the search scope . Select a pre-defined scope from the drop-down list or click  to define a custom scope in the Scopes dialog .
Open in new tab	Select this check box to have the results of each search shown on a separate tab of the Find tool window. If the check box is not selected, the search results will be shown on the current tab.

Generate Instance Document from Schema Dialog

Tools | XML Actions | Generate XML Document from XSD Schema

In this dialog box, specify the options for generating an XML file based on the XSD (XML Schema Definition) [Schema](#) that describes its structure.

The menu item and the dialog box are available when the file opened in the active editor tab contains an XML Schema.

ItemDescription

Schema path	<p>In this field, specify the location of the <code>.xsd</code> Schema file to be used as the basis for XML document generation.</p> <p>By default, the field shows the full path to the current file. To use another Schema, click the Browse button  and select the desired file in the dialog that opens.</p>
Instance document name	<p>In this text box, specify the name of the output XML file to be generated. By default, the generated XML file will inherit the name of the source Schema and will have the <code>.xml</code> extension. If you type another name for the document to be generated, make sure the extension is correct.</p> <p>The default location for the generated document is the directory where the source <code>.xsd</code> Schema file resides. To specify another location, click the Browse button  and select the desired path in the dialog that opens.</p>
Element name	<p>In this drop-down list, specify the local name of the global element to be used as the root of the generated document.</p>
Enable restriction check	<p>Select this check box to have PyCharm take restriction particles into consideration (if the specified Schema uses any).</p>
Enable unique check	<p>Select this check box to have PyCharm take uniqueness particles into consideration (if the specified Schema uses any).</p>
Status	<p>View the information in this read-only field to track and improve discrepancies when configuring the generation procedure.</p>

Import File Dialog

This dialog opens when [importing delimiter-separated values \(e.g. CSV, TSV\) into a database](#).

In the left-hand part, specify how your delimiter-separated values should be converted into table format. In the right-hand part, specify the settings for the target table in your database.

- [Conversion settings](#)
- [Table name, structure and data mappings](#)
- [Data and DDL previews](#)
- [Encoding, Write errors to file and Insert inconvertible values as null](#)

Conversion settings

When working on the conversion settings, use the table preview in the right-hand part of the dialog underneath the table settings.

ItemDescription

Formats	Select your file format and check the table preview. If you haven't achieved the desired result yet, adjust the settings.  If you have changed the settings and want to save the changes, click this icon and select one of the following: <ul style="list-style-type: none">- Save Changes. The settings are saved "under the same name", without creating a new format. (A format, in fact, is a named set of settings.)- Save As. The settings are saved "under a different name": a new format is created and you can specify the name for that new format.
Value separator	Select or type the character used for separating individual values.
Row separator	Select or type the character that should be treated as a row separator.
Null value text	The text to be used as a value if a cell contains <code>null</code> (an unknown value).
Add row prefix/suffix	Row prefix and suffix are character sequences which in addition to the row separator indicate the beginning and end of a row. If necessary, click the link and specify the row prefix and suffix in the fields that appear.
Quotation	Each line in the area under Quotation is a quotation pattern (see Quote values). A quotation pattern includes: <ul style="list-style-type: none">- The left quotation character, the one inserted before a value.- The right quotation character, the one inserted after a value; usually, the same as the left quotation character.- An escape method or character for the cases when the quotation character is part of a value. E.g. Escape: duplicate means that if a quotation character occurs within a value, it is doubled. (You can specify your own escape character instead.) <p>If there is more than one pattern, the first of the patterns is used.</p> <p>Use , ,  and  to create, delete and reorder the patterns.</p> <p>To start editing an existing pattern, just click the pattern of interest.</p>
Quote values	Specify in which cases the values should be quoted (i.e. enclosed within quotation characters). <ul style="list-style-type: none">- When needed. A value is quoted only if it contains the value and/or the row separator.- Always. Any value is quoted in its text representation.
Trim whitespaces	If this check box is not selected, the Unicode whitespace characters that precede and follow the value separators are treated as parts of the corresponding values. If this check box is selected, the corresponding whitespace characters are ignored or removed.
First row is header	If this check box is selected, the first row is treated as containing column names. The settings that appear under Header Format have the same meanings as the ones above but are applied to the first row.
First column is header	If this check box is selected, the first column is treated as containing row names.

Table name, structure and data mappings

ItemDescription

Table	The name of the table.
Comment	The table comment.
Columns / Keys etc.	Data mappings for columns, and the definitions of the columns, constraints and indexes. <ul style="list-style-type: none">- To start editing the information for a column, double-click the corresponding line in the list of columns. Note the Mapped to field. This field lets you specify which data column in the file being imported should be used as a source of data for the corresponding column in the database. If you clear this field, no data will be added to the target column in the database.- To remove a column, select the corresponding line and click .

Data and DDL previews

ItemDescription

Data preview	The preview of data you are about to import.
DDL preview	The statement or statements that will be run. You can edit the statements right in the preview pane.

Import File dialog when called from the table editor

This dialog opens when [exporting delimiter-separated values from the table editor to a database](#).

- [Table name, structure and data mappings](#)
- [DDL preview](#)
- [Encoding, Write errors to file and Insert inconvertible values as null](#)

Table name, structure and data mappings

ItemDescription

Table	The name of the table.
Comment	The table comment.
Columns / Keys etc.	Data mappings for columns, and the definitions of the columns, constraints and indexes. <ul style="list-style-type: none">- To start editing the information for a column, double-click the corresponding line in the list of columns. Note the Mapped to field. This field lets you specify which data column in the file being imported should be used as a source of data for the corresponding column in the database. If you clear this field, no data will be added to the target column in the database.- To remove a column, select the corresponding line and click .

DDL preview

The statement or statements that will be run. You can edit the statements right in this pane.

Encoding, Write errors to file and Insert inconvertible values as null

ItemDescription

Encoding	The character encoding for your data in the source file.
Write errors to file	If you select the check box, the lines that cannot be imported are written to the specified text file.
Insert inconvertible values as null	If you select the check box, NULLs will be inserted into the table for the data that cannot be converted.

Import Table dialog

This dialog opens when exporting data from one table to another one, or when exporting a table to another database or schema.

Specify the data mapping info and the settings for the destination table.

- [Table name, structure and data mappings](#)
- [Data and DDL previews](#)
- [Write errors to file and Insert inconvertible values as null](#)

Table name, structure and data mappings

ItemDescription

Table	The name of the destination table.
Comment	The table comment.
Columns / Keys etc.	Data mappings for columns, and the definitions of the columns, constraints and indexes. <ul style="list-style-type: none">– To start editing the information for a column, double-click the corresponding line in the list of columns. Note the Mapped to field. This field lets you specify which column in the table being imported should be used as a source of data for the corresponding column in the destination table. If you clear this field, no data will be added to the target column.– To remove a column, select the corresponding line and click .

Data and DDL previews

ItemDescription

Data preview	The preview of data you are about to import. This tab is not available if you started the import by using the Export to Database command  in the table editor or in the Result pane of the database console.
DDL preview	The statement or statements that will be run. You can edit the statements right in the preview pane.

Write errors to file and Insert inconvertible values as null

ItemDescription

Write errors to file	If you select the check box, the data that cannot be imported are written to the specified text file.
Insert inconvertible values as null	If you select the check box, NULLs will be inserted into the table for the data that cannot be imported.

Login to IntelliJ Configuration Server Dialog

- [Before you start](#)
- [Overview](#)
- [Controls](#)

Before you start

Install and **enable** the IntelliJ Configuration Server plugin. The plugin is not bundled with PyCharm, but it can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).

Overview

IntelliJ Configuration Server is intended for storing common IDE settings to be used by a team. This ensures that all the team members have unified IDE look and feel.

The dialog box opens:

- When you start up the PyCharm for the first time after installing the plugin. In this case, the dialog box contains an additional area [On next startup](#).
- When you click the IntelliJ Configuration Server Status button  on the Status bar during a session and then click the Login button in the dialog box that opens.

Use this dialog box to log in to the IntelliJ Configuration Server and specify the Proxy settings to access the Server, if necessary.

Controls

ItemDescription

Login	In this text box, type your JetBrains Account login.
Password	In this text box, type your JetBrains account password.
Create or manage your JetBrains account	Click this link to create a JetBrains Account or edit your account if you already have it.
Use HTTP Proxy	Select this check box if you want to access the IntelliJ Configuration Server through a Proxy server.
Host name	In this text box, type the Proxy host name.
Port Number	In this text box, type the Proxy port number.
Proxy Authentication	Select this check box if you want to use a login and password to access the HTTP Proxy server.
Login	In this text box, type your Proxy login.
Password	In this text box, type your Proxy password.
Remember password	If this check box is selected, PyCharm remembers your Proxy password and in the future fills in the Password text box automatically when you log in.

On Next Startup Area

The area is displayed in the dialog box only when PyCharm starts up for the first time. Use this area to configure how you want to log in to IntelliJ Configuration Server in the future.

The following controls are available:

ItemDescription

Show login dialog	Select this option to have the Login to IntelliJ Configuration Server dialog box displayed during the next PyCharm startup.
Login silently	Select this option to log in to the Server without asking for login and password.
Do not login	Select this option if you do not plan to log in to the IntelliJ Configuration Server. You can reconfigure this behavior at any time during the session .
Login	Click this button to log in to the IntelliJ Configuration Server.
Skip	Click this button to start an PyCharm session without logging to the IntelliJ Configuration Server. You can log in to the Server at any time during the session .

IntelliJ Configuration Server Settings

This dialog box opens, when you click the IntelliJ Configuration Server Status icons  on the Status bar during a session.

ItemDescription

Show login dialog	Select this option to have the Login to IntelliJ Configuration Server dialog box displayed during the next PyCharm startup.
-------------------	---

Login silently	Select this option to log in to the IntelliJ Configuration Server without asking for login and password.
----------------	--

Do not login	Select this option if you do not plan to log in to the IntelliJ Configuration Server. You can reconfigure this behavior at any time during the session .
--------------	---

Login / Logout	Click this button to log in to the IntelliJ Configuration Server.
----------------	---

Map External Resource Dialog

Specify a URI and select a local XSD or DTD file for the specified URI. (If an XML file references the specified URI, it's validated according to a selected local file.)

ItemDescription

URI	Use this field to edit the URI. This may be a URL of an XSD or DTD file (e.g. http://www.example.org/xsds/example.xsd) or a namespace URI (e.g. http://www.example.org).
-----	---

Schemas	This tab lists XSD and DTD files available in your project, libraries, SDKs and PyCharm installation.
---------	---

Explorer	This tab implements a file browser .
----------	--

Non-Project Files Protection Dialog

This dialog appears when you try to edit non-project files (e.g. library sources, external sources etc.), and protects them from accidental modifications.

ItemDescription

These files do not belong to the project	This area displays a non-project file that you are trying to edit.
I want to edit this file anyway	Select this option to disable protection for the listed files.
I want to edit all files in this directory	Select this option to disable protection for the listed files and all files in the same directory.
I want to edit any non-project file in the current session	Select this option to disable protection completely.

All options are effective during current session, once the IDE restarted, protection will be reenabled.

Open Task Dialog

Tools | Task | Open Task+

Tools | Task | Switch Task - Open Task+

Tasks combo on the main toolbar | Open Task +

Shift+Alt+T

Use this dialog box to create a new task or specify the name of an existing task to open. Optionally, specify whether the PyCharm's behavior related to opening or creating a task, changes.

On this page:

- [Suggestion list](#)
- [Open Task dialog](#)

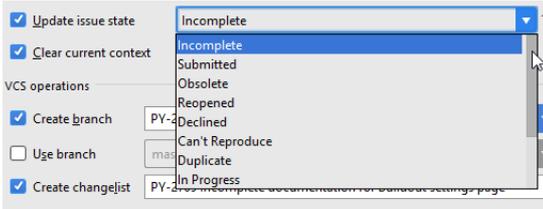
Suggestion list

ItemDescription

Enter task name	In this text box, type the name of the new task or the name of an existing task to open.
Include closed tasks	Select this check box to include closed tasks into the suggestion list.
	Click this button to open the Servers page of the Settings dialog box.

Open Task dialog

ItemDescription

Open task: <task name>	This read-only field shows the name of the task selected from the suggestion list .
Update issue state	Select this check box to choose the task state from the drop-down list: 
Clear current context	Select this check box to have the context associated with the current task cleared, when the new/selected task is opened.
VCS operations	Note This section only appears in the dialog box, when version control is enabled in your project.
Create branch <branch name> from <base branch name>	Select this check box to create a new branch from the existing one.
Use branch	Select this check box to use the existing branch.
Create changelist	Select this check box to have PyCharm create a new changelist for the specified task. By default, the text field shows the issue tracker item's description.

Optimize Imports Dialog

Code | Optimize Imports

Ctrl+Alt+O

Note that this dialog box appears when you select a Optimize Imports action on the directory.

In this dialog box, specify where you want PyCharm to remove unused import statements from, in order to optimize the import procedure.

Item Description

Only VCS changed files

If this check box is selected, then reformatting will apply only to the files that have been [changed locally](#), but not yet checked in to the repository.

This check box is only available for the files under version control.

Print

File | Print

Use this dialog box to specify the text to print and configure the print layout.

In this section:

- [Scope](#)
- [Settings](#)
- [Header and Footer](#)
- [Advanced](#)

Scope

ItemDescription

File <name>	Select this option to print the file, which is currently selected in the Project view or opened in the editor.
Selected text	Select this option to print the text selected in the editor.
All files in the directory	Select this option to print all files in the current directory.
Include subdirectories	Select this check box to have the files in the subdirectories of the current directory printed as well.

Settings

In this tab, specify the basic print layout settings.

ItemDescription

Paper size	From this drop-down list, select the desired paper size.
Font	From the drop-down lists in this area, select the desired font style and size.
Show line numbers	Select this check box to have line numbers printed.
Draw border	Select this check box to have a border printed.
Orientation	In this area, specify the paper orientation. The available options are: <ul style="list-style-type: none">- Landscape- Portrait
Style	In this area, specify the style of the printout by selecting the relevant check boxes. The available options are: <ul style="list-style-type: none">- Color printing- Syntax printing- Print as graphics

Header and Footer

In this tab, specify the contents and placement of the header and footer.

ItemDescription

Text line	In this text box, specify the contents of the header or footer. If necessary, combine plain text with print keywords. By default, PyCharm suggests to print the name of a file <code>\$FILE\$</code> in the header and the current page number <code>\$PAGE\$</code> of all pages <code>\$TOTALPAGES\$</code> in the footer. The following print keywords are recognized: <ul style="list-style-type: none">- <code>\$FILE\$</code> prints fully qualified file name.- <code>\$PAGE\$</code>- <code>\$DATE\$</code>- <code>\$TIME\$</code>- <code>\$FILENAME\$</code> prints file name without path.- <code>\$TOTALPAGES\$</code>
Placement	Use this drop-down list to specify whether the above line will be printed in the header or in the footer.
Alignment	From this drop-down list, select the desired alignment.
Font	In this area, specify the desired font style and size to print the header and footer text.

Advanced

ItemDescription

Wrapping	In this area, configure text wrapping. The available options are: <ul style="list-style-type: none">- No wrap- Wrap at word breaks
Margins	Use the text boxes in this area to specify the margins in inches.

Productivity Guide

Help | Productivity Guide

The Productivity Guide shows usage statistics for PyCharm features.

The table in the upper part lists the features. Select a feature to see its description in the lower part.

To sort the information, click a cell in the header row.

ItemDescription

Feature	The name of the feature.
Group	The group to which the feature belongs.
Used	How many times you used the feature.
Last used	When the feature was last used.

PSI Viewer

Tools | View PSI Structure

Use this viewer to explore internal structure of the various files or fragments of source code, as they are interpreted by PyCharm. The dialog box is non-modal and enables you to keep on working with PyCharm while being opened.

ItemDescription

Show PSI structure for	Use this drop-down list to specify file type, or language constructs to be explored. The set of recognized file types depends on the supported languages and installed plugins.
Show PsiWhiteSpace	If this check box is selected, the generated tree view will contain <code>PsiWhiteSpace</code> nodes, corresponding to the spaces in the source code. When you select or clear this check box, the tree view of the PSI structure changes accordingly.
Show Tree Nodes	
Dialect	This drop down list becomes available for the languages that support dialects, for example, SQL, JavaScript etc.
Text	Use this pane to enter source code to be explored. PyCharm suggests the following ways to supply code: <ul style="list-style-type: none">– Type immediately within the text area.– Paste from clipboard. If you have copied some text from the editor, and then open the PSI viewer, the previous contents of the Text pane is selected, which enables you to overwrite it from the clipboard using <code>Ctrl+V</code>, or <code>Ctrl+Shift+V</code>. <p>Note that some editing features are also available: removing line at caret <code>Ctrl+Y</code>, duplicating text <code>Ctrl+D</code>, and adding line with <code>Shift+Enter</code>.</p>
PSI Structure	This read-only pane displays the PSI structure tree view, generated on clicking the Build PSI Tree button, according to the file type selected in the Show PSI structure for drop-down list. Navigating through the tree view highlights the corresponding fragments of source code in the Text pane. If currently selected tree node has references, they are also displayed in the References pane.
References	This read-only field shows references to the nodes of the PSI Structure tree view (if any). Unresolved references are shown red; the corresponding fragments of source code are also highlighted with a red frame.
Build PSI Tree	Click this button to generate PSI structure tree view of the code in Text pane, according to the file type selected in the Show PSI structure for drop-down list. If source code in the Text pane has been modified, use this button to refresh the tree view.

Pull Image Dialog

For this dialog to be available, the Docker integration plugin must be installed and enabled.

Specify the settings for pulling an image from a [Docker](#) image repository such as [Docker Hub](#) or [Quay](#).

ItemDescription

Registry	The URL of the image repository service or a Docker Registry configuration: <ul style="list-style-type: none">– If pulling an image doesn't assume logging on to the corresponding server, specify the repository service URL. E.g. registry.hub.docker.com corresponds to public repositories on Docker Hub.– Otherwise, specify the corresponding Docker Registry configuration. Click New to create such a configuration.
New	Click this button to create a Docker Registry configuration. The Docker Registry dialog will open. (A Docker Registry configuration in PyCharm represents your Docker image repository user account.)
Repository	The name of the repository (image) to be pulled, e.g. <code>centos</code> , <code>jboss/wildfly</code> .
Tag	The tag of the image to be pulled.

Push Image Dialog

For this dialog to be available, the Docker integration plugin must be installed and enabled.

Specify the settings for pushing an image to a [Docker](#) image repository such as [Docker Hub](#) or [Quay](#).

ItemDescription

Registry	Select the Docker Registry configuration to be used. Click New to create such a configuration.
New	Click this button to create a Docker Registry configuration. The Docker Registry dialog will open. (A Docker Registry configuration in PyCharm represents your Docker image repository user account.)
Repository	The name of the repository (image) you are pushing.
Tag	The tag for the repository (image) you are pushing.

PyCharm License Activation Dialog

To open this dialog: Register on the [Welcome screen](#) or Help | Register.

You can evaluate PyCharm Ultimate for 30 days. After that period, you need to buy PyCharm and activate your license.

The upper part of the dialog reflects your PyCharm usage status (e.g. Free evaluation) and, if appropriate, provides related controls (e.g. Buy PyCharm).

The license activation options are in the lower part of the dialog under Activate new license via.

– [PyCharm usage status-related controls](#)

– [License activation options](#)

PyCharm usage status-related controls

ItemDescription

Buy PyCharm	Click this button to go to the JetBrains Web site to study the PyCharm purchasing options and to buy a license.
-------------	---

Evaluate for free for 30 days	Click this button to start evaluating PyCharm.
-------------------------------	--

License activation options

ItemDescription

License key	Select this option if you have a license key. Specify your user name (or company name) and the key. (The corresponding information is in your license certificate email.)
-------------	---

The recommended way of entering the name and the key is by copying the information from the certificate email and then pasting it into the fields (

 ).

If PyCharm rejects the license information, see [The key is not accepted](#) for a possible solution.

License server	Select this option if there is the PyCharm License Server on your company's intranet. Specify the server URL in the License server address field. For the address to be entered automatically, click the Discover server button.
----------------	--

To find out if your company is using the License Server and what its URL should be, contact your system administrator.

JetBrains Account	Select this option if your PyCharm license is linked with your JetBrains Account . Specify your JetBrains Account access credentials.
-------------------	---

Recent Changes Dialog

View | Recent Changes

Shift+Alt+C

In the Recent Changes popup, use the arrow keys to move in the list, and the `Enter` key to see details and, if necessary, to revert to the previous state.

Refactoring Dialogs

All refactoring dialogs provide similar controls to preview results and perform the refactoring.

Specific options and controls are described in the following topics:

- [Copy Dialog](#)
- [Change Signature Dialog](#)
- [Change Signature Dialog for JavaScript](#)
- [Extract Dialogs](#)
- [Invert Boolean Dialog](#)
- [Move Dialogs](#)
- [Pull Members Up Dialog](#)
- [Push Members Down Dialog](#)
- [Rename Dialogs](#)
- [Rename dialog for a table or column](#)
- [Safe Delete Dialog](#)

Copy Dialog

Refactor | Copy

F5

Use this dialog to specify the settings for the [Copy refactoring](#).

ItemDescription

New name	Specify the name of the file, or directory to be created.
To directory	Specify the directory where to create a copy. Select the destination directory from the list, or click  (<code>Shift+Enter</code>) and select the directory in the <code>Select target directory</code> dialog that opens. (If necessary, you can create a new directory.)
Open copy in editor	Select this check box to automatically open a file, directory or package after it is copied. If you clear this check box, then the file, directory or package that you have copied will not be opened.

Change Signature Dialog

Refactor | Change Signature

Ctrl+F6

Use the Change Signature dialog to perform the [Change Signature](#) refactoring.

Item	Keyboard shortcut	Description
Name		Use this field to change the function name.
Parameters		
		Use this button to add a parameter.
		Use this button to delete the selected parameter.
	 	Use these buttons to reorder parameters by moving them up or down in the parameter list.
Name		Specify the parameter name.
Default value		Specify the default parameter value to be added to the function calls. (This is the value to be passed to the function in the function calls.)
Use default value in signature		Select this check box to have the default values included in the function signature. If this check box is not selected, the default values will be included in the function calls.
Signature Preview		In this area, the current function signature is shown. (The information in this area is synchronized with the changes you are making to the function signature.)
Refactor		Click this button to perform the refactoring right away.
Preview		Click this button to see the expected changes in the Find tool window prior to actually performing the refactoring.

Change Signature Dialog for JavaScript

Refactor | Change Signature

Ctrl+F6

Use this dialog to [change the function signature](#) and to perform other, related tasks.

ItemDescription

Name Use this field to modify the function name.

Use the table and the [controls](#) to the right of it to manage the function parameters and their [properties](#).

Name Use this field to specify the name of a parameter.

Value A parameter value.
If the parameter is a required one, the specified value (or expression) will be passed to the function in the function calls. If the parameter is an optional one, this value will be used in the function body to initialize the parameter.

Optional Select this check box if the parameter is optional. Your selection will define how the [parameter value](#) is used.

+ or Use this icon or shortcut to start adding a new parameter.

Alt+Insert

Specify the parameter [name](#) and [value](#). Also, specify whether the parameter is [optional](#).

Note that you can [propagate](#) the parameters you have added to the calling methods.

- or Use this icon or shortcut to delete the selected parameter.

Alt+Delete

↑ or Use this icon or shortcut to move the selected parameter one line up in the list of parameters.

Alt+Up

↓ or Use this icon or shortcut to move the selected parameter one line down in the list of parameters.

Alt+Down

🔄 Use this icon or shortcut to propagate the added parameters to the calling methods.
You can propagate new function parameters to any function that directly or indirectly calls the function whose signature you are changing.

(There may be the functions that call the current function. These functions, in their turn, may be called by other functions. You can propagate new parameters to any of the functions in such sequences.)

In the left-hand pane of the Select Methods to Propagate New Parameters dialog, expand the necessary nodes and select the check boxes next to the functions you want the new parameters to be propagated to.

Signature Preview In this area, the current function signature is shown. (The information in this area is synchronized with the changes you are making to the function signature.)

Refactor Click this button to perform the refactoring right away.

Preview Click this button to see the expected changes prior to actually performing the refactoring.

Extract Dialogs

In this section:

- [Extract Constant Dialog](#)
- [Extract Field Dialog](#)
- [Extract Method Dialog](#)
- [Extract Parameter Dialog for JavaScript](#)
- [Extract Parameter Dialog](#)
- [Extract Superclass Dialog](#)
- [Extract Variable Dialog](#)

Extract Constant Dialog
Refactor | Extract Constant

Ctrl+Alt+C

ItemDescription

Name	Specify the name of the new constant.
Replace all occurrences	Select this check box to automatically replace all the occurrences of the selected expression (if the selected expression is found more than once in the method).

Extract Field Dialog

Extract Field refactoring declares a new field and initializes it with the selected expression. The original expression is replaced with the usage of the field.

- [Example](#)
- [Extracting a field in-place](#)
- [Extracting a field using the dialog box](#)

Example

Before/After

```
import math

class Solver:
    def roots(self):
        a = 3
        b = 25
        c = 46
        root1 = (-b + math.sqrt(b**2 - 4*c)) / (2*a)
        root2 = (-b - math.sqrt(b**2 - 4*c)) / (2*a)
        print (root1, root2)

Solver().demo()
```

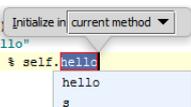
```
import math

class Solver:
    def demo(self):
        a = 3
        b = 25
        c = 46
        self.return_type_of_sqrt = math.sqrt(b ** 2 - 4 * a * c)
        root1 = (-b + self.return_type_of_sqrt) / (2*a)
        root2 = (-b - self.return_type_of_sqrt) / (2*a)
        print(root1,root2)

Solver().demo()
```

```
def print_hello(self):
    s = "Hello, World!"
    print(s)
```

```
def print_hello(self):
    self.hello = "Hello"
    s = "%s, World!" % self.hello
    print(s)
```

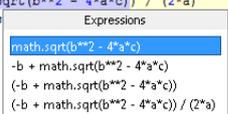


To extract a field in-place

The [in-place refactorings](#) are enabled in PyCharm by default. So, if you haven't changed this setting, the Introduce Field refactorings are performed in-place, right in the editor:

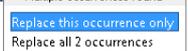
1. Place the cursor within the expression or declaration of a variable to be replaced by a field.
2. Do one of the following:
 - Press `Ctrl+Alt+F`.
 - Choose Refactor | Introduce Field on the main menu, or on the context menu.
3. If more than one expression is detected for the current cursor position, the Expressions list appears. If this is the case, select the required expression. To do that, click the expression. Alternatively, use the `Up` and `Down` arrow keys to navigate to the expression of interest, and then press `Enter` to select it.

```
c = 46
root1 = (-b + math.sqrt(b**2 - 4*a*c)) / (2*a)
root2 = (-b - math.s
print(root1,root2)
```



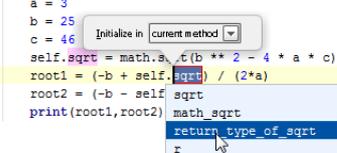
4. If more than one occurrence of the expression is found within the class, specify whether you wish to replace only the selected occurrence, or all the found occurrences with the new constant:

```
c = 46
root1 = (-b + math.sqrt(b**2 - 4*a*c)) / (2*a)
root2 = (-b - math.sq
print(root1,root2)
```



5. If relevant, specify where the new field will be initialized - in the current method, or in a class constructor.
6. Specify the name of the field. Select the name from the list or type the name in the box with a red border.

```
class SolverEquation:
    def demo(self):
        a = 3
        b = 25
        c = 46
        self.sqrt = math.a
        root1 = (-b + self.sqrt) / (2*a)
        root2 = (-b - self.sqrt
        print (root1,root2)
```



7. To complete the refactoring, press `Tab` or `Enter`.
- If you haven't completed the refactoring and want to cancel the changes you have made, press `Escape`.

Note that sometimes you may need to press the corresponding key more than once.

```

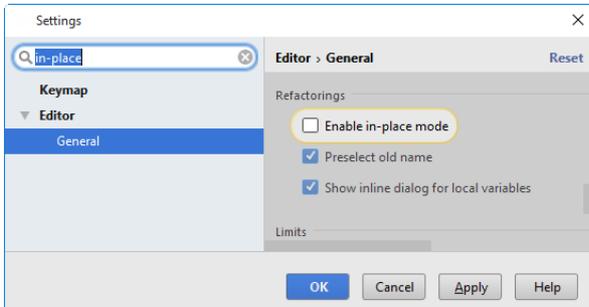
import math

__author__ = 'wombat'
class SolverEquation:
) def demo(self):
    a = 3
    b = 25
    c = 46
    self.return_type_of_sqrt = math.sqrt(b ** 2 - 4 * a * c)
    root1 = (-b + self.return_type_of_sqrt) / (2*a)
    root2 = (-b - self.return_type_of_sqrt) / (2*a)
    print(root1,root2)
)

```

To extract a field using the dialog box

If the Enable in place refactorings check box is cleared on the Editor settings, the Extract Field refactoring is performed by means of the [Extract Field Dialog](#).



1. In the editor, select the expression or variable to be replaced with a field, or just place the cursor within such an expression or variable declaration.
2. In the main menu, or the context menu of the selection, choose Refactor | Extract | Field, or press `Ctrl+Alt+F`.
3. In the Expressions pop-up menu, select the expression to be replaced. Note that PyCharm highlights the selected expression in the editor.
4. In the dialog that opens, specify the type and name of the new field.
5. In the Initialize in section, specify where the new field will be initialized.
6. To automatically replace all occurrences of the selected expression (if it is found more than once), select the option Replace all occurrences.
7. Click OK to create the field.

Ctrl+Alt+M

In the JavaScript context, the title of the dialog box may change to Extract Function.

ItemDescription

Name	In this text box, specify the name of the function or method to be generated on the basis of the selected source code.
Parameters	In this area, select parameters to be passed to the new method/function.  1. If a parameter that is critical for the functionality of the new method is not selected, PyCharm will be unable to proceed with the refactoring.
Move Up/Down	Use these buttons to change the order of the parameters. 
Signature preview	In this read-only field, view the declaration of the new method/function.
Output variables	This read-only field shows local variables or parameters that have been changed within the method body, and will be returned by the method.

Ctrl+Alt+P

Use this dialog to specify the options and settings related to the [Extract Parameter refactoring in JavaScript](#).

ItemDescription

Type	Specify the type of the new parameter. Usually, you don't need to change the type suggested by PyCharm.
Name	Specify the name for the new parameter.
Value	The expression to be replaced with the new parameter. Initially, this field contains the expression that you have selected. Normally, this initial value does not need to be changed.
Optional parameter	If this option is selected, the new parameter is assigned a value in the function body. The value corresponds to that currently set in the Value field. The calls to the function do not change. If this option is not selected, all the function calls change according to the new function signature. The value specified in the Value field is added to the function calls and thus is passed to the function. No explicit value assignment for the new parameter is added to the function body.
Replace all occurrences	Select this option to replace all the occurrences of the selected expression within the function.

Ctrl+Alt+P

Specify the settings for extracting a parameter.

ItemDescription

Name	Specify the name for the new parameter.
Replace all occurrences	Select this option to replace all the occurrences of the selected expression within the method.

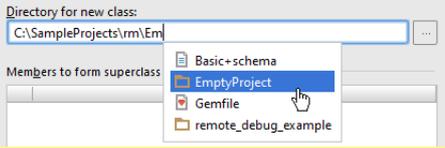
ItemDescription

Extract superclass from This read-only field displays the name of the class, from which a superclass should be extracted.

Superclass name In this text field, type the Superclass name.

Directory for new class In this text field, specify the path to the target directory, where the new class will be stored.

Tip Use code completion while you type the path:



The screenshot shows a dialog box with a text field for the directory path containing "C:\SampleProjects\rm\Em" and an ellipsis button. Below it is a list titled "Members to form superclass" with four items: "Basic+schema", "EmptyProject", "Gemfile", and "remote_debug_example". A mouse cursor is pointing at "EmptyProject".

You can also click the ellipsis button, or press **Shift+Enter**, and select the desired path in the Select Path dialog box.

Members to form superclass This area displays the list of members detected in the original class. Select the check boxes next to the members to be included in the new superclass.

Extract Variable Dialog
Refactor | Extract Variable

Ctrl+Alt+V

Use this dialog box to extract:

- Variables in Python or JavaScript contexts.
- Local variables for JavaScript 1.7 and higher.
- JavaScript constants.

Item	Description	Language
Name	Specify the name for the new variable.	All
Replace all occurrences	Select this option to replace all the occurrences of the selected expression (if more than one occurrence of the selected expression is found).	All
Var kind	Choose to extract a (global) variable, a constant, or a local variable: <ul style="list-style-type: none">- extract var - select this option to extract a global variable.- extract constant - select this option to extract a constant.- extract local variable - select this option to extract a local variable. The extract local variable option is available only for JavaScript 1.7 or a later version.	JavaScript

Invert Boolean Dialog

Refactor | Invert Boolean

Use this dialog to flip the Boolean value of a variable and its usages. The command becomes available, when the caret is at the variable name.

ItemDescription

Name of the inverted variable

This field shows the name of the variable at caret.

Move Dialogs

Refactor | Move

F6

In this section:

- [Move Class/Function Dialog](#)
- [Move File Dialog](#)
- [Move Module Members Dialog](#)



The Move Class or Move Function refactoring dialog box is invoked for the classes and functions in the Python files, selected in the Project view, or opened in the editor. The import statements are updated accordingly.

ItemDescription

Move class/function This read-only field shows the name of the class or function to be moved, in the format `<file name>.<class name>`, or function `<function name>`.

To file Specify the fully-qualified path to the target file, where the selected class or function should be moved. Click the browse button  to find the desired location.



The Refactor | Move menu command invokes one of the following dialog boxes depending on the selected item to be moved:

- The Move File dialog box is invoked when a file is selected in the [Project](#) tool window or opened in the editor.
- The Move File dialog box is invoked when a file opened in the editor.

ItemDescription

To package	Specify the destination package. Type the path manually or click  and select the target package in the dialog that opens .
Search in comments and strings	Select the check box to find and update the comments and strings to the file being moved.
Open moved in editor	Select this check box to see the moved file in the editor.

F6

This dialog box appears only for the top-level symbols. Using this dialog, you can:

- Select and move several members at once. So doing, dependent symbols are highlighted.
- Select several symbols in the editor; so doing, they will be pre-selected in the dialog.
- Besides classes and functions, move the global variables, defined in a module on the top-level.

ItemDescription

From	This read-only field shows the fully-qualified path to the original file.
To	In this field, specify the fully-qualified path to the destination file. Click  to locate the target file in the Select Path dialog. If the destination file does not yet exist, it will be created.
Bulk move	This section shows the list of top-level members. Each member has a check box to the left. If several members get selection in the editor, they will become pre-selected in the list. To hide the list of members, click  .

Pull Members Up Dialog

Refactor | Pull Members Up

Use this dialog to pull selected members up to the selected superclass.

ItemDescription

Pull Up Members of <class_name> to

Select the destination object (superclass).

Members to be pulled up

In this area, select the members you want to move.

Make abstract

Select this check box to move the selected method as abstract.

Push Members Down Dialog

Refactor | Push Members Down

Use this dialog to push selected members down to immediate subclass.

ItemDescription

Members to be pushed down

In this area, select the members you want to move.

Rename Dialogs

Refactor | Rename

Shift+F6

ItemDescription

Rename <symbol name> and its usages to	In this field, specify a new name for the symbol.
Search in comments and strings	Select this check box to have the changes applied to comments and strings.
Search for text occurrences	Select this check box to have the changes applied to the documentation, HTML, and other files included in the project.
Rename inheritors	Select this check box to have the changes applied to the entire hierarchy. This option is available when renaming method parameters, or Python classes.
Rename containing file	Select this check box to have the changes applied to the containing file. This option is only available when the class name coincides with the name of the containing file.

Rename dialog for a table or column

Refactor | Rename

Shift+F6

Use this dialog to rename a table or column.

In addition to renaming the table or column itself, PyCharm can also look for the usages of the table or column name. If found, the changes you are making to the table or column name can also be applied to these usages.

ItemDescription

Rename <table or column> and its usages to	Specify a new name for the table or column.
Search in comments and strings	If this check box is selected, PyCharm will look for occurrences of the table or column name within comments and string literals in your source code files.
SQL Script	The statement to be run to rename the table or column. If necessary, you can edit the statement right in this pane.
Refactor	Execute the statement and make associated changes right away.
Preview	Preview potential associated changes prior to executing the statement. See Previewing changes .
Rename inheritors	Select this check box to have the changes applied to the entire hierarchy. This option is available when renaming method parameters, or Python classes.
Rename containing file	Select this check box to have the changes applied to the containing file. This option is only available when the class name coincides with the name of the containing file.

Safe Delete Dialog

Refactor | Safe Delete

Alt+Delete

Use this dialog to define the scope of the [Safe Delete](#) refactoring.

ItemDescription

Search in comments and strings Select this check box to display the usages of the specified symbol in comments and strings in the Refactoring Preview tool window.

Search for text occurrences Select this check box to apply the changes text files within the project (such as documentation, HTML, JSP, etc.)

Reformat File Dialog

Code | Reformat Code

Ctrl+Alt+L

The `Ctrl+Alt+L` command will try to reformat the source code of the specified scope automatically. This dialog box appears when you press `Ctrl+Shift+Alt+L` in the editor of the current file.

ItemDescription

Only VCS changed text	If this check box is selected, then reformatting will apply only to the files that have been changed locally , but not yet checked in to the repository. This check box is only available for the files under version control.
Selected text	Choose this option to have the currently selected fragment of source code reformatted.
Whole file	Choose this option to have all the source code in the current file reformatted.
Optimize imports	Select this check box to remove unused import statements from the code within the selected scope.
Rearrange code	Select this check box to reorder your source code entries according to the configurations specified in the Arrangement tab of your Code Style settings .
Run	Click this button to start reformatting the source code within the specified scope.

Register New File Type Association Dialog

The dialog box opens when you attempt to open a file in the editor, but PyCharm does not recognize its type by the file extension. Use the dialog box to associate unknown file extensions with existing file types.

ItemDescription

Open matching files in PyCharm	When this option is selected, PyCharm treats the type of the file to be opened as one of the recognized file types. Choose the relevant type from the list box below, that displays all the file types recognized by PyCharm.
File Pattern	In this text box, specify the file pattern to be associated with the selected file type. By default, the text box shows the following pattern: <code><current file full extension>.*</code> .
Open matching files in associated application	When this option is selected, PyCharm attempts to open the selected file using its native application, if this application is available.

Tip You can [change the association](#) later in the [File Types](#) settings page.

Run/Debug Configurations

Run | Edit Configurations

Shift+Alt+F10

Shift+Alt+F9

Use this dialog box to create, edit, or remove [run/debug configurations](#), and configure the [default settings](#) that will apply to all newly created run/debug configurations.

The default settings are grouped under the Defaults node in the left-hand part of the dialog box.

Click [here](#) for the description of the options that are common for all run/debug configurations.

For some of the run/debug configurations, PyCharm supports viewing output in Log files. Find the detailed information in the corresponding run/debug configuration dialogs.

Find the descriptions of specific run/debug configurations in the following pages:

- [Run/Debug Configuration: App Engine Server](#)
- [Run/Debug Configuration: Behave](#)
- [Run/Debug Configuration: Chromium Remote](#)
- [Run/Debug Configuration: Compound](#)
- [Run/Debug Configuration: Django Server](#)
- [Run/Debug Configuration: Django Test](#)
- [Run/Debug Configuration: Docker Deployment](#)
- [Run/Debug Configuration: Firefox Remote](#)
- [Run/Debug Configuration: Grunt.js](#)
- [Run/Debug Configuration: Gulp.js](#)
- [Run/Debug Configuration: JavaScript Debug](#)
- [Run/Debug Configuration: Jest](#)
- [Run/Debug Configuration: JSTestDriver](#)
- [Run/Debug Configuration: Lettuce](#)
- [Run/Debug Configuration: Mocha](#)
- [Run/Debug Configuration: Meteor](#)
- [Run/Debug Configuration: Node JS](#)
- [Run/Debug Configuration: Node JS Remote Debug](#)
- [Run/Debug Configuration: NodeUnit](#)
- [Run/Debug Configuration: NPM](#)
- [Run/Debug Configuration: PhoneGap/Cordova](#)
- [Run/Debug Configuration: Protractor](#)
- [Run/Debug Configuration: Python](#)
- [Run/Debug Configuration: Python Remote Debug](#)
- [Run/Debug Configurations: Python Docs](#)
- [Run/Debug Configuration: Pyramid Server](#)
- [Python Tests](#)
- [Run/Debug Configuration: Tox](#)

Toolbar

ItemShortcutDescription

		Click this button to add a new configuration to the list.
		Click this button to remove the selected configuration from the list.
		Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
	 or 	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the  buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

ItemDescription

Name	In this text box, specify the name of a new run/debug configuration. This field is not available for the default run/debug configurations .
Defaults	This node in the left-hand pane contains the default run/debug configuration settings. Select configuration to modify its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. The shared run/debug configurations are kept in separate xml files under <code>.idea\runConfigurations</code> folder, while the local run/debug configurations are kept in the <code>.idea\workspace.xml</code> . This check box is not available when editing the run/debug configuration defaults.
Before launch	Use the controls in this area to specify which tasks must performed before applying the run/debug configuration. The tasks are performed in the order they appear in the list. The following options are available: <ul style="list-style-type: none">-  (<code>Alt+Insert</code>): click this icon to add a task to the list. Select one of the following task types:<ul style="list-style-type: none">- Run External tool: select this option to run an application that is external to PyCharm. In the dialog that opens, select the application(s) that you want to run. If the required application is not defined in PyCharm yet, add its definition (for details see Configuring Third-Party Tools and External Tools).- Run Another Configuration: select this option to execute one on the existing run/debug configuration. Choose a configuration to execute from the dialog box that opens.- Compile TypeScript: select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:<ul style="list-style-type: none">- If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.- If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.- Run File Watcher: select this option to run the File Watchers that are active in the current project. For more information, see Using File Watchers.- Generate CoffeeScript Source Maps: select this option to generate source maps for your CoffeeScript sources. In the dialog that opens, specify the path to the folder where CoffeeScript files will be generated. For more information, see CoffeeScript Support.- Run Remote External tool: select this option to run a remote SSH external tool. In the dialog that opens, specify the SSH external tool that you want to run. For more information, see Remote SSH External Tools.-  (<code>Alt+Delete</code>): click this icon to remove the selected task from the list.-  (<code>Enter</code>): click this icon to modify the selected task. Edit the task settings in the dialog that opens.-  (<code>Alt+Up</code>): click this icon to move the selected task up in the list.-  (<code>Alt+Down</code>): click this icon to move the selected task down in the list.- Show this page: select this check box if you want to display the run/debug configuration settings before applying it.- Active tool window: select this option if you want the Run/Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

ItemDescription

Confirm rerun with process termination	The behavior of this button depends on selecting the check box Single instance only for a particular run/debug configuration. <ul style="list-style-type: none">- If this check box is selected, then, in case of a single instance, launching a new process (for example, by clicking  on the main toolbar) while the other process is still running, results in showing a dialog box, where one should confirm termination of the current process and launching a new one.- If this check box is not selected (or in case of multiple instances), PyCharm starts the new process silently.
Temporary configurations limit	Specify here the maximum number of temporary configurations to be stored and shown in the Select Run/Debug Configuration drop-down list.

Run/Debug Configuration: App Engine Server

This feature is supported in the Professional edition only.

Use this dialog box to create a run/debug configuration for App Engine server.

In this section:

- [Configuration tab](#)
- [Logs tab](#)
- [Toolbar](#)
- [Common options](#)

Configuration tab

ItemDescription

Port	In this text box, specify the port number where the server will start.
Additional options	In this field, type the additional options to be passed to the server. Refer to the following resources for details: <ul style="list-style-type: none">– Google App Engine command line arguments– Django server command line arguments
Run browser	Select this check box, if you want your Google App Engine application to open in the default browser. In the text field below, enter the IP address where your application will be opened.
Environment	
Project	Click this drop-down list to select one of the projects, opened in the same PyCharm window , where this run/debug configuration should be used. If there is only one open project, this field is not displayed.
Environment variable	This field shows the list of environment variables. If the list contains several variables, they are delimited with semicolons. To fill in the list, click the browse button, or press <code>Shift+Enter</code> and specify the desired set of environment variables in the Environment Variables dialog box. To create a new variable, click <code>+</code> , and type the desired name and value.
Python Interpreter	Select one of the pre-configured Python interpreters from the drop-down list. Note that you can select a remote interpreter , as well as the local one. If a remote interpreter is selected, you have to specify path mappings in the corresponding field that appears .
Interpreter options	In this field, specify the string to be passed to the interpreter. If necessary, click  , and type the string in the editor.
Working directory	Specify a directory to be used by the running task. <ul style="list-style-type: none">– When a default run/debug configuration is created by the keyboard shortcut <code>Ctrl+Shift+F10</code>, or by choosing Run on the context menu of a script, the working directory is the one that contains the executable script. This directory may differ from the project directory.– When this field is left blank, the <code>bin</code> directory of the PyCharm installation will be used.
Path mappings	This field appears, if a remote interpreter has been selected in the field Python interpreter. Click the browse button  to define the required mappings between the local and remote paths. In the Edit Path Mappings dialog box, use <code>+</code> / <code>-</code> buttons to create new mappings, or delete the selected ones.
Add content roots to PYTHONPATH	Select this check box to add all content roots of your project to the environment variable PYTHONPATH;
Add source roots to PYTHONPATH	Select this check box to add all source roots of your project to the environment variable PYTHONPATH;
Docker container settings	<div style="background-color: #ffff00; padding: 5px;"><p>Warning! This field only appears when Docker-based remote interpreter has been selected for a project.</p></div> <div style="background-color: #ffff00; padding: 5px; margin-top: 5px;"><p>Note Speaking about the correspondence of settings with some options (<code>--net</code>, <code>--link</code>, etc.), note that these options come from Docker command line arguments.</p></div> <p>Click  to open the dialog and specify the following settings:</p> <ul style="list-style-type: none">– Disable networking: select this check box to have the networking disabled. This corresponds to <code>--net="none"</code>, which means that inside a container the external network resources are not available.– Network mode: corresponds to the other values of the option <code>--net</code>.<ul style="list-style-type: none">– <code>bridge</code> is the default value. An IP address will be allocated for container on the bridge's network and traffic will be routed through this bridge to the container. <p>Containers can communicate via their IP addresses by default. To communicate by name, they must be linked.</p> <ul style="list-style-type: none">– <code>host</code>: use the host's network stack inside the container.– <code>container:<name id></code>: use the network stack of another container, specified via its name or id. <p>Refer to the Network settings documentation for details.</p> <ul style="list-style-type: none">– Links: Use this section to link the container to be created with the other containers. This is applicable to <code>Network mode = bridge</code> and corresponds to the <code>--link</code> option.– Publish all ports: If the check box is selected, all the container's exposed ports are bound to a random host port. <p>See e.g. EXPOSE (incoming ports) in Docker run reference. This corresponds to the option <code>--publish-all</code>.</p>

- Port bindings: Use this field to specify the Container port/protocol - Host IP address/port mappings. E.g. `8080 tcp 127.0.0.1 18080` would bind the TCP port `8080` inside the container to port `18080` on the `localhost` or `127.0.0.1` interface on the host machine.

For more info, see e.g. [Connect using network port mapping](#) in [Legacy container links](#).

- Extra hosts: This corresponds to the `--add-host` option. Refer to the page [Managing /etc/hosts](#) for details.
- Volume bindings: Use this field to specify the bindings between the special folders-volumes and the folders of the computer, where the Docker daemon runs. This corresponds to the `-v` option.

See [Managing data in containers](#) for details.

- Environment variables: Use this field to specify the list of environment variables and their values. This corresponds to the `-e` option. Refer to the page [ENV \(environment variables\)](#) for details.

Click  to expand the tables. Click ,  or  to make up the lists.

Logs tab

Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the [Run](#) or [Debug tool window](#).

ItemDescription

Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window .
Log File Entry	The read-only fields in this column list the log files to show. The list can contain: <ul style="list-style-type: none"> – Full paths to specific files. – Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.
Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point to the desired location in the dialog that opens .
Show console when a message is printed to standard output stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.
Show console when a message is printed to standard error stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.
	Click this button to open the Edit Log Files Aliases dialog where you can select a new log entry and specify an alias for it.
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog .
	Click this button to remove the selected log entry from the list.
	Click this button to edit the select log file entry. The button is available only when an entry is selected.

Toolbar

ItemShortcutDescription

		Click this button to add a new configuration to the list.
		Click this button to remove the selected configuration from the list.
		Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
	 or 	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the  buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

ItemDescription

Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. The shared run/debug configurations are kept in separate xml files under <code>.idea\runConfigurations</code> folder, while the local run/debug configurations are kept in the <code>.idea\workspace.xml</code> .

This check box is not available when editing the run/debug configuration defaults.

Single instance only If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, PyCharm checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place.

This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources.

If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.

Before launch Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
------	-------------------	-------------



Alt+Insert

Click this icon to add a task to the list. Select the task to be added:

- Run External tool. Select this option to run an application which is external to PyCharm. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in PyCharm yet, add its definition. For more information, see [Configuring Third-Party Tools](#) and [External Tools](#).
- Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project.
- Run File Watchers. Select this option to have PyCharm apply all the currently active file watchers, see [Using File Watchers](#) for details.
- Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package.
- Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package.
- Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it.
- Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:
 - If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.
 - If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.
- Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see [CoffeeScript Support](#).
- Run Remote External tool: Add a remote SSH external tool. Refer to the section [Remote SSH External Tools](#) for details.
- Upload files to Remote Host. Select this option to have the application files automatically uploaded to the server according to the [default server access configuration](#). For more information, see [Configuring Synchronization with a Web Server](#) and [Uploading and Downloading Files](#).



Alt+Delete

Click this icon to remove the selected task from the list.



Enter

Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.



Alt+Up

Click this icon to move the selected task one line up in the list.



Alt+Down

Click this icon to move the selected task one line down in the list.

Show this page

Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.

Active tool window

Select this option if you want the [Run/Debug](#) tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Behave

This feature is supported in the Professional edition only.

Use this dialog box to create a run/debug configuration for Behave tests.

In this section:

- [Configuration tab](#)
- [Logs tab](#)
- [Toolbar](#)
- [Common options](#)

Configuration tab

ItemDescription

Feature files or folders	<p>In this text field, type the fully-qualified names of the feature files or directories which contain feature files.</p> <p>Multiple names should be delimited with .</p> <p>Use the browse button to locate the desired paths in the file system.</p>
Params	<p>In this text field, type the Behave-specific parameters to be passed to the tests. PyCharm provides the possibility to pass parameters to the test runner.</p> <p>In particular, the Behave parameters are described in the Tag expressions section of the Behave documentation.</p>
Scenario	<p>Type the name of the scenario to be executed. If this field is left blank, all the available scenarios in the specified feature files will be executed.</p>
Environment	
Project	<p>Click this drop-down list to select one of the projects, opened in the same PyCharm window, where this run/debug configuration should be used. If there is only one open project, this field is not displayed.</p>
Environment variable	<p>This field shows the list of environment variables. If the list contains several variables, they are delimited with semicolons.</p> <p>To fill in the list, click the browse button, or press <code>Shift+Enter</code> and specify the desired set of environment variables in the Environment Variables dialog box.</p> <p>To create a new variable, click <code>+</code>, and type the desired name and value.</p>
Python Interpreter	<p>Select one of the pre-configured Python interpreters from the drop-down list.</p> <p>Note that you can select a remote interpreter, as well as the local one. If a remote interpreter is selected, you have to specify path mappings in the corresponding field that appears.</p>
Interpreter options	<p>In this field, specify the string to be passed to the interpreter. If necessary, click <code>⌨</code>, and type the string in the editor.</p>
Working directory	<p>Specify a directory to be used by the running task.</p> <ul style="list-style-type: none">– When a default run/debug configuration is created by the keyboard shortcut <code>Ctrl+Shift+F10</code>, or by choosing Run on the context menu of a script, the working directory is the one that contains the executable script. This directory may differ from the project directory.– When this field is left blank, the <code>bin</code> directory of the PyCharm installation will be used.
Path mappings	<p>This field appears, if a remote interpreter has been selected in the field Python interpreter.</p> <p>Click the browse button <code>⌨</code> to define the required mappings between the local and remote paths. In the Edit Path Mappings dialog box, use <code>+</code>/<code>-</code> buttons to create new mappings, or delete the selected ones.</p>
Add content roots to PYTHONPATH	<p>Select this check box to add all content roots of your project to the environment variable PYTHONPATH;</p>
Add source roots to PYTHONPATH	<p>Select this check box to add all source roots of your project to the environment variable PYTHONPATH;</p>
Docker container settings	<p>Warning! This field only appears when Docker-based remote interpreter has been selected for a project.</p> <p>Note Speaking about the correspondence of settings with some options (<code>--net</code>, <code>--link</code>, etc.), note that these options come from Docker command line arguments.</p>

Click `⌨` to open the dialog and specify the following settings:

- **Disable networking:** select this check box to have the networking disabled. This corresponds to `--net="none"`, which means that inside a container the external network resources are not available.
- **Network mode:** corresponds to the other values of the option `--net`.
 - `bridge` is the default value. An IP address will be allocated for container on the bridge's network and traffic will be routed through this bridge to the container.

Containers can communicate via their IP addresses by default. To communicate by name, they must be linked.

- `host`: use the host's network stack inside the container.
- `container:<name|id>`: use the network stack of another container, specified via its `name` or `id`.

Refer to the [Network settings](#) documentation for details.

- **Links:** Use this section to link the container to be created with the other containers. This is applicable to `Network mode = bridge` and corresponds to the `--link` option.
- **Publish all ports:** If the check box is selected, all the container's exposed ports are bound to a random host port.

See e.g. [EXPOSE \(incoming ports\)](#) in [Docker run reference](#). This corresponds to the option `--publish-all`.

- Port bindings: Use this field to specify the Container port/protocol - Host IP address/port mappings.
E.g. `8080 tcp 127.0.0.1 18080` would bind the TCP port `8080` inside the container to port `18080` on the `localhost` or `127.0.0.1` interface on the host machine.

For more info, see e.g. [Connect using network port mapping](#) in [Legacy container links](#).

- Extra hosts: This corresponds to the `--add-host` option. Refer to the page [Managing /etc/hosts](#) for details.
- Volume bindings: Use this field to specify the bindings between the special folders-volumes and the folders of the computer, where the Docker daemon runs. This corresponds to the `-v` option.

See [Managing data in containers](#) for details.

- Environment variables: Use this field to specify the list of environment variables and their values. This corresponds to the `-e` option. Refer to the page [ENV \(environment variables\)](#) for details.

Click  to expand the tables. Click ,  or  to make up the lists.

Logs tab

Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the [Run](#) or [Debug tool window](#).

ItemDescription

Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window .
Log File Entry	The read-only fields in this column list the log files to show. The list can contain: <ul style="list-style-type: none">– Full paths to specific files.– Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.
Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point to the desired location in the dialog that opens .
Show console when a message is printed to standard output stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.
Show console when a message is printed to standard error stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.
	Click this button to open the Edit Log Files Aliases dialog where you can select a new log entry and specify an alias for it.
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog .
	Click this button to remove the selected log entry from the list.
	Click this button to edit the select log file entry. The button is available only when an entry is selected.

Toolbar

ItemShortcutDescription

		Click this button to add a new configuration to the list.
		Click this button to remove the selected configuration from the list.
		Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
	 or 	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the  buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

ItemDescription

Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members.

The shared run/debug configurations are kept in separate xml files under `.idea\runConfigurations` folder, while the local run/debug configurations are kept in the `.idea\workspace.xml`.

This check box is not available when editing the run/debug configuration defaults.

Single instance only If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, PyCharm checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place.

This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources.

If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.

Before launch Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

ItemKeyboardDescription shortcut

-   Click this icon to add a task to the list. Select the task to be added:
 - Run External tool. Select this option to run an application which is external to PyCharm. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in PyCharm yet, add its definition. For more information, see [Configuring Third-Party Tools](#) and [External Tools](#).
 - Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project.
 - Run File Watchers. Select this option to have PyCharm apply all the currently active file watchers, see [Using File Watchers](#) for details.
 - Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package.
 - Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package.
 - Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it.
 - Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:
 - If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.
 - If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.
 - Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see [CoffeeScript Support](#).
 - Run Remote External tool: Add a remote SSH external tool. Refer to the section [Remote SSH External Tools](#) for details.
 - Upload files to Remote Host. Select this option to have the application files automatically uploaded to the server according to the default server access configuration. For more information, see [Configuring Synchronization with a Web Server](#) and [Uploading and Downloading Files](#).

-   Click this icon to remove the selected task from the list.
-   Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
-   Click this icon to move the selected task one line up in the list.
-   Click this icon to move the selected task one line down in the list.

Show this page Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.

Active tool window Select this option if you want the [Run/Debug](#) tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Chromium Remote

This feature is supported in the Professional edition only.

Use this dialog box to create configurations for [debugging applications remotely](#) in [ChromiumOS](#).

In this section:

- [Chromium Remote-specific configuration settings](#)
- [Toolbar](#)
- [Common options](#)

Chromium Remote-specific configuration settings

ItemDescription

Host	In this text box, specify the host where the application is running.
Port	In this spin box, specify the port to connect to.

Toolbar

ItemShortcutDescription

		Click this button to add a new configuration to the list.
		Click this button to remove the selected configuration from the list.
		Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
		Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the  buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

ItemDescription

Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. The shared run/debug configurations are kept in separate xml files under <code>.idea\runConfigurations</code> folder, while the local run/debug configurations are kept in the <code>.idea\workspace.xml</code> . This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, PyCharm checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

ItemKeyboardDescription shortcut

		Click this icon to add a task to the list. Select the task to be added: <ul style="list-style-type: none">– Run External tool. Select this option to run an application which is external to PyCharm. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in PyCharm yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools.– Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project.– Run File Watchers. Select this option to have PyCharm apply all the currently active file watchers, see Using File Watchers for details.– Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the <code>gruntfile.js</code> where the required task is defined, select the task to execute, and specify the arguments to
---	---	---

pass to the Grunt tool.

Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package.

- Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool.
Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package.
- Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with.
Specify the location of the Node.js interpreter and the parameters to pass to it.
- Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:
 - If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.
 - If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.
- Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see [CoffeeScript Support](#).
- Run Remote External tool: Add a remote SSH external tool. Refer to the section [Remote SSH External Tools](#) for details.

		Click this icon to remove the selected task from the list.
		Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
		Click this icon to move the selected task one line up in the list.
		Click this icon to move the selected task one line down in the list.
Show this page	<input type="checkbox"/>	Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window	<input type="checkbox"/>	Select this option if you want the Run/Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Compound

Use this dialog box to create configurations containing multiple run/debug configurations that you can launch at once. This is useful, for example, if you want to launch various automated tests and get test results in one window.

Press **+** to select which of the existing configurations you want to include into the Compound configuration, and fill in the following fields:

ItemDescription

Name	Specify the name of the new Run/Debug Configuration.
Share	Select this check box to make the run/debug configuration available to other team members. The shared run/debug configurations are kept in separate xml files under <code>.idea\runConfigurations</code> folder, while the local run/debug configurations are kept in the <code>.idea\workspace.xml</code> . This check box is not available when editing the run/debug configuration defaults.

Run/Debug Configuration: Cucumber.js

In this dialog box, create configurations for running and debugging **JavaScript unit tests** using the **Cucumber.js test runner**.

On this page:

- [Getting access to the Run/Debug Configuration: Cucumber.js dialog](#)
- [Cucumber.js-specific configuration settings](#)
- [Toolbar](#)
- [Common options](#)

Getting access to the Run/Debug Configuration: Cucumber.js dialog

1. **Install** and **enable** the Node.js plugin. The plugin is not bundled with PyCharm, but it can be installed from the **JetBrains plugin repository** as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).
2. Download and install the **NodeJS** runtime environment that contains the **Node Package Manager(npm)**.
3. Using the **Node Package Manager**, install the **Cucumber.js** as described in [Testing JavaScript with Cucumber.js](#).
4. Install and enable the **Cucumber.js** and **Gherkin** plugins. The plugins are not bundled with PyCharm, but they can be installed from the **JetBrains plugin repository** as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#). Once enabled, the plugins are available at the IDE level, that is, you can use them in all your PyCharm projects.

Cucumber.js-specific configuration settings

ItemDescription

Feature file or directory In this text box, specify the tests to run. **Cucumber.js** runs tests that are called **features** and are written in the **Gherkin** language. Each feature is described in a separate file with the extension `feature`.

- To have one **feature** executed, specify the path to the corresponding `.feature` file.
- To have a bunch of **features** executed, specify the folder where the `.feature` files to run are stored.

Type the path manually or click the Browse button  and choose the file or folder in the dialog box that opens.

Cucumber.js arguments In this text box, specify the command line arguments to be passed to the **Cucumber.js** executable file, such as `-r (--require LIBRARY|DIR), -t (--tags TAG_EXPRESSION), or --coffee`. For details, see **Cucumber's** native built-in help available through the `cucumber-js --help` command.

Executable path In this text box, specify the location of the **cucumber** executable file, `.cmd`, `.bat`, or other depending on the operating system used. The location depends on the installation mode, see [Installing the Cucumber.js test runner](#).

Type the path manually or click the Browse button  and choose the file in the dialog box that opens.

Toolbar

ItemShortcutDescription

		Click this button to add a new configuration to the list.
		Click this button to remove the selected configuration from the list.
		Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
	 or 	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the  buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

ItemDescription

Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. The shared run/debug configurations are kept in separate <code>xml</code> files under <code>.idea\runConfigurations</code> folder, while the local run/debug configurations are kept in the <code>.idea\workspace.xml</code> . This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, PyCharm checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place.

This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources.

If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.

Before launch Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
------	-------------------	-------------



Alt+Insert

Click this icon to add a task to the list. Select the task to be added:

- Run External tool. Select this option to run an application which is external to PyCharm. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in PyCharm yet, add its definition. For more information, see [Configuring Third-Party Tools](#) and [External Tools](#).
- Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project.
- Run File Watchers. Select this option to have PyCharm apply all the currently active [file watchers](#), see [Using File Watchers](#) for details.
- Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package.
- Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package.
- Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it.
- Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:
 - If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.
 - If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.
- Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see [CoffeeScript Support](#).
- Run Remote External tool: Add a remote SSH external tool. Refer to the section [Remote SSH External Tools](#) for details.



Alt+Delete

Click this icon to remove the selected task from the list.



Enter

Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.



Alt+Up

Click this icon to move the selected task one line up in the list.



Alt+Down

Click this icon to move the selected task one line down in the list.

Show this page

Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.

Active tool window

Select this option if you want the [Run/Debug](#) tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Django Server

This feature is supported in the Professional edition only.

Use this dialog box to create run/debug configuration for Django server.

In this section:

- [Configuration tab](#)
- [Logs tab](#)
- [Toolbar](#)
- [Common options](#)

Configuration tab

ItemDescription

Host	In this text box, specify the host name to be used.
Port	In this text box, specify the port number where the server will start.
Additional options	In this text box, specify the options of the <code>django-admin.py</code> utility. Refer to the <code>django-admin.py</code> and <code>manage.py</code> documentation for details.
Run browser	Select this check box, if you want your Django application to open in the default browser. In the text field below, enter the IP address where your application will be opened.
Test server	If this check box is selected, a Django development server is launched with the test database. If this check box is not selected, the development server will be used.
No reload	If this check box is selected, the <code>--noreload</code> option of the <code>runserver</code> command becomes enabled. If this check box is not selected, PyCharm will not select it automatically, which means that debugging in autoreload mode is possible. Refer to the option description for details. This field is only available when the Test server check box is cleared.
Custom run command	Specify here the custom command you want to register with <code>manage.py</code> utility. Such command, being properly added to your project, becomes available via the Run manage.py task command on the Tools menu. Refer to the section Writing custom django-admin commands for details.
Project	Click this drop-down list to select one of the projects, opened in the same PyCharm window , where this run/debug configuration should be used. If there is only one open project, this field is not displayed.
Environment variable	This field shows the list of environment variables. If the list contains several variables, they are delimited with semicolons. To fill in the list, click the browse button, or press <code>Shift+Enter</code> and specify the desired set of environment variables in the Environment Variables dialog box. To create a new variable, click <code>+</code> , and type the desired name and value. By default, the variable <code>PYTHONUNBUFFERED</code> is set to 1.
Python Interpreter	Select one of the pre-configured Python interpreters from the drop-down list. Note that you can select a remote interpreter , as well as the local one. If a remote interpreter is selected, you have to specify path mappings in the corresponding field that appears .
Interpreter options	In this field, specify the string to be passed to the interpreter. If necessary, click <code>ⓘ</code> , and type the string in the editor.
Working directory	Specify a directory to be used by the running task. <ul style="list-style-type: none">– When a default run/debug configuration is created by the keyboard shortcut <code>Ctrl+Shift+F10</code>, or by choosing Run on the context menu of a script, the working directory is the one that contains the executable script. This directory may differ from the project directory.– When this field is left blank, the <code>bin</code> directory of the PyCharm installation will be used.
Path mappings	This field appears, if a remote interpreter has been selected in the field Python interpreter. Click the browse button <code>⋮</code> to define the required mappings between the local and remote paths. In the Edit Path Mappings dialog box, use <code>+</code> / <code>-</code> buttons to create new mappings, or delete the selected ones.
Add content roots to PYTHONPATH	Select this check box to add all content roots of your project to the environment variable PYTHONPATH;
Add source roots to PYTHONPATH	Select this check box to add all source roots of your project to the environment variable PYTHONPATH;
Docker container settings	<div style="background-color: #ffff00; padding: 5px;">Warning! This field only appears when Docker-based remote interpreter has been selected for a project.</div> <div style="background-color: #ffff00; padding: 5px; margin-top: 10px;">Note Speaking about the correspondence of settings with some options (<code>--net</code>, <code>--link</code>, etc.), note that these options come from Docker command line arguments.</div>

Click `⋮` to open the dialog and specify the following settings:

- Disable networking: select this check box to have the networking disabled. This corresponds to `--net="none"`, which means that inside a container the external network resources are not available.

- Network mode: corresponds to the other values of the option `--net`.
- `bridge` is the default value. An IP address will be allocated for container on the bridge's network and traffic will be routed through this bridge to the container.

Containers can communicate via their IP addresses by default. To communicate by name, they must be linked.

- `host`: use the host's network stack inside the container.
- `container:<name|id>`: use the network stack of another container, specified via its name or id.

Refer to the [Network settings](#) documentation for details.

- Links: Use this section to link the container to be created with the other containers. This is applicable to `Network mode = bridge` and corresponds to the `--link` option.
- Publish all ports: If the check box is selected, all the container's exposed ports are bound to a random host port.

See e.g. [EXPOSE \(incoming ports\)](#) in [Docker run reference](#). This corresponds to the option `--publish-all`.

- Port bindings: Use this field to specify the Container port/protocol - Host IP address/port mappings.
E.g. `8080 tcp 127.0.0.1 18080` would bind the TCP port `8080` inside the container to port `18080` on the `localhost` or `127.0.0.1` interface on the host machine.

For more info, see e.g. [Connect using network port mapping](#) in [Legacy container links](#).

- Extra hosts: This corresponds to the `--add-host` option. Refer to the page [Managing /etc/hosts](#) for details.
- Volume bindings: Use this field to specify the bindings between the special folders-volumes and the folders of the computer, where the Docker daemon runs. This corresponds to the `-v` option.

See [Managing data in containers](#) for details.

- Environment variables: Use this field to specify the list of environment variables and their values. This corresponds to the `-e` option. Refer to the page [ENV \(environment variables\)](#) for details.

Click  to expand the tables. Click ,  or  to make up the lists.

Logs tab

Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the [Run](#) or [Debug tool window](#).

ItemDescription

Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window .
Log File Entry	The read-only fields in this column list the log files to show. The list can contain: <ul style="list-style-type: none"> - Full paths to specific files. - Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.
Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point to the desired location in the dialog that opens .
Show console when a message is printed to standard output stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.
Show console when a message is printed to standard error stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.
	Click this button to open the Edit Log Files Aliases dialog where you can select a new log entry and specify an alias for it.
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog .
	Click this button to remove the selected log entry from the list.
	Click this button to edit the selected log file entry. The button is available only when an entry is selected.

Toolbar

ItemShortcutDescription

		Click this button to add a new configuration to the list.
		Click this button to remove the selected configuration from the list.
		Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
	 or 	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the  buttons.



Sort configurations

Click this button to sort configurations in alphabetical order.

Common options

ItemDescription

Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	<p>Select this check box to make the run/debug configuration available to other team members.</p> <p>The shared run/debug configurations are kept in separate xml files under <code>.idea\runConfigurations</code> folder, while the local run/debug configurations are kept in the <code>.idea\workspace.xml</code>.</p> <p>This check box is not available when editing the run/debug configuration defaults.</p>
Single instance only	<p>If this check box is selected, this run/debug configuration cannot be launched more than once.</p> <p>Every time a new run/debug configuration is launched, PyCharm checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place.</p> <p>This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources.</p> <p>If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.</p>
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

ItemKeyboardDescription shortcut

	Alt+Insert Click this icon to add a task to the list. Select the task to be added: <ul style="list-style-type: none"> – Run External tool. Select this option to run an application which is external to PyCharm. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in PyCharm yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools. – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run File Watchers. Select this option to have PyCharm apply all the currently active file watchers, see Using File Watchers for details. – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the <code>gruntfile.js</code> where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the <code>grunt-cli</code> package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the <code>gulpfile.js</code> where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the <code>gulp</code> package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the <code>package.json</code> file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: <ul style="list-style-type: none"> – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support. – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details. – Upload files to Remote Host. Select this option to have the application files automatically uploaded to the server according to the default server access configuration. For more information, see Configuring Synchronization with a Web Server and Uploading and Downloading Files.
	Alt+Delete Click this icon to remove the selected task from the list.
	Enter Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up Click this icon to move the selected task one line up in the list.
	Alt+Down Click this icon to move the selected task one line down in the list.
Show this page	Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window	Select this option if you want the Run/Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Django Test

This feature is supported in the Professional edition only.

Use this dialog box to create a run/debug configuration for Django tests.

In this section:

- [Configuration tab](#)
- [Logs tab](#)
- [Toolbar](#)
- [Common options](#)

Configuration tab

ItemDescription

Target Specify the target to be executed. If the field is left empty, it means that all the tests in all the applications specified in `INSTALLED_APPS` will be executed.

- If you want to run tests in a certain application, specify the application name.
- To run a specific test case, specify its name after the application name, delimited with a dot.
- To run a single test method within a test case, add the test method name after dot.

Same rules apply to the doctests contained in the test targets. The test label is used as the path to the test method or class to be executed. If there is function with a doctest, or a class with a class-level doctest, you can invoke that test by appending the name of the test method or class to the label.

Custom settings If this check box is selected, Django test will run with the specified custom settings, rather than with the default ones. Specify the fully qualified name of the file that contains Django settings. You can either type it manually, in the text field to the right, or click the browse button, and select one in the [dialog box that opens](#).

If this check box is not selected, Django test will run with the default settings, defined in the Settings field of the [Django](#) page. The text field is disabled.

Options If this check box is selected, it is possible to specify parameters to be passed to the Django tests. Type the list of parameters in the text field to the right, prepending parameters with '-' and using spaces as delimiters. For example:

```
--noinput --failfast
```

If this check box is not selected, the text field is disabled.

Environment

Project Click this drop-down list to select one of the projects, [opened in the same PyCharm window](#), where this run/debug configuration should be used. If there is only one open project, this field is not displayed.

Environment variable This field shows the list of environment variables. If the list contains several variables, they are delimited with semicolons.

To fill in the list, click the browse button, or press `Shift+Enter` and specify the desired set of environment variables in the Environment Variables dialog box.

To create a new variable, click `+`, and type the desired name and value.

Python interpreter Select one of the pre-configured [Python interpreters](#) from the drop-down list.

Note that you can select a [remote interpreter](#), as well as the `local` one. If a remote interpreter is selected, you have to specify path mappings in the [corresponding field that appears](#).

Interpreter options In this field, specify the string to be passed to the interpreter. If necessary, click `⌨` and type the string in the editor.

Working directory Specify a directory to be used by the running task.

- When a default run/debug configuration is created by the keyboard shortcut `Ctrl+Shift+F10`, or by choosing Run on the context menu of a script, the working directory is the one that contains the executable script. This directory may differ from the project directory.
- When this field is left blank, the `bin` directory of the PyCharm installation will be used.

Path mappings This field appears, if a remote interpreter has been selected in the field Python interpreter.

Click the browse button `⌨` to define the required mappings between the local and remote paths. In the Edit Path Mappings dialog box, use `+`/`-` buttons to create new mappings, or delete the selected ones.

Add content roots to PYTHONPATH Select this check box to add all [content roots](#) of your project to the environment variable PYTHONPATH;

Add source roots to PYTHONPATH Select this check box to add all [source roots](#) of your project to the environment variable PYTHONPATH;

Docker container settings **Warning!** This field only appears when [Docker-based remote interpreter](#) has been selected for a project.

Note Speaking about the correspondence of settings with some options (`--net`, `--link`, etc.), note that these options come from [Docker command line arguments](#).

Click `⌨` to open the dialog and specify the following settings:

- Disable networking: select this check box to have the networking disabled. This corresponds to `--net="none"`, which means that inside a container the

external network resources are not available.

- Network mode: corresponds to the other values of the option `--net`.
 - `bridge` is the default value. An IP address will be allocated for container on the bridge's network and traffic will be routed through this bridge to the container.

Containers can communicate via their IP addresses by default. To communicate by name, they must be linked.

- `host`: use the host's network stack inside the container.
- `container:<name|id>`: use the network stack of another container, specified via its `name` or `id`.

Refer to the [Network settings](#) documentation for details.

- Links: Use this section to link the container to be created with the other containers. This is applicable to `Network mode = bridge` and corresponds to the `--link` option.
- Publish all ports: If the check box is selected, all the container's exposed ports are bound to a random host port.

See e.g. [EXPOSE \(incoming ports\)](#) in [Docker run reference](#). This corresponds to the option `--publish-all`.

- Port bindings: Use this field to specify the Container port/protocol - Host IP address/port mappings.
E.g. `8080 tcp 127.0.0.1 18080` would bind the TCP port `8080` inside the container to port `18080` on the `localhost` or `127.0.0.1` interface on the host machine.

For more info, see e.g. [Connect using network port mapping](#) in [Legacy container links](#).

- Extra hosts: This corresponds to the `--add-host` option. Refer to the page [Managing /etc/hosts](#) for details.
- Volume bindings: Use this field to specify the bindings between the special folders-volumes and the folders of the computer, where the Docker daemon runs. This corresponds to the `-v` option.

See [Managing data in containers](#) for details.

- Environment variables: Use this field to specify the list of environment variables and their values. This corresponds to the `-e` option. Refer to the page [ENV \(environment variables\)](#) for details.

Click  to expand the tables. Click ,  or  to make up the lists.

Logs tab

Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the [Run](#) or [Debug tool window](#).

ItemDescription

Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window .
Log File Entry	The read-only fields in this column list the log files to show. The list can contain: <ul style="list-style-type: none">- Full paths to specific files.- Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.
Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point to the desired location in the dialog that opens .
Show console when a message is printed to standard output stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.
Show console when a message is printed to standard error stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.
	Click this button to open the Edit Log Files Aliases dialog where you can select a new log entry and specify an alias for it.
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog .
	Click this button to remove the selected log entry from the list.
	Click this button to edit the selected log file entry. The button is available only when an entry is selected.

Toolbar

ItemShortcutDescription

	 <code>Alt+Insert</code>	Click this button to add a new configuration to the list.
	 <code>Alt+Delete</code>	Click this button to remove the selected configuration from the list.
	 <code>Ctrl+D</code>	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
	 <code>Alt+Up</code> or <code>Alt+Down</code>	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the  buttons.

Common options

ItemDescription

Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	<p>Select this check box to make the run/debug configuration available to other team members.</p> <p>The shared run/debug configurations are kept in separate xml files under <code>.idea\runConfigurations</code> folder, while the local run/debug configurations are kept in the <code>.idea\workspace.xml</code>.</p> <p>This check box is not available when editing the run/debug configuration defaults.</p>
Single instance only	<p>If this check box is selected, this run/debug configuration cannot be launched more than once.</p> <p>Every time a new run/debug configuration is launched, PyCharm checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place.</p> <p>This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources.</p> <p>If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.</p>
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

ItemKeyboardDescription shortcut

	Alt+Insert	<p>Click this icon to add a task to the list. Select the task to be added:</p> <ul style="list-style-type: none"> – Run External tool. Select this option to run an application which is external to PyCharm. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in PyCharm yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools. – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run File Watchers. Select this option to have PyCharm apply all the currently active file watchers, see Using File Watchers for details. – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the <code>gruntfile.js</code> where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the <code>grunt-cli</code> package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the <code>gulpfile.js</code> where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the <code>gulp</code> package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the <code>package.json</code> file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: <ul style="list-style-type: none"> – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support. – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details. – Upload files to Remote Host. Select this option to have the application files automatically uploaded to the server according to the default server access configuration. For more information, see Configuring Synchronization with a Web Server and Uploading and Downloading Files.
	Alt+Delete	Click this icon to remove the selected task from the list.
	Enter	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up	Click this icon to move the selected task one line up in the list.
	Alt+Down	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run/Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Docker Deployment

Run | Edit Configurations | + | Docker Deployment

Docker Deployment [run/debug configurations](#) let you download and build [Docker](#) images, and create and start Docker containers. (The Docker integration plugin must be enabled.)

- [Name, Share, and Single instance only](#)
- [Deployment tab](#)
- [Container tab](#)
- [Before Launch options](#)
- [Toolbar](#)

See also, [Docker](#).

Name, Share, and Single instance only

ItemDescription

Name	The name of the run configuration.
Share	Select this check box to share the run configuration through version control. If the check box is not selected, the run configuration settings are stored in <code>.idea/workspace.xml</code> . If the check box is selected, the settings are stored in a separate <code>.xml</code> file in <code>.idea/runConfigurations</code> or in the <code>.ipr</code> file.
Single instance only	If you select this check box, only one instance of the run configuration will run at a time.

Deployment tab

ItemDescription

Server	Select the Docker configuration to be used. To create a new configuration, or to edit an existing one, click  (<code>Shift+Enter</code>). For more information, see Docker .
Deployment	One of the following: <ul style="list-style-type: none">- Docker Image. Select this option if you want to deploy a Docker image available locally.- <code><docker-dir>/Dockerfile</code>. Select the Dockerfile to be used. (The Dockerfiles available in your project are suggested.)- <code><docker-dir>/docker-compose.yml</code>. Select the Docker Compose YAML file to be used. (The <code>docker-compose.yml</code> files available in your project are suggested.)
The following settings are not available if a <code>docker-compose.yml</code> file is selected in the Deployment field.	
Image ID	If Docker Image is selected in the Deployment field: the ID of the image to be deployed, a sequence of hexadecimal symbols. For the image of interest, you can, for example, use the Copy image ID command in the Docker tool window and then paste the ID into this field.
Image tag	If <code><docker-dir>/Dockerfile</code> is selected in the Deployment field: if you want to use an image with a particular tag, specify that tag. (The image in this case is specified in the Dockerfile.) If nothing is specified, Docker will look for the image with the tag <code>latest</code> .
Container name	If you want to give the container that will be created a particular name, specify that name. Otherwise, Docker will itself decide what the name should be.
After launch	Select this check box to start a web browser after starting the run/debug configuration. Select the browser from the list. Click  (<code>Shift+Enter</code>) to configure your web browsers.
With JavaScript debugger	If this check box is selected, the web browser is started with the JavaScript debugger enabled. Note that JavaScript debugging is available only for Firefox and Google Chrome. When you debug your JavaScript in Firefox for the first time, the JetBrains Firefox extension is installed.
The field underneath After launch	Specify the URL the browser should go to when started. In most typical cases, this URL corresponds to the root of your Web application or its starting page.
Debug Port	The port to be used for debugging. (Debugging is supported only for Java.) This may be any unused port on your computer. If there is a warning <i>Debug port forwarding not found</i> , click Fix. As a result, the necessary port mapping info is added to the JSON container settings file . Note the command line arguments suggested by PyCharm. To make debugging possible, these should be used to start the JVM in the container.

Container tab

This tab contains the settings for your container.

NOTE: The settings on this tab are ignored if a `docker-compose.yml` file is selected in the [Deployment field](#).

ItemDescription

JSON file	Path to a JSON file with container settings in the Docker Engine API format. <ul style="list-style-type: none">-  lets you select an existing file.-  lets you generate and save a sample container settings <code>.json</code> file. <p>CLI lets you translate the <code>docker run</code> options into JSON container settings Docker Engine API format. The result is saved in a JSON file which is associated with your container.</p> <p>(For <code>docker run [OPTIONS] IMAGE [COMMAND] [ARG...]</code>)</p>
-----------	--

OPTIONS are specified in the CLI options field of the dialog that opens when you click CLI,

COMMAND is specified in the Command field,

ARG. . . part is specified in the Entrypoint or Command filed.)

The settings below the JSON file field take precedence when overlapping ones set in the JSON file.

Entrypoint	The entry point for your container, see e.g. <code>ENTRYPOINT</code> (default command to execute at runtime) in Docker run reference .
Command	See <code>CMD</code> (default command or options) in the Docker run reference .
Publish all ports	If the check box is selected, all the container's exposed ports are bound to a random host port. See e.g. <code>EXPOSE (incoming ports)</code> in Docker run reference .
Port bindings	Container port/protocol - Host IP address/port mappings. E.g. <code>8080 tcp 127.0.0.1 18080</code> would bind the TCP port <code>8080</code> inside the container to port <code>18080</code> on the <code>localhost</code> or <code>127.0.0.1</code> interface on the host machine. For more info, see e.g. Connect using network port mapping in Legacy container links .
Links	Container name - alias mappings. See e.g. Communication across links in Legacy container links .
Volume bindings	Volume bindings such as container path - host path. See e.g. Manage data in containers .
Environment variables	Environment variables as variable - value pairs. See e.g. ENV (environment variables) in Docker run reference .

Before Launch options

Specify which tasks should be carried out before starting the run/debug configuration.

ItemShortcutDescription

		Click this icon to add a task to the list. Select the task to be added, for example: <ul style="list-style-type: none">– Run External tool. Select this option to run an application which is external to PyCharm. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in PyCharm yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools.– Run Another Configuration. Select this option to execute another run/debug configuration. In the dialog that opens, select the configuration to be run.– Generate CoffeeScript Source Maps. Select this option to generate the source maps for your CoffeeScript sources. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support.– Run Remote External Tool. Select this option to run a remote application which is external to PyCharm. In the dialog that opens, select one or more remote applications to be run. If the necessary applications are not defined in PyCharm yet, add their definitions. For more information, see Configuring Third-Party Tools and External Tools.
		Click this icon to remove the selected task from the list.
		Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
		Click these icons to move the selected task one line up or down in the list. (The tasks are performed in the order that they appear in the list.)
Show this page	<input type="checkbox"/>	Select this check box to show the run/debug configuration settings prior to actually starting the run/debug configuration.
Activate tool window	<input type="checkbox"/>	If this check box is selected, the Debug tool window opens when you start the run/debug configuration in the debug mode. Otherwise, the tool window isn't shown. However, when the configuration is running in the debug mode, you can open the Debug tool window for it yourself if necessary.

Toolbar

ItemShortcutDescription

		Create a run/debug configuration.
		Delete the selected run/debug configuration.
		Create a copy of the selected run/debug configuration.
		View and edit the default settings for the selected run/debug configuration.
		Move the selected run/debug configuration up and down in the list. The order of configurations in the list defines the order in which the configurations appear in the corresponding list on the main toolbar.
		You can group run/debug configurations by placing them into folders. To create a folder, select the configurations to be grouped and click  . Specify the name of the folder. Then, to move a configuration into a folder, between the folders or out of a folder, use  and  . You can also drag a configuration into a folder. To remove grouping, select a folder and click  .
		See also, Creating Folders and Grouping Run/Debug Configurations .

Run/Debug Configuration: Firefox Remote

This feature is supported in the Professional edition only.

In this dialog box, create configurations to use for debugging applications locally in Firefox, versions 36 and higher. Note that remote debugging in Firefox is currently not supported at all.

On this page:

- [Firefox Remote-specific configuration settings](#)
- [Toolbar](#)
- [Common options](#)

Firefox Remote-specific configuration settings

ItemDescription

Host	In this text box, specify the host where the application is running. Currently it is just <code>localhost</code> .
Port	In this spin box, specify the port the debugger will listen to. It must be the port that you specified when enabling debugging in Firefox, see Debugging JavaScript in Chrome . The default value is 6000.

Toolbar

ItemShortcutDescription

		Click this button to add a new configuration to the list.
		Click this button to remove the selected configuration from the list.
		Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
		Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the  buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

ItemDescription

Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. The shared run/debug configurations are kept in separate xml files under <code>.idea\runConfigurations</code> folder, while the local run/debug configurations are kept in the <code>.idea\workspace.xml</code> . This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, PyCharm checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

ItemKeyboardDescription shortcut

		Click this icon to add a task to the list. Select the task to be added: <ul style="list-style-type: none">– Run External tool. Select this option to run an application which is external to PyCharm. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in PyCharm yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools.– Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project.– Run File Watchers. Select this option to have PyCharm apply all the currently active file watchers, see Using File Watchers for details.
---	---	---

- Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool.
Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package.
- Run Gulp task. Select this option to run a Gulp task. In the Gulp task dialog box that opens, specify the `gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool.
Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package.
- Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with.
Specify the location of the Node.js interpreter and the parameters to pass to it.
- Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:
 - If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.
 - If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.
- Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see [CoffeeScript Support](#).
- Run Remote External tool: Add a remote SSH external tool. Refer to the section [Remote SSH External Tools](#) for details.

	<code>Alt+Delete</code>	Click this icon to remove the selected task from the list.
	<code>Enter</code>	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	<code>Alt+Up</code>	Click this icon to move the selected task one line up in the list.
	<code>Alt+Down</code>	Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run/Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Grunt.js

This feature is supported in the Professional edition only.

In this dialog box, create configurations for running **Grunt.js** tasks.

On this page:

- [Getting access to the Run/Debug Configuration: Grunt dialog](#)
- [Grunt-specific configuration settings](#)
- [Toolbar](#)
- [Common options](#)

Getting access to the Run/Debug Configuration: Grunt dialog

1. **Install** and **enable** the Node.js plugin. The plugin is not bundled with PyCharm, but it can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).
2. Download and install the [NodeJS](#) runtime environment that contains the [Node Package Manager\(npm\)](#).
3. Install **Grunt** as described in [Installing Grunt](#).

Grunt-specific configuration settings

ItemDescription

Gruntfile	In this field, specify the location of the <code>Gruntfile.js</code> file to retrieve the definitions of the tasks from. Select the path from the drop-down list or click the  button and choose the file from the dialog box that opens.
Tasks	In this field, specify the tasks to run. Do one of the following: <ul style="list-style-type: none">– To run one task, select it from the drop-down list.– To run several tasks, type their names in the text box using blank spaces as separators.
Force execution	Select this check box to have Grunt ignore warnings and continue executing the launched task until the task is completed successfully or an error occurs, if the check box is cleared, the task execution is stopped by the first reported warning.
Verbose mode	Select this check box to have the <code>verbose</code> mode applied and thus have a full detailed log of a task execution displayed.
Node Interpreter	In this field, specify the NodeJS installation home. Type the path to the NodeJS executable file manually, or click the  button and select the location in the dialog box, that opens. If you have appointed one of the installations as default , the field displays the path to its executable file.
Node Options	In this text box, type the NodeJS-specific command line options to be passed to the NodeJS executable file. See Node Parameters for details.
Grunt-cli Package	In this field, specify the path to the globally installed <code>Grunt-cli</code> package installed globally See Installing Grunt.js for details.
Environment Variables	In this field, specify the environment variables for the NodeJS executable file, if applicable. Click the Browse button  to the right of the field and configure a list of variables in the Environment Variables dialog box, that opens: <ul style="list-style-type: none">– To define a new variable, click the Add toolbar button  and specify the variable name and value.– To discard a variable definition, select it in the list and click the Delete toolbar button .– Click OK, when ready <p>The definitions of variables are displayed in the Environment variables read-only field with semicolons as separators. The acceptable variables are:</p> <ul style="list-style-type: none">– <code>NODE_PATH</code> : A <code>:</code>-separated list of directories prefixed to the module search path.– <code>NODE_MODULE_CONTEXTS</code> : Set to 1 to load modules in their own global contexts.– <code>NODE_DISABLE_COLORS</code> : Set to 1 to disable colors in the REPL.

Toolbar

ItemShortcutDescription

		Click this button to add a new configuration to the list.
		Click this button to remove the selected configuration from the list.
		Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
	 or 	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the  buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

ItemDescription

Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. The shared run/debug configurations are kept in separate xml files under <code>.idea\runConfigurations</code> folder, while the local run/debug configurations are kept in the <code>.idea\workspace.xml</code> . This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, PyCharm checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

ItemKeyboardDescription shortcut

	 Click this icon to add a task to the list. Select the task to be added: <ul style="list-style-type: none">– Run External tool. Select this option to run an application which is external to PyCharm. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in PyCharm yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools.– Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project.– Run File Watchers. Select this option to have PyCharm apply all the currently active file watchers, see Using File Watchers for details.– Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the <code>Gruntfile.js</code> where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the <code>grunt-cli</code> package.– Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the <code>Gulpfile.js</code> where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the <code>gulp</code> package.– Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the <code>package.json</code> file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it.– Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:<ul style="list-style-type: none">– If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.– If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.– Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support.– Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.– Upload files to Remote Host. Select this option to have the application files automatically uploaded to the server according to the default server access configuration. For more information, see Configuring Synchronization with a Web Server and Uploading and Downloading Files.
	 Click this icon to remove the selected task from the list.
	 Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	 Click this icon to move the selected task one line up in the list.
	 Click this icon to move the selected task one line down in the list.
Show this page	Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window	Select this option if you want the Run/Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Gulp.js

This feature is supported in the Professional edition only.

In this dialog box, create configurations for running **Gulp.js** tasks.

On this page:

- [Getting access to the Run/Debug Configuration: Gulp.js dialog](#)
- [Gulp.js-specific configuration settings](#)
- [Toolbar](#)
- [Common options](#)

Getting access to the Run/Debug Configuration: Gulp.js dialog

1. **Install** and **enable** the Node.js plugin. The plugin is not bundled with PyCharm, but it can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).
2. Download and install the [NodeJS](#) runtime environment that contains the [Node Package Manager\(npm\)](#).
3. Install the `gulp` package as described in [Installing Gulp.js](#).

Gulp.js-specific configuration settings

ItemDescription

Gulpfile	In this field, specify the location of the <code>gulpfile.js</code> file to retrieve the definitions of the tasks from. Select the path from the drop-down list or click the  button and choose the file from the dialog box that opens.
Tasks	In this field, specify the tasks to run. Do one of the following: <ul style="list-style-type: none">– To run one task, select it from the drop-down list.– To run several tasks, type their names in the text box using blank spaces as separators.
Arguments	In this text box, specify the arguments for tasks to be executed with. Use the following format: <pre>--<parameter_name> <parameter_value></pre> <p>For example: <code>--env development</code>.</p> <p>For details about passing task arguments, see https://github.com/gulpjs/gulp/blob/master/docs/recipes/pass-arguments-from-cli.md</p>
Node Interpreter	In this field, specify the NodeJS installation home. Type the path to the NodeJS executable file manually, or click the  button and select the location in the dialog box, that opens. If you have appointed one of the installations as default , the field displays the path to its executable file.
Node Options	In this text box, type the NodeJS-specific command line options to be passed to the NodeJS executable file. See Node Parameters for details. In the default configuration, type <code>--harmony</code> in this text box to have PyCharm build a tasks tree according to a <code>gulpfile.js</code> written in ECMAScript 6. Technically, PyCharm invokes <code>gulp.js</code> and processes <code>gulpfile.js</code> according to the default <code>gulp.js</code> run configuration. However this is done silently and does not require any steps from your side. However, if your <code>gulpfile.js</code> is written in ECMAScript 6 , by default PyCharm does not recognize this format and fails to build a tasks tree. To solve this problem, specify <code>--harmony</code> as a Node parameter of the default <code>gulp.js</code> run configuration.
Gulp Package	In this field, specify the path to the <code>gulp</code> package installed locally , under the project root. See Installing Gulp.js for details.
Environment Variables	In this field, specify the environment variables for the NodeJS executable file, if applicable. Click the Browse button  to the right of the field and configure a list of variables in the Environment Variables dialog box, that opens: <ul style="list-style-type: none">– To define a new variable, click the Add toolbar button  and specify the variable name and value.– To discard a variable definition, select it in the list and click the Delete toolbar button .– Click OK, when ready <p>The definitions of variables are displayed in the Environment variables read-only field with semicolons as separators. The acceptable variables are:</p> <ul style="list-style-type: none">– <code>NODE_PATH</code> : A <code>:</code>-separated list of directories prefixed to the module search path.– <code>NODE_MODULE_CONTEXTS</code> : Set to 1 to load modules in their own global contexts.– <code>NODE_DISABLE_COLORS</code> : Set to 1 to disable colors in the REPL.

Toolbar

ItemShortcutDescription

		Click this button to add a new configuration to the list.
		Click this button to remove the selected configuration from the list.
		Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
	 or 	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to

the newly created folder. If only a category is in focus, an empty folder is created.

Move run/debug configurations to a folder using drag-and-drop, or the   buttons.



Sort configurations

Click this button to sort configurations in alphabetical order.

Common options

ItemDescription

Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. The shared run/debug configurations are kept in separate xml files under <code>.idea\runConfigurations</code> folder, while the local run/debug configurations are kept in the <code>.idea\workspace.xml</code> . This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, PyCharm checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.

Before launch Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

ItemKeyboardDescription shortcut



 Alt+Insert

Click this icon to add a task to the list. Select the task to be added:

- Run External tool. Select this option to run an application which is external to PyCharm. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in PyCharm yet, add its definition. For more information, see [Configuring Third-Party Tools](#) and [External Tools](#).
- Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run.
This option is available only if you have already at least one run/debug configuration in the current project.
- Run File Watchers. Select this option to have PyCharm apply all the currently active [file watchers](#), see [Using File Watchers](#) for details.
- Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `Gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool.
Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package.
- Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `Gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool.
Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package.
- Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with.
Specify the location of the Node.js interpreter and the parameters to pass to it.
- Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:
 - If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.
 - If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.
- Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see [CoffeeScript Support](#).
- Run Remote External tool: Add a remote SSH external tool. Refer to the section [Remote SSH External Tools](#) for details.
- Upload files to Remote Host. Select this option to have the application files automatically uploaded to the server according to the default server access configuration. For more information, see [Configuring Synchronization with a Web Server](#) and [Uploading and Downloading Files](#).



 Alt+Delete

Click this icon to remove the selected task from the list.



 Enter

Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.



 Alt+Up

Click this icon to move the selected task one line up in the list.



 Alt+Down

Click this icon to move the selected task one line down in the list.

Show this page

Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.

Active tool window

Select this option if you want the [Run/Debug](#) tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: JavaScript Debug

This feature is supported in the Professional edition only.

In this dialog box, create a configuration to be used for debugging JavaScript in applications running on the built-in or on an external web server and for debugging Dart Web applications.

PyCharm supports debugging applications running on the built-in or an external web server only in [Google Chrome](#) and other browsers of the **Chrome** family. Debugging applications running on the built-in server is supported for Firefox, version 36 or higher, through the **Firefox Remote** debug configuration. Debugging applications running on external web servers in Firefox is not supported at all.

On this page:

- [JavaScript Debug-specific configuration settings](#)
- [Toolbar](#)
- [Common options](#)

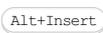
JavaScript Debug-specific configuration settings

ItemDescription

Name	In this text box, specify the name of the current run/debug configuration.
Share	Select this check box to make the run/debug configuration available to other team members. The shared run/debug configurations are kept in separate xml files under <code>.idea\runConfigurations</code> folder, while the local run/debug configurations are kept in the <code>.idea\workspace.xml</code> . This check box is not available when editing the run/debug configuration defaults.
URL	<ul style="list-style-type: none">– Debugging JavaScript: In this text box, specify the URL address of the HTML file that references the JavaScript to debug. For local debugging, type the URL in the format <code>http://localhost:<built-in server port>/<project root></code>. The built-in server port (1024 or higher) is specified on the page of the Settings dialog box.– Debugging a Dart Web application: In this text box, specify the URL address of the HTML file that references the Dart code to debug. Make sure that the port specified in this URL address is the same as the Built-in server port on the Debugger page of the Settings dialog box and the port specified in the settings for the Chrome Extension.
Browser	From this drop down list, select the browser, where your application will be debugged. <ul style="list-style-type: none">– Debugging JavaScript: Chrome– Debugging a Dart Web application: If you choose Dartium which has a built-in Dart virtual machine, simply the Dart code is executed. If you choose Chrome, the Dart code is compiled into JavaScript through the Dart2js tool, which is a part of the Pub tool. Whether such compilation is required or not is decided by the Pub Serve tool. The tool is invoked automatically when you run or debug a Dart web application, either by opening an HTML file or by launching a run/debug configuration. The tool analyzes the <code><script></script></code> element in the HTML file. If the element references a Dart file, Pub Serve invokes the Dart2js compiler whereupon the compiled JavaScript files are passed to the browser where they are processed. Specify the URL address of the HTML file that references the Dart code to debug. Make sure that the port specified in this URL address is the same as the Built-in server port on the Debugger page of the Settings dialog box and the port specified in the settings for the Chrome Extension.
Remote URLs of local files	<ul style="list-style-type: none">– Debugging JavaScript: PyCharm displays this area only when you create a permanent debug configuration manually. For automatically generated temporary configurations the area is not shown. In this area, map the local files to be involved in debugging to the URL addresses of their copies on the server.<ul style="list-style-type: none">– File/Directory - in this read-only field, select the desired local file or directory in the project tree.– Remote URL - in this text box, type the absolute URL address of the corresponding file or folder on the server.– Debugging a Dart Web application: PyCharm displays this area only when the port specified in the URL field is different from the port of the built-in Web server specified on the Debugger page of the Settings dialog box.

Toolbar

ItemShortcutDescription

		Click this button to add a new configuration to the list.
		Click this button to remove the selected configuration from the list.
		Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
	 or 	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created.

Move run/debug configurations to a folder using drag-and-drop, or the   buttons.



Sort configurations Click this button to sort configurations in alphabetical order.

Common options

ItemDescription

Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	<p>Select this check box to make the run/debug configuration available to other team members.</p> <p>The shared run/debug configurations are kept in separate xml files under <code>.idea\runConfigurations</code> folder, while the local run/debug configurations are kept in the <code>.idea\workspace.xml</code>.</p> <p>This check box is not available when editing the run/debug configuration defaults.</p>
Single instance only	<p>If this check box is selected, this run/debug configuration cannot be launched more than once.</p> <p>Every time a new run/debug configuration is launched, PyCharm checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place.</p> <p>This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources.</p> <p>If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.</p>

Before launch Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

ItemKeyboardDescription shortcut

	 Click this icon to add a task to the list. Select the task to be added:
	<ul style="list-style-type: none">– Run External tool. Select this option to run an application which is external to PyCharm. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in PyCharm yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools.– Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project.– Run File Watchers. Select this option to have PyCharm apply all the currently active file watchers, see Using File Watchers for details.– Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the <code>gruntfile.js</code> where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the <code>grunt-cli</code> package.– Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the <code>gulpfile.js</code> where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the <code>gulp</code> package.– Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the <code>package.json</code> file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it.– Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:<ul style="list-style-type: none">– If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.– If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.– Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support.– Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.– Upload files to Remote Host. Select this option to have the application files automatically uploaded to the server according to the default server access configuration. For more information, see Configuring Synchronization with a Web Server and Uploading and Downloading Files.
	 Click this icon to remove the selected task from the list.
	 Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	 Click this icon to move the selected task one line up in the list.
	 Click this icon to move the selected task one line down in the list.
Show this page	Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.

Active tool window

Select this option if you want the [Run/Debug](#) tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Jest

In this dialog box, create configurations for running [Jest](#) tests.

On this page:

- [Getting access to the Run/Debug Configuration: Jest dialog](#)
- [Jest-specific configuration settings](#)
- [Toolbar](#)
- [Common options](#)

Getting access to the Run/Debug Configuration: Jest dialog

1. **Install** and **enable** the Node.js plugin. The plugin is not bundled with PyCharm, but it can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).
2. Download and install the [NodeJS](#) runtime environment that contains the [Node Package Manager\(npm\)](#).

Jest-specific configuration settings

ItemDescription

Configuration file	In this field, specify the <code>jest.config</code> file to use. If the field is empty, PyCharm looks for a <code>package.json</code> file with a <code>jest</code> key. The search is performed in the file system upwards from the <code>working directory</code> . If no appropriate <code>package.json</code> file is found, then the Jest default configuration is used.
Node interpreter	In this field, specify the NodeJS installation home. Type the path to the NodeJS executable file manually, or click the  button and select the location in the dialog box, that opens. If you have appointed one of the installations as default , the field displays the path to its executable file.
Jest package	In this field, specify the location of the <code>jest</code> package.
Working Directory	In this field, specify the working directory of the application. All references in the starting NodeJS application file, for example, imports, will be resolved relative to this folder, unless such references use full paths. By default, the field shows the <code>project root folder</code> . To change this predefined setting, choose the desired folder from the drop-down list, or type the path manually, or click the Browse button  and select the location in the dialog box, that opens.
Jest options	In this text box, type the Jest CLI options to be passed to Jest.
Environment variables	In this field, optionally specify the environment variables for executing commands.

Toolbar

ItemShortcutDescription

		Click this button to add a new configuration to the list.
		Click this button to remove the selected configuration from the list.
		Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
	 or 	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the  buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

ItemDescription

Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. The shared run/debug configurations are kept in separate xml files under <code>.idea\runConfigurations</code> folder, while the local run/debug configurations are kept in the <code>.idea\workspace.xml</code> . This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, PyCharm checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes

too much of the CPU and memory resources.

If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.

Before launch Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
------	-------------------	-------------



Alt+Insert

Click this icon to add a task to the list. Select the task to be added:

- Run External tool. Select this option to run an application which is external to PyCharm. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in PyCharm yet, add its definition. For more information, see [Configuring Third-Party Tools](#) and [External Tools](#).
- Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project.
- Run File Watchers. Select this option to have PyCharm apply all the currently active file watchers, see [Using File Watchers](#) for details.
- Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package.
- Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package.
- Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it.
- Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:
 - If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.
 - If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.
- Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see [CoffeeScript Support](#).
- Run Remote External tool: Add a remote SSH external tool. Refer to the section [Remote SSH External Tools](#) for details.
- Upload files to Remote Host. Select this option to have the application files automatically uploaded to the server according to the default server access configuration. For more information, see [Configuring Synchronization with a Web Server](#) and [Uploading and Downloading Files](#).



Alt+Delete

Click this icon to remove the selected task from the list.



Enter

Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.



Alt+Up

Click this icon to move the selected task one line up in the list.



Alt+Down

Click this icon to move the selected task one line down in the list.

Show this page

Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.

Active tool window

Select this option if you want the [Run/Debug](#) tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: JSTestDriver

This feature is supported in the Professional edition only.

In this dialog box, create a configuration to be used for running JavaScript unit tests in the browser against a JSTestDriver server. Configurations of this type enable running unit tests based on the [JSTestDriver Assertion](#), [Jasmine](#), and [JUnit](#) frameworks.

The dialog box is available when the **JSTestDriver** plugin is activated. The plugin is not bundled with PyCharm, but it can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).

On this page:

- [Configuration tab](#)
 - [Test](#)
 - [Server](#)
- [Debug tab](#)
- [Coverage tab](#)
- [Toolbar](#)
- [Common options](#)

Configuration tab

In this tab, specify the tests to run, the configuration files to use, and the server to run the tests against.

Test

In this area, tell PyCharm where to find the tests to execute and how to get [test runner configuration files](#) that define which test files to load and in which order.

The main approaches are:

- Specify the location of one or several previously created configuration files.
- Point at the target test file, test case, or test method, and then specify the location of the corresponding configuration file.

The way to find tests and configuration files is defined in the Test drop-down list. This choice determines the set of other controls in the area.

Item	Description	Available for
Test	In this drop-down list, specify how PyCharm will get test runner configuration files. <ul style="list-style-type: none">– All configuration files in directory: select this option to use all the test runner configuration files in a specific folder.– Configuration file: select this option to use a specific test runner configuration file.– JavaScript test file: select this option to have tests from a specific file executed using one of the previously defined configuration files.– Case: select this option to run a specific test case using one of the previously defined configuration files.– Method: select this option to run a specific test method using one of the previously defined configuration files.	
Directory	In this text box, specify the folder to look for test runner configuration files in. Type the path manually or click the Browse button  and select the required folder in the dialog box, that opens.	All configuration files in directory
Matched configuration files	This read-only field shows a list of all the <code>*jstd</code> and <code>JSTestDriver.conf</code> test runner configuration files detected in the specified folder .	All configuration files in directory
Configuration file	In this text box, specify the test runner configuration file to use. Type the path manually or click the Browse button  and select the required file in the dialog box that opens.	Configuration file
JS test file	In this text box, specify the JavaScript files with tests to be executed. Type the path manually or click the Browse button  and select the required file in the dialog box, that opens.	JavaScript test file Case Method
Case	In this text box, type the name of the target case from the specified JavaScript file .	Case Method
Method	In this text box, type the name of the target method from the specified test case within the specified JavaScript file .	Method

Server

In this area, appoint the test server to run tests against.

Item	Description
At address	Choose this option to have the test execution handles by a remote test server. In the text box, specify the URL address to access the server through.
Running in IDE	Choose this option to have test execution handles through the JSTestDriver server that comes bundled with PyCharm and can be launched from it.
Test Connection	Click this button to check that the specified test server is accessible. The server must be running. Start the server from PyCharm or manually, according to the server-specific instructions.

Debug tab

In this tab, appoint the browser to debug the unit test in when two or more browsers are captured simultaneously.

ItemDescription

Debug From this drop-down list, choose the browser to debug the specified tests in when two browsers are captured at a time. The available options are:

- Chrome
- Firefox

Coverage tab

In this tab, specify the files that you do not want to be involved in coverage analysis.

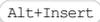
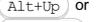
ItemDescription

Exclude file path In this area, create a list of files to be skipped during coverage analysis.

- To add an item to the list, click the Add button **+** and choose the required file in the [dialog that opens](#).
- To delete a file from the list so coverage is measured for it, select the file and click the Remove button **-**.

Toolbar

ItemShortcutDescription

		Click this button to add a new configuration to the list.
		Click this button to remove the selected configuration from the list.
		Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
	 or 	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the  buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

ItemDescription

Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. The shared run/debug configurations are kept in separate xml files under <code>.idea\runConfigurations</code> folder, while the local run/debug configurations are kept in the <code>.idea\workspace.xml</code> . This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, PyCharm checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

ItemKeyboardDescription shortcut

		Click this icon to add a task to the list. Select the task to be added: <ul style="list-style-type: none">– Run External tool. Select this option to run an application which is external to PyCharm. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in PyCharm yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools.– Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project.– Run File Watchers. Select this option to have PyCharm apply all the currently active file watchers, see Using File Watchers for details.– Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the <code>gruntfile.js</code> where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the <code>grunt-cli</code> package.
---	---	--

- Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package.
- Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it.
- Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:
 - If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.
 - If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.
- Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see [CoffeeScript Support](#).
- Run Remote External tool: Add a remote SSH external tool. Refer to the section [Remote SSH External Tools](#) for details.

		Click this icon to remove the selected task from the list.
		Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
		Click this icon to move the selected task one line up in the list.
		Click this icon to move the selected task one line down in the list.
Show this page	<input type="checkbox"/>	Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window	<input type="checkbox"/>	Select this option if you want the Run/Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Lettuce

This feature is supported in the Professional edition only.

Use this dialog box to create a run/debug configuration for Lettuce tests.

In this section:

- [Configuration tab](#)
- [Logs tab](#)
- [Toolbar](#)
- [Common options](#)

Configuration tab

ItemDescription

Feature files or folders	<p>In this text field, type the fully-qualified names of the feature files or directories which contain feature files.</p> <p>Multiple names should be delimited with .</p> <p>Use the browse button to locate the desired paths in the file system.</p>
Params	<p>In this text field, type the Lettuce-specific parameters to be passed to the tests.</p>
Scenario	<p>Type the name of the scenario to be executed. If this field is left blank, all the available scenarios in the specified feature files will be executed.</p>
Environment	
Project	<p>Click this drop-down list to select one of the projects, opened in the same PyCharm window, where this run/debug configuration should be used. If there is only one open project, this field is not displayed.</p>
Environment variable	<p>This field shows the list of environment variables. If the list contains several variables, they are delimited with semicolons.</p> <p>To fill in the list, click the browse button, or press <code>Shift+Enter</code> and specify the desired set of environment variables in the Environment Variables dialog box.</p> <p>To create a new variable, click <code>+</code>, and type the desired name and value.</p>
Python Interpreter	<p>Select one of the pre-configured Python interpreters from the drop-down list.</p> <p>Note that you can select a remote interpreter, as well as the local one. If a remote interpreter is selected, you have to specify path mappings in the corresponding field that appears.</p>
Interpreter options	<p>In this field, specify the string to be passed to the interpreter. If necessary, click <code>⌨</code>, and type the string in the editor.</p>
Working directory	<p>Specify a directory to be used by the running task.</p> <ul style="list-style-type: none">– When a default run/debug configuration is created by the keyboard shortcut <code>Ctrl+Shift+F10</code>, or by choosing Run on the context menu of a script, the working directory is the one that contains the executable script. This directory may differ from the project directory.– When this field is left blank, the <code>bin</code> directory of the PyCharm installation will be used.
Path mappings	<p>This field appears, if a remote interpreter has been selected in the field Python interpreter.</p> <p>Click the browse button <code>⌨</code> to define the required mappings between the local and remote paths. In the Edit Path Mappings dialog box, use <code>+</code>/<code>-</code> buttons to create new mappings, or delete the selected ones.</p>
Add content roots to PYTHONPATH	<p>Select this check box to add all content roots of your project to the environment variable PYTHONPATH;</p>
Add source roots to PYTHONPATH	<p>Select this check box to add all source roots of your project to the environment variable PYTHONPATH;</p>

Docker container settings

Warning! This field only appears when [Docker-based remote interpreter](#) has been selected for a project.

Note Speaking about the correspondence of settings with some options (`--net`, `--link`, etc.), note that these options come from [Docker command line arguments](#).

Click `⌨` to open the dialog and specify the following settings:

- **Disable networking:** select this check box to have the networking disabled. This corresponds to `--net="none"`, which means that inside a container the external network resources are not available.
- **Network mode:** corresponds to the other values of the option `--net`.
 - `bridge` is the default value. An IP address will be allocated for container on the bridge's network and traffic will be routed through this bridge to the container.

Containers can communicate via their IP addresses by default. To communicate by name, they must be linked.

- `host`: use the host's network stack inside the container.
- `container:<name|id>`: use the network stack of another container, specified via its `name` or `id`.

Refer to the [Network settings](#) documentation for details.

- **Links:** Use this section to link the container to be created with the other containers. This is applicable to `Network mode = bridge` and corresponds to the `--link` option.
- **Publish all ports:** If the check box is selected, all the container's exposed ports are bound to a random host port.

See e.g. [EXPOSE \(incoming ports\)](#) in [Docker run reference](#). This corresponds to the option `--publish-all`.

- **Port bindings:** Use this field to specify the Container port/protocol - Host IP address/port mappings.

E.g. `8080 tcp 127.0.0.1 18080` would bind the TCP port `8080` inside the container to port `18080` on the `localhost` or `127.0.0.1` interface on the host machine.

For more info, see e.g. [Connect using network port mapping](#) in [Legacy container links](#).

- Extra hosts: This corresponds to the `--add-host` option. Refer to the page [Managing /etc/hosts](#) for details.
- Volume bindings: Use this field to specify the bindings between the special folders-volumes and the folders of the computer, where the Docker daemon runs. This corresponds to the `-v` option.

See [Managing data in containers](#) for details.

- Environment variables: Use this field to specify the list of environment variables and their values. This corresponds to the `-e` option. Refer to the page [ENV \(environment variables\)](#) for details.

Click  to expand the tables. Click ,  or  to make up the lists.

Logs tab

Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the [Run](#) or [Debug tool window](#).

ItemDescription

Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window .
Log File Entry	The read-only fields in this column list the log files to show. The list can contain: <ul style="list-style-type: none"> – Full paths to specific files. – Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.
Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point to the desired location in the dialog that opens .
Show console when a message is printed to standard output stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.
Show console when a message is printed to standard error stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.
	Click this button to open the Edit Log Files Aliases dialog where you can select a new log entry and specify an alias for it.
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog .
	Click this button to remove the selected log entry from the list.
	Click this button to edit the select log file entry. The button is available only when an entry is selected.

Toolbar

ItemShortcutDescription

		Click this button to add a new configuration to the list.
		Click this button to remove the selected configuration from the list.
		Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
	 or 	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the  buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

ItemDescription

Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. The shared run/debug configurations are kept in separate xml files under <code>.idea\runConfigurations</code> folder, while the local run/debug configurations are kept in the <code>.idea\workspace.xml</code> .

This check box is not available when editing the run/debug configuration defaults.

Single instance only If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, PyCharm checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place.

This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources.

If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.

Before launch Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

**ItemKeyboardDescription
shortcut**

  Click this icon to add a task to the list. Select the task to be added:

- Run External tool. Select this option to run an application which is external to PyCharm. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in PyCharm yet, add its definition. For more information, see [Configuring Third-Party Tools](#) and [External Tools](#).
- Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project.
- Run File Watchers. Select this option to have PyCharm apply all the currently active file watchers, see [Using File Watchers](#) for details.
- Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package.
- Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package.
- Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it.
- Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:
 - If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.
 - If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.
- Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see [CoffeeScript Support](#).
- Run Remote External tool: Add a remote SSH external tool. Refer to the section [Remote SSH External Tools](#) for details.
- Upload files to Remote Host. Select this option to have the application files automatically uploaded to the server according to the default [server access configuration](#). For more information, see [Configuring Synchronization with a Web Server](#) and [Uploading and Downloading Files](#).

  Click this icon to remove the selected task from the list.

  Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.

  Click this icon to move the selected task one line up in the list.

  Click this icon to move the selected task one line down in the list.

Show this page Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.

Active tool window Select this option if you want the [Run/Debug](#) tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Mocha

This feature is supported in the Professional edition only.

In this dialog box, create configurations for running and debugging JavaScript unit tests using the Mocha test framework.

On this page:

- [Getting access to the Run/Debug Configuration: Mocha dialog](#)
- [Mocha-specific configuration settings](#)
- [Toolbar](#)
- [Common options](#)

Getting access to the Run/Debug Configuration: Mocha dialog

1. **Install** and **enable** the Node.js plugin. The plugin is not bundled with PyCharm, but it can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).
2. Download and install the [NodeJS](#) runtime environment that contains the [Node Package Manager\(npm\)](#).
3. Using the [Node Package Manager](#), install the [Mocha test framework](#) as described in [Testing JavaScript with Mocha](#).

Mocha-specific configuration settings

ItemDescription

Node.js interpreter	In this field, specify the NodeJS installation home. Type the path to the NodeJS executable file manually, or click the  button and select the location in the dialog box, that opens. If you have appointed one of the installations as default , the field displays the path to its executable file.
Node options	In this text box, type the NodeJS-specific command line options to be passed to the NodeJS executable file. See Node Parameters for details.
Working directory	In this field, specify the working directory of the application. All references in test scripts will be resolved relative to this folder, unless such references use full paths. By default, the field shows the project root folder . To change this predefined setting, choose the desired folder from the drop-down list, or type the path manually, or click the Browse button  and select the location in the dialog box, that opens.
Environment variables	In this field, specify the environment variables for the NodeJS executable file, if applicable. Click the Browse button  to the right of the field and configure a list of variables in the Environment Variables dialog box, that opens: <ul style="list-style-type: none">– To define a new variable, click the Add toolbar button  and specify the variable name and value.– To discard a variable definition, select it in the list and click the Delete toolbar button .– Click OK, when ready The definitions of variables are displayed in the Environment variables read-only field with semicolons as separators. The acceptable variables are: <ul style="list-style-type: none">– <code>NODE_PATH</code> : A  -separated list of directories prefixed to the module search path.– <code>NODE_MODULE_CONTEXTS</code> : Set to 1 to load modules in their own global contexts.– <code>NODE_DISABLE_COLORS</code> : Set to 1 to disable colors in the REPL.
Mocha package	In this field, specify the Mocha installation home <code>/npm/node_modules/mocha</code> . If you installed Mocha regularly through the Node Package Manager , PyCharm detects the Mocha installation home itself. Alternatively, type the path to executable file manually, or click the Browse button  and select the location in the dialog box, that opens.
User interface	From this drop-down list, choose the interface according to which the tests in the test folder are written. PyCharm will recognize only tests that comply with the chosen interface . If during test execution PyCharm runs against a test of another interface , the test session will be canceled with an error. This means that all the tests in the specified test folder must be written according to the same interface and this interface must be chosen from the drop-down list.
Tests	In this area, specify the tests to be executed. The available options are: <ul style="list-style-type: none">– All in directory: choose this option to run all the tests from files stored in a folder. In the Test directory field, specify the folder with the tests. To have PyCharm look for tests in the subfolders under the specified directory, select the Include subdirectories check box.– File pattern: choose this option to have PyCharm look for tests in all the files with the names that match a certain mask and specify the mask in the Test file pattern field.– Test file: choose this option to run only the tests from one file and specify the path to this file in the Test file field.– Suite: choose this option to run individual suites from a test file. In the Test file field, specify the file where the required suites are defined. Click the Suite name field and configure a list of suites to run. To add a suite to the list, click  and type the name of the required suite. To remove a suite, select it in the list and click .– Test: choose this option to run individual tests from a test file. In the Test file field, specify the file where the required tests are defined. Click the Test name field and configure a list of tests to run. To add a test to the list, click  and type the name of the required test. To remove a test, select it in the list and click .

Toolbar

ItemShortcutDescription

		Click this button to add a new configuration to the list.
		Click this button to remove the selected configuration from the list.
		Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
	 or 	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.

 Move into new folder / Create new folder Use this button to [create a new folder](#).
 If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created.

 Move run/debug configurations to a folder using drag-and-drop, or the  buttons.

 Sort configurations Click this button to sort configurations in alphabetical order.

Common options

ItemDescription

Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	<p>Select this check box to make the run/debug configuration available to other team members.</p> <p>The shared run/debug configurations are kept in separate xml files under <code>.idea\runConfigurations</code> folder, while the local run/debug configurations are kept in the <code>.idea\workspace.xml</code>.</p> <p>This check box is not available when editing the run/debug configuration defaults.</p>
Single instance only	<p>If this check box is selected, this run/debug configuration cannot be launched more than once.</p> <p>Every time a new run/debug configuration is launched, PyCharm checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place.</p> <p>This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources.</p> <p>If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.</p>
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

ItemKeyboardDescription shortcut

	 Click this icon to add a task to the list. Select the task to be added: <ul style="list-style-type: none"> – Run External tool. Select this option to run an application which is external to PyCharm. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in PyCharm yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools. – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run File Watchers. Select this option to have PyCharm apply all the currently active file watchers, see Using File Watchers for details. – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the <code>gruntfile.js</code> where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the <code>grunt-cli</code> package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the <code>gulpfile.js</code> where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the <code>gulp</code> package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the <code>package.json</code> file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: <ul style="list-style-type: none"> – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support. – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details. – Upload files to Remote Host. Select this option to have the application files automatically uploaded to the server according to the default server access configuration. For more information, see Configuring Synchronization with a Web Server and Uploading and Downloading Files.
	 Click this icon to remove the selected task from the list.
	 Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	 Click this icon to move the selected task one line up in the list.
	Click this icon to move the selected task one line down in the list.

Alt+Down

Show this page

Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.

Active tool window

Select this option if you want the [Run/Debug](#) tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Meteor

This feature is supported in the Professional edition only.

Use this dialog box to create configurations for running and debugging **Meteor** applications.

The dialog box is available when the **Meteor** plugin is enabled. The plugin is not bundled with PyCharm, but it can be installed from the **JetBrains plugin repository** as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).

On this page:

- [Configuration tab](#)
- [Browser / Live Edit tab](#)
- [Toolbar](#)
- [Common options](#)

Configuration tab

ItemDescription

Name	In this text box, specify the name of the current run/debug configuration.
Share	Select this check box to make the run/debug configuration available to other team members. The shared run/debug configurations are kept in separate xml files under <code>.idea\runConfigurations</code> folder, while the local run/debug configurations are kept in the <code>.idea\workspace.xml</code> . This check box is not available when editing the run/debug configuration defaults.
Meteor Executable	In this field, specify the location of the Meteor executable file (see Installing Meteor).
Program Arguments	In this field, specify the command line additional parameters to be passed to the executable file on start up, if applicable. These can be, for example, <code>--dev</code> , <code>--test</code> , or <code>--prod</code> to indicate the environment in which the application is running (development, test, or production environments) so different resources are loaded on start up.
Working Directory	In this field, specify the folder under which the application files to run are stored. This folder must contain a <code>.meteor</code> folder in the root to be treated a Meteor project . By default, the field shows the path to the PyCharm project root. Technically, several Meteor projects that implement different applications can be combined within one single PyCharm project. To run and debug these applications independently, create a separate run configuration for each of them with the relevant working directory . To avoid port conflicts, these run configurations should use different ports. In the Program Arguments field, specify a separate port for each run configuration in the format <code>--port=<port_number></code> .
Environment Variables	In this field, specify the environment variables for the Meteor executable file, if applicable.

Browser / Live Edit tab

In this tab, configure the behaviour of the browser and enable debugging the client-side code of the application. This functionality is provided through a `JavaScript Debug` run configuration, so technically, PyCharm creates separate run configurations for the server-side and the client-side code, but you specify all your settings in one dedicated **Meteor** run configuration.

ItemDescription

Open Browser	In the text box in this area, specify the URL address to open the application at. If you select the After Launch check box, the browser will open this page automatically after the application starts. Alternatively you can view the same result by opening the page with this URL address in the browser of your choice manually.
After Launch	Select this check box to have a browser started automatically after a debugging session is launched. Specify the browser to use in the drop-down list next to the check box. <ul style="list-style-type: none">– To use the system default browser, choose Default.– To use a custom browser, choose it from the list. Note that Live Edit is fully supported only in Chrome.– To configure browsers, click the Browse button  and adjust the settings in the Web Browsers dialog box that opens. For more information, see Configuring Browsers.
With JavaScript Debugger	Select this check box to enable debugging the client-side code in the selected browser.

Toolbar

ItemShortcutDescription

	<code>Alt+Insert</code>	Click this button to add a new configuration to the list.
	<code>Alt+Delete</code>	Click this button to remove the selected configuration from the list.
	<code>Ctrl+D</code>	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
	<code>Alt+Up</code> or <code>Alt+Down</code>	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to

the newly created folder. If only a category is in focus, an empty folder is created.

Move run/debug configurations to a folder using drag-and-drop, or the   buttons.



Sort configurations

Click this button to sort configurations in alphabetical order.

Common options

ItemDescription

Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. The shared run/debug configurations are kept in separate xml files under <code>.idea\runConfigurations</code> folder, while the local run/debug configurations are kept in the <code>.idea\workspace.xml</code> . This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, PyCharm checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.

Before launch Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

ItemKeyboardDescription shortcut



 Alt+Insert

Click this icon to add a task to the list. Select the task to be added:

- Run External tool. Select this option to run an application which is external to PyCharm. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in PyCharm yet, add its definition. For more information, see [Configuring Third-Party Tools](#) and [External Tools](#).
- Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run.
This option is available only if you have already at least one run/debug configuration in the current project.
- Run File Watchers. Select this option to have PyCharm apply all the currently active [file watchers](#), see [Using File Watchers](#) for details.
- Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool.
Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package.
- Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool.
Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package.
- Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with.
Specify the location of the Node.js interpreter and the parameters to pass to it.
- Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:
 - If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.
 - If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.
- Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see [CoffeeScript Support](#).
- Run Remote External tool: Add a remote SSH external tool. Refer to the section [Remote SSH External Tools](#) for details.
- Upload files to Remote Host. Select this option to have the application files automatically uploaded to the server according to the [default server access configuration](#). For more information, see [Configuring Synchronization with a Web Server](#) and [Uploading and Downloading Files](#).



 Alt+Delete

Click this icon to remove the selected task from the list.



 Enter

Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.



 Alt+Up

Click this icon to move the selected task one line up in the list.



 Alt+Down

Click this icon to move the selected task one line down in the list.

Show this page

Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.

Active tool window

Select this option if you want the [Run/Debug](#) tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Node JS

This feature is supported in the Professional edition only.

Warning! The following is only valid when Node.js Plugin is installed and enabled!

In this dialog box, create configurations for running and debugging of **NodeJS** applications locally. "Locally" in the current context means that PyCharm itself starts the **NodeJS** runtime environment installed on your computer, whereupon initiates a running or debugging session.

On this page:

- [Getting access to the Run/Debug Configuration: NodeJS dialog](#)
- [Configuration tab](#)
- [Browser / Live Edit tab](#)
- [V8 Profiling tab](#)
- [Toolbar](#)
- [Common options](#)

Getting access to the Run/Debug Configuration: NodeJS dialog

1. **Install** and **enable** the Node.js plugin. The plugin is not bundled with PyCharm, but it can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).
2. Download and install the [NodeJS](#) runtime environment that contains the [Node Package Manager\(npm\)](#).

Configuration tab

ItemDescription

Node Interpreter	<p>In this field, specify the NodeJS installation home. Type the path to the NodeJS executable file manually, or click the  button and select the location in the dialog box, that opens.</p> <p>If you have appointed one of the installations as default, the field displays the path to its executable file.</p>
Node Parameters	<p>In this text box, type the NodeJS-specific command line options to be passed to the NodeJS executable file. The most common options are:</p> <p><code>--require coffee-script/register</code> Specify this parameter to have CoffeeScript files compiled into JavaScript on the fly during run. This mode requires that the <code>register.js</code> file, which is a part of the <code>coffee-script</code> package, should be located inside the project. Therefore you need to install the <code>coffee-script</code> package on the Node.js page locally, as described in Installing and Removing External Software Using Node Package Manager.</p> <p><code>--inspect</code> Specify this parameter when you are using Node.js v7 for Chrome Debugging Protocol support. Otherwise, by default the debug process will use V8 Debugging Protocol.</p> <p>For a full list, see NodeJS command line options.</p>
Working Directory	<p>In this field, specify the working directory of the application. All references in the starting NodeJS application file, for example, imports, will be resolved relative to this folder, unless such references use full paths.</p> <p>By default, the field shows the project root folder. To change this predefined setting, choose the desired folder from the drop-down list, or type the path manually, or click the Browse button  and select the location in the dialog box, that opens.</p>
JavaScript File	<p>In this field, specify the full path to the file to start running or debugging the application from.</p> <p>If you are going to debug CoffeeScript, specify the path to the generated JavaScript file with source maps. The file can be generated externally or through compilation using file watchers. For more details, see Compiling CoffeeScript to JavaScript.</p>
Application Parameters	<p>In this text box, type the NodeJS-specific arguments to be passed to the application start file through the process.argv array.</p>
Environment Variables	<p>In this field, specify the environment variables for the NodeJS executable file, if applicable. Click the Browse button  to the right of the field and configure a list of variables in the Environment Variables dialog box, that opens:</p> <ul style="list-style-type: none">– To define a new variable, click the Add toolbar button  and specify the variable name and value.– To discard a variable definition, select it in the list and click the Delete toolbar button .– Click OK, when ready <p>The definitions of variables are displayed in the Environment variables read-only field with semicolons as separators. The acceptable variables are:</p> <ul style="list-style-type: none">– <code>NODE_PATH</code> : A <code>:</code>-separated list of directories prefixed to the module search path.– <code>NODE_MODULE_CONTEXTS</code> : Set to 1 to load modules in their own global contexts.– <code>NODE_DISABLE_COLORS</code> : Set to 1 to disable colors in the REPL.
Docker container settings	<p>In this field, type the settings manually, or click  next to the field and specify the settings in the Edit Docker Container Settings dialog that opens, or select the Auto configure check box to have PyCharm do it automatically.</p> <p>The field is available only when a remote Node.js interpreter running on a Docker container is chosen.</p>
Auto configuration	<p>Select this check box to have PyCharm configure the container settings. In the Automatic configuration mode:</p> <ul style="list-style-type: none">– PyCharm creates a new image and installs the <code>npm</code> modules on it.– To avoid reinstalling the modules on every start of the container and messing up with the local and system-specific dependencies, PyCharm copies the <code>package.json</code> configuration file to the <code>/tmp/project_modules</code> folder on the image, runs the <code>npm install</code> command, and then copies the modules to the project folder in the container. Changing <code>package.json</code> in the project results in re-building the image.– PyCharm runs the Docker container with the created image and binds your project folder to <code>/opt/project</code> folder in the Docker container to ensure synchronization on update. In addition to that, <code>/opt/project/node_modules</code> will be mapped to the OS temporary directory. <p>With automatic configuration, you still need to bind the port that your application is running on with the port of the container. Those exposed ports are available on the Docker host's IP address (by default 192.168.99.100). Such binding is required when you debug the client side of a Node.js Express application so you need to open the browser from your computer and access the application at the host of the container through the port specified in the</p>

application.

1. Click  next to the Docker Container Settings field.
2. In the Edit Docker Container Settings dialog that opens, expand the Port bindings area.
3. Click . The Port bindings dialog box opens. In this dialog box, map the ports as follows:
 - In the Container port text box, type the port specified in your application.
 - In the Host port text box, type the port through which you want to open the application in the browser from your computer.
 - In the Host IP text box, type the IP address of the Docker's host, the default IP is 192.168.99.100. The host is specified in the API URL field on the [Docker](#) page of the .
 - Click OK to return to the Edit Docker Container Settings dialog where the new port mapping is added to the list.
4. Click OK to return to the [Run/Debug Configuration: Node JS](#) dialog.

The field is available only when a remote Node.js interpreter running on a Docker container is chosen.

Browser / Live Edit tab

In this tab, configure the behaviour of the browser and enable debugging the client-side code of the application. This functionality is provided through a [JavaScript Debug](#) run configuration, so technically, PyCharm creates separate run configurations for the server-side and the client-side code, but you specify all your settings in one dedicated [NodeJS](#) run configuration.

However, activating [Live Edit](#) in this tab lets you invoke [Live Edit](#) without creating a separate configuration, during a debugging session launched through the Node.js configuration.

For more information, see [Live Editing of HTML, CSS, and JavaScript](#).

ItemDescription

Open Browser	In the text box in this area, specify the URL address to open the application at. If you select the After Launch check box, the browser will open this page automatically after the application starts. Alternatively you can view the same result by opening the page with this URL address in the browser of your choice manually.
After launch	Select this check box to have a browser started automatically after a debugging session is launched. Specify the browser to use in the drop-down list next to the check box. <ul style="list-style-type: none">– To use the system default browser, choose Default.– To use a custom browser, choose it from the list. Note that Live Edit is fully supported only in Chrome.– To configure browsers, click the Browse button  and adjust the settings in the Web Browsers dialog box that opens. For more information, see Configuring Browsers.

With JavaScript debugger Select this check box to enable debugging the client-side code in the selected browser.

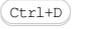
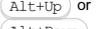
V8 Profiling tab

ItemDescription

Record CPU profiling info	Select this check box to start logging the CPU profiling data when the application is launched. The controls in the area below become enabled. Specify the following: <ul style="list-style-type: none">– Log folder: in this field, specify the folder to store recorded logs in. Profiling data are stored in V8 log files <code>isolate-<session number></code>.– One log file for isolates:– Tick package: in this field, specify the tick package to use. Choose the relevant package from the Tick package drop-down list or click the  button next to it and choose the package in the dialog box that opens.– Gnuplot package: in this field, specify the location of the Gnuplot executable file to explore a timeline view that shows where V8 is spending time.
Allow taking heap snapshots	Select this check box to... The controls in the area below become enabled. Specify the following: <ul style="list-style-type: none">– V8 profiler package: in this field, specify the v8-profiler package to use. Choose the relevant package from the v8-profiler package drop-down list or click the  button next to it and choose the package in the dialog box that opens.– Communication port: in this field, specify the port through which PyCharm communicates with the profiler, namely, sends a command to take a snapshot when you click the Take Heap Snapshot button  on the toolbar of the Run tool window.

Toolbar

ItemShortcutDescription

		Click this button to add a new configuration to the list.
		Click this button to remove the selected configuration from the list.
		Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
	 or 	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the  buttons.



Sort configurations Click this button to sort configurations in alphabetical order.

Common options

ItemDescription

Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	<p>Select this check box to make the run/debug configuration available to other team members.</p> <p>The shared run/debug configurations are kept in separate xml files under <code>.idea\runConfigurations</code> folder, while the local run/debug configurations are kept in the <code>.idea\workspace.xml</code>.</p> <p>This check box is not available when editing the run/debug configuration defaults.</p>
Single instance only	<p>If this check box is selected, this run/debug configuration cannot be launched more than once.</p> <p>Every time a new run/debug configuration is launched, PyCharm checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place.</p> <p>This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources.</p> <p>If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.</p>
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

ItemKeyboardDescription shortcut

	Alt+Insert Click this icon to add a task to the list. Select the task to be added: <ul style="list-style-type: none"> – Run External tool. Select this option to run an application which is external to PyCharm. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in PyCharm yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools. – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run File Watchers. Select this option to have PyCharm apply all the currently active file watchers, see Using File Watchers for details. – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the <code>gruntfile.js</code> where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the <code>grunt-cli</code> package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the <code>gulpfile.js</code> where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the <code>gulp</code> package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the <code>package.json</code> file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: <ul style="list-style-type: none"> – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support. – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details. – Upload files to Remote Host. Select this option to have the application files automatically uploaded to the server according to the default server access configuration. For more information, see Configuring Synchronization with a Web Server and Uploading and Downloading Files.
	Alt+Delete Click this icon to remove the selected task from the list.
	Enter Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	Alt+Up Click this icon to move the selected task one line up in the list.
	Alt+Down Click this icon to move the selected task one line down in the list.
Show this page	Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window	Select this option if you want the Run/Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

This feature is supported in the Professional edition only.

Warning! The following is only valid when Node.js Plugin is installed and enabled!

File | Settings | Languages and Frameworks | Node.js and NPM for Windows and Linux

PyCharm | Preferences | Languages and Frameworks | Node.js and NPM for macOS

The dialog box opens when you click the Browse button  next to the Node Interpreter drop-down list in the [Run/Debug Configuration: Node JS](#) dialog or on the [Node.js and NPM](#) page.

Use this dialog box to configure Node.js installations as local and remote interpreters.

The term **local Node.js interpreter** denotes a Node.js installation on your computer. The term **remote Node.js interpreter** denotes a Node.js installation on a remote host or in a virtual environment set up in a **Vagrant** instance. See [Configuring Node.js Interpreters](#) for details.

When the dialog box opens from the [Node.js and NPM](#) page, you can only configure local interpreters installed on your computer. When the dialog box is accessed from the [Run/Debug Configuration: Node JS](#) dialog, both local and remote interpreters can be configured.

ItemTooltipDescription

Node.js Interpreters		The list shows all the configured Node.js interpreters, both local and remote ones. For local interpreters, PyCharm also shows the path to the Node.js executable file and to the associated npm executable file, see Installing and Removing External Software Using Node Package Manager .
	Add	Click this button to add a new Node.js interpreter to the list. From the drop-down menu, choose All Local or Add Remote. Note that the Add Remote item is available only you opened the dialog box from the Run/Debug Configuration: Node JS dialog. Depending on your choice, either select the relevant local Node.js installation or configure a remote interpreter in the Configure Node.js Remote Interpreter Dialog that opens.
	Delete	Click this button to remove the selected interpreter from the list.
	Edit	Click this button to create a new interpreter with the settings copied from the selected one.
Node interpreter	This read-only field shows the path to the selected local interpreter.	
Npm package	In this field, specify the Node package manager (npm) associated with the selected interpreter. Choose the relevant npm from the drop-down list or click  next to it and in the dialog box that opens choose the location of the npm to use. Alternatively, you can specify the path to the Yarn package manager if you want to use it instead of npm.	
	The field is available only if the selected interpreter is of the type local.	

Warning! The following is only valid when Node.js Plugin is installed and enabled!

The dialog box is available only when the **Node.js Remote Interpreter** plugin is enabled. The plugin is not bundled with PyCharm, but it can be installed from the **JetBrains plugin repository** as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).

The dialog box opens when you click the Add toolbar button  in the **Node.js Interpreters Dialog** and choose Remote... from the drop-down menu. This menu item is available only when you open the **Node.js Interpreters Dialog** from the **Run/Debug Configuration: Node JS**.

Use this dialog box to configure access to Node.js installations on remote hosts or in development environments set up in **Vagrant** instances.

ItemDescription

SSH Credentials	<p>Choose this option to configure access to a Node.js interpreter on a remote host through SSH credentials. In the fields of the dialog box, specify the following:</p> <ul style="list-style-type: none"> – Host: in this field, type the name of the host on which the interpreter is installed. – Port: in this field, type the port which the SSH server on the remote host listens to. The default port number is 22. – User name: in the field, type the user name under which you are registered on the SSH server. – Auth type: from this drop-down list, choose the authentication method. <ul style="list-style-type: none"> – To access the host through a password, choose Password from the Auth type drop-down list and type the password. – To access the host through a pair of SSH keys, choose Key pair, then specify the path to the file where your private key is stored or the passphrase if you have configured it during the generation of the key pair. <p>To use an interpreter configuration, you need path mappings that set correspondence between the project folders, the folders on the server to copy project files to, and the URL addresses to access the copied data on the server. PyCharm first attempts to retrieve path mappings itself by processing all the available application-level configurations. If PyCharm finds the configurations with the same host as the one specified above, in the Host field, the mappings from these configurations are merged automatically. If no configurations with this host are found, PyCharm displays an error message informing you that path mappings are not configured.</p> <p>To fix the problem, open the Deployment page under the Build, Execution, Deployment node, select the server access configuration in question, switch to the Mappings tab, and map local folders to folders on the server as described in Creating a Remote Server Configuration, section Mapping Local Folders to Folders on the Server and the URL Addresses to Access Them.</p>
Vagrant	<p>This option is available only when the Vagrant repository plugin is installed and enabled. The plugin is not bundled with PyCharm, but it can be installed from the JetBrains plugin repository as described in Installing, Updating and Uninstalling Repository Plugins and Enabling and Disabling Plugins.</p> <p>Choose this option to configure access to a Node.js interpreter installed in a Vagrant instance using your Vagrant credentials. Technically, it is the folder where the VagrantFile configuration file for the desired environment is located. Based on this setting, PyCharm detects the Vagrant host and shows it as a link in the Vagrant Host URL read-only field.</p> <p>To use an interpreter configuration, you need path mappings that set correspondence between the project folders, the folders on the server to copy project files to, and the URL addresses to access the copied data on the server. PyCharm evaluates path mappings from the VagrantFile configuration file.</p>
Deployment Configuration	<p>This option is available only when the Remote Hosts Access plugin is enabled. The plugin is bundled with PyCharm and activated by default. If the plugin is not activated, enable it on the Plugins page of the Settings / Preferences Dialog as described in Enabling and Disabling Plugins.</p> <p>Choose this option to configure access to a Node.js interpreter on a remote host using a server access configuration. This option is available only if you have at least one server access configuration of the type SFTP, see Creating a Remote Server Configuration.</p> <p>From the Deployment Configuration drop-down list, choose the server access configuration of the type SFTP according to which you want PyCharm to connect to the target host. If the settings specified in the chosen configuration ensure successful connection, PyCharm displays the URL address of the target host as a link in the Deployment Host URL field.</p> <p>To use an interpreter configuration, you need path mappings that set correspondence between the project folders, the folders on the server to copy project files to, and the URL addresses to access the copied data on the server. By default, PyCharm retrieves path mappings from the chosen server access (deployment) configuration. If the configuration does not contain path mappings, PyCharm displays the corresponding error message.</p> <p>To fix the problem, open the Deployment page under the Build, Execution, Deployment node, select the relevant server access configuration, switch to the Mappings tab, and map the local folders to the folders on the server as described in Creating a Remote Server Configuration, section Mapping Local Folders to Folders on the Server and the URL Addresses to Access Them.</p>
Docker	<p>This option is available only when the Node.js, Node.js Remote Interpreter, and Docker Integration plugins are enabled. The plugins are bundled with PyCharm and activated by default. If the plugins are not activated, enable them on the Plugins page of the Settings / Preferences Dialog as described in Enabling and Disabling Plugins.</p> <p>Choose this option to configure access to a Node.js interpreter running in a Docker container.</p> <ol style="list-style-type: none"> 1. In the Server field, specify the Docker configuration to use, see Working with Docker: Process overview. Choose a configuration from the drop-down list or click  next to it and create a new configuration in the Docker dialog box that opens. 2. In the Image name field, specify the base Docker image to use. Choose one of the previously downloaded or your custom images from the drop-down list or type the image name manually, for example, <code>node:argon</code> or <code>mhart/alpine-node</code>. When you later launch the run configuration, Docker will search for the specified image on your machine. If the search fails, the image will be downloaded from the image repository specified on the Docker Registry page. 3. The Node.js interpreter path field shows the location of the default Node.js interpreter from the specified image. 4. When you click OK, PyCharm closes the Configure Node.js Remote Interpreter Dialog and brings you to the Node.js Interpreters Dialog where the new interpreter configuration is added to the list. Click OK to return to the run configuration.
Node.js Interpreter Path	<p>In this field, specify the location of the Node.js executable file in accordance with the configuration of the selected remote development environment. By default PyCharm suggests the <code>/usr/bin/node</code> folder for remote hosts and Vagrant instances and <code>node</code> for Docker containers. To specify another folder, click the Browse button  and choose the relevant folder in the dialog box that opens. Note that the Node.js home directory must be open for edit. When you click OK, PyCharm checks whether the Node.js executable is actually stored in the specified folder.</p> <ul style="list-style-type: none"> – If no Node.js executable is found, PyCharm displays an error message asking you whether to continue searching or save the interpreter configuration anyway. – If the Node.js executable is found, you return to the Node.js Interpreters where the installation folder and the detected version of the Node.js interpreter are displayed.

Run/Debug Configuration: Node JS Remote Debug

This feature is supported in the Professional edition only.

Warning! The following is only valid when Node.js Plugin is installed and enabled!

In this dialog box, create configurations for debugging already running [Node.js](#) applications. This approach gives you the possibility to re-start a debugging session without re-starting the NodeJS server.

On this page:

- [Getting access to the Run/Debug Configuration: NodeJS Remote Debug dialog](#)
- [NodeJS Remote Debug-specific configuration settings](#)
- [Toolbar](#)
- [Common options](#)

Getting access to the Run/Debug Configuration: NodeJS Remote Debug dialog

1. **Install** and **enable** the Node.js plugin. The plugin is not bundled with PyCharm, but it can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).
2. Download and install the [NodeJS](#) runtime environment that contains the [Node Package Manager\(npm\)](#).

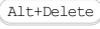
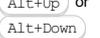
NodeJS Remote Debug-specific configuration settings

ItemDescription

Host	In this text box, specify the host where the application is running.
Debug port	In this text box, specify the port to connect to. Do one of the following: <ul style="list-style-type: none">– Copy the port number from the information message in the Run tool window that controls the running application.– Copy the port number from the Node parameter text box in the Run/Debug Configuration:NodeJS dialog box.

Toolbar

ItemShortcutDescription

		Click this button to add a new configuration to the list.
		Click this button to remove the selected configuration from the list.
		Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
		Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the  buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

ItemDescription

Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. The shared run/debug configurations are kept in separate xml files under <code>.idea\runConfigurations</code> folder, while the local run/debug configurations are kept in the <code>.idea\workspace.xml</code> . This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, PyCharm checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.

Before launch Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
------	-------------------	-------------

		Click this icon to add a task to the list. Select the task to be added: <ul style="list-style-type: none">– Run External tool. Select this option to run an application which is external to PyCharm. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in PyCharm yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools.– Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project.– Run File Watchers. Select this option to have PyCharm apply all the currently active file watchers, see Using File Watchers for details.– Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the <code>gruntfile.js</code> where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the <code>grunt-cli</code> package.– Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the <code>gulpfile.js</code> where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the <code>gulp</code> package.– Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the <code>package.json</code> file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it.– Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:<ul style="list-style-type: none">– If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.– If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.– Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support.– Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
		Click this icon to remove the selected task from the list.
		Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
		Click this icon to move the selected task one line up in the list.
		Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run/Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: NodeUnit

This feature is supported in the Professional edition only.

Warning! The following is only valid when Node.js Plugin is installed and enabled!

In this dialog box, create configurations to run unit tests for NodeJS applications.

On this page:

- [Getting access to the Run/Debug Configuration: NodeUnit dialog](#)
- [NodeUnit-specific configuration settings](#)
- [Toolbar](#)
- [Common options](#)

Getting access to the Run/Debug Configuration: NodeUnit dialog

1. **Install** and **enable** the Node.js plugin. The plugin is not bundled with PyCharm, but it can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).
2. Download and install the [NodeJS](#) runtime environment that contains the [Node Package Manager\(npm\)](#).
3. Install the **Nodeunit** testing framework in one of the following ways:
 - Download the framework at <https://github.com/caolan/nodeunit> and install it according to the official instructions.
 - Install the `nodeunit` package using the [Node Package Manager \(NPM\)](#) as described in [Installing and Removing External Software Using Node Package Manager](#).

NodeUnit-specific configuration settings

ItemDescription

Node Interpreter	In this field, specify the NodeJS installation home. Type the path to the NodeJS executable file manually, or click the Browse button  and select the location in the dialog box, that opens.
Working Directory	In this text box, specify the folder to find tests under. This can be the project root folder or the parent directory for the <code>test</code> folder. Type the path manually or click the Browse button  and select the location in the dialog box, that opens.
Environment Variables	<p>In this field, specify the environment variables for the NodeJS executable file, if applicable. Click the Browse button  to the right of the field and configure a list of variables in the Environment Variables dialog box, that opens:</p> <ul style="list-style-type: none">– To define a new variable, click the Add toolbar button  and specify the variable name and value.– To discard a variable definition, select it in the list and click the Delete toolbar button .– Click OK, when ready <p>The definitions of variables are displayed in the Environment variables read-only field with semicolons as separators. The acceptable variables are:</p> <ul style="list-style-type: none">– <code>NODE_PATH</code> : A <code>:</code>-separated list of directories prefixed to the module search path.– <code>NODE_MODULE_CONTEXTS</code> : Set to 1 to load modules in their own global contexts.– <code>NODE_DISABLE_COLORS</code> : Set to 1 to disable colors in the REPL.
Nodeunit Module	In this text box, specify the installation folder of the Nodeunit framework. Type the path manually or click the  button and choose the folder in the dialog box that opens.
Run	<p>From this drop-down list, choose the scope of tests to execute. The available options are:</p> <ul style="list-style-type: none">– All JavaScript test files in the directory: choose this option to to have PyCharm run all the test files in a folder. In the Directory text box below, specify the path to the test folder relative to the working directory.– JavaScript test file: choose this option to have a specific test executed. In the JavaScript test file text box, type the path to the file relative to the working directory.

Toolbar

ItemShortcutDescription

	 Alt+Insert	Click this button to add a new configuration to the list.
	 Alt+Delete	Click this button to remove the selected configuration from the list.
	 Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
	 Alt+Up or  Alt+Down	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	<p>Use this button to create a new folder.</p> <p>If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created.</p> <p>Move run/debug configurations to a folder using drag-and-drop, or the  buttons.</p>
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

ItemDescription

Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. The shared run/debug configurations are kept in separate xml files under <code>.idea\runConfigurations</code> folder, while the local run/debug configurations are kept in the <code>.idea\workspace.xml</code> . This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, PyCharm checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

ItemKeyboardDescription shortcut

	 Click this icon to add a task to the list. Select the task to be added: <ul style="list-style-type: none">– Run External tool. Select this option to run an application which is external to PyCharm. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in PyCharm yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools.– Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project.– Run File Watchers. Select this option to have PyCharm apply all the currently active file watchers, see Using File Watchers for details.– Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the <code>Gruntfile.js</code> where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the <code>grunt-cli</code> package.– Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the <code>Gulpfile.js</code> where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the <code>gulp</code> package.– Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the <code>package.json</code> file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it.– Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:<ul style="list-style-type: none">– If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.– If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.– Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support.– Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.
	 Click this icon to remove the selected task from the list.
	 Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	 Click this icon to move the selected task one line up in the list.
	 Click this icon to move the selected task one line down in the list.
Show this page	Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window	Select this option if you want the Run/Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: NPM

This feature is supported in the Professional edition only.

Warning! The following is only valid when Node.js Plugin is installed and enabled!

In this dialog box, create configurations for running [npm scripts](#) locally. "Locally" in the current context means that PyCharm itself starts the NodeJS runtime environment installed on your computer, whereupon initiates script execution.

On this page:

- [Getting access to the Run/Debug Configuration: NPM dialog](#)
- [Configuration tab](#)
 - [Toolbar](#)
 - [Common options](#)

Getting access to the Run/Debug Configuration: NPM dialog

1. **Install** and **enable** the Node.js plugin. The plugin is not bundled with PyCharm, but it can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).
2. Download and install the [NodeJS](#) runtime environment that contains the [Node Package Manager\(npm\)](#).

Configuration tab

ItemDescription

Name	In this text box, specify the name of the run/debug configuration.
Share	Select this check box to make the run/debug configuration available to other team members. The shared run/debug configurations are kept in separate xml files under <code>.idea\runConfigurations</code> folder, while the local run/debug configurations are kept in the <code>.idea\workspace.xml</code> . This check box is not available when editing the run/debug configuration defaults.
package.json	In this field, specify the <code>package.json</code> file to run the scripts from. Choose the file from the drop-down list which shows all the <code>package.json</code> files detected in the current project or click  and choose the required <code>package.json</code> in the dialog box that opens.
Command	From this drop-down list, choose the <code>npm CLI command</code> to execute, by default <code>run-script</code> is selected. Learn more at npm documentation , under the section CLI Commands.
Scripts	From this drop-down list, choose the script to which the chosen command will be applied. The list contains all the scripts defined within the <code>scripts</code> property in the <code>Package.json</code> file.
Arguments	In this field, specify the command line arguments to execute a script with. Learn more at Arguments .
Node Interpreter	In this field, specify the NodeJS installation home. Type the path to the NodeJS executable file manually, or click the  button and select the location in the dialog box, that opens. If you have appointed one of the installations as default , the field displays the path to its executable file.
Node Options	In this text box, type the NodeJS-specific command line options to be passed to the NodeJS executable file. The acceptable options are: <code>--require coffee-script/register</code> Specify this parameter to have <code>CoffeeScript</code> files compiled into JavaScript on the fly during run. This mode requires that the <code>register.js</code> file, which is a part of the <code>coffee-script</code> package, should be located inside the project. Therefore you need to install the <code>coffee-script</code> package on the Node.js page locally, as described in Installing and Removing External Software Using Node Package Manager . <code>--inspect</code> Specify this parameter when you are using Node.js v7 for Chrome Debugging Protocol support. Otherwise, by default the debug process will use V8 Debugging Protocol .
Environment Variables	In this field, specify the environment variables for the NodeJS executable file, if applicable. Click the Browse button  to the right of the field and configure a list of variables in the Environment Variables dialog box, that opens: <ul style="list-style-type: none">- To define a new variable, click the Add toolbar button  and specify the variable name and value.- To discard a variable definition, select it in the list and click the Delete toolbar button .- Click OK, when ready The definitions of variables are displayed in the Environment variables read-only field with semicolons as separators. The acceptable variables are: <ul style="list-style-type: none">- <code>NODE_PATH</code> : A  -separated list of directories prefixed to the module search path.- <code>NODE_MODULE_CONTEXTS</code> : Set to 1 to load modules in their own global contexts.- <code>NODE_DISABLE_COLORS</code> : Set to 1 to disable colors in the REPL.

Toolbar

ItemShortcutDescription

	 <code>Alt+Insert</code>	Click this button to add a new configuration to the list.
	 <code>Alt+Delete</code>	Click this button to remove the selected configuration from the list.
	 <code>Ctrl+D</code>	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
	 <code>Alt+Up</code> or <code>Alt+Down</code>	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.

	<p>Move into new folder / Create new folder</p> <p>Use this button to create a new folder.</p> <p>If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created.</p> <p>Move run/debug configurations to a folder using drag-and-drop, or the  buttons.</p>
	<p>Sort configurations</p> <p>Click this button to sort configurations in alphabetical order.</p>

Common options

ItemDescription

Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	<p>Select this check box to make the run/debug configuration available to other team members.</p> <p>The shared run/debug configurations are kept in separate xml files under <code>.idea\runConfigurations</code> folder, while the local run/debug configurations are kept in the <code>.idea\workspace.xml</code>.</p> <p>This check box is not available when editing the run/debug configuration defaults.</p>
Single instance only	<p>If this check box is selected, this run/debug configuration cannot be launched more than once.</p> <p>Every time a new run/debug configuration is launched, PyCharm checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place.</p> <p>This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources.</p> <p>If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.</p>
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

ItemKeyboardDescription shortcut

		<p>Click this icon to add a task to the list. Select the task to be added:</p> <ul style="list-style-type: none"> – Run External tool. Select this option to run an application which is external to PyCharm. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in PyCharm yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools. – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run File Watchers. Select this option to have PyCharm apply all the currently active file watchers, see Using File Watchers for details. – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the <code>gruntfile.js</code> where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the <code>grunt-cli</code> package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the <code>gulpfile.js</code> where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the <code>gulp</code> package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the <code>package.json</code> file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: <ul style="list-style-type: none"> – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support. – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details. – Upload files to Remote Host. Select this option to have the application files automatically uploaded to the server according to the default server access configuration. For more information, see Configuring Synchronization with a Web Server and Uploading and Downloading Files.
		Click this icon to remove the selected task from the list.
		Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
		Click this icon to move the selected task one line up in the list.



Alt+Down

Click this icon to move the selected task one line down in the list.

Show this page

Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.

Active tool window

Select this option if you want the [Run/Debug](#) tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: PhoneGap/Cordova

This feature is supported in the Professional edition only.

Use this dialog box to create configurations for running and debugging applications developed through integration with the help of [PhoneGap](#), [Apache Cordova](#), and [Ionic](#) frameworks and intended for running on various mobile platforms, including [Android](#).

On this page:

- [Getting access to the Run/Debug Configuration: PhoneGap/Cordova dialog](#)
- [PhoneGap/Cordova/Ionic-specific configuration settings](#)
- [Toolbar](#)
- [Common options](#)

Getting access to the Run/Debug Configuration: PhoneGap/Cordova dialog

Make sure the **PhoneGap/Cordova** plugin is enabled. The plugin is not bundled with PyCharm, but it can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).

PhoneGap/Cordova/Ionic-specific configuration settings

ItemDescription

PhoneGap/Cordova Executable Path	In this field, specify the location of the executable file <code>phonegap.cmd</code> , <code>cordova.cmd</code> , or <code>ionic.cmd</code> (see Installing PhoneGap/Cordova/Ionic).
----------------------------------	---

PhoneGap/Cordova Working Directory	In this field, specify the folder under which the PhoneGap/Cordova/Ionic application files to run are stored.
------------------------------------	--

Command	From this drop-down list, choose the command to run. The contents of the drop-down list, depend on the actually used framework, namely, on the executable file specified in the PhoneGap/Cordova Executable Path field. The available options are:
---------	--

– For PhoneGap:

- emulate
- run
- prepare
- serve
- remote build
- remote run

See <https://www.npmjs.org/package/phonegap> for a list of PhoneGap-specific commands with descriptions.

– For Cordova:

- emulate
- run
- prepare
- serve

See <https://www.npmjs.org/package/cordova> for a list of Cordova-specific commands with descriptions.

– For Ionic:

- emulate
- run
- prepare
- serve

See <https://www.npmjs.org/package/ionic> for a list of Ionic-specific commands with descriptions.

Platform	From this drop-down list, choose the platform for running on which the application is intended. The available options are:
----------	--

- Android
- ios To emulate this platform, you need to install the [ios-sim command line tool](#) globally. You can do it through the Node Package Manager (npm), see [Installing and Removing External Software Using Node Package Manager](#) or by running the `sudo npm install ios-sim -g` command, depending on your operating system.
- amazon-fireos
- firefoxos
- blackberry10
- ubuntu
- wp8
- windows8
- browser

Learn more about targeted platforms at http://docs.phonegap.com/en/edge/guide_platforms_index.md.html#Platform%20Guides and http://cordova.apache.org/docs/en/4.0.0/guide_cli_index.md.html#The%20Command-Line%20Interface.

Specify Target	Select this check box to appoint an Android physical or virtual device to run the application on and select the required device from the drop-down list. The list shows all the virtual and physical devices that are currently configured on our machine. See http://docs.phonegap.com/en/edge/guide_platforms_android_index.md.html#Android%20Platform%20Guide for details. If PyCharm displays the following error message: <code>Cannot detect ios-sim in path</code> , make sure you have installed the <code>ios-sim</code> , see Before you start .
----------------	---

Toolbar

ItemShortcutDescription

+

Alt+Insert

Click this button to add a new configuration to the list.

–

Alt+Delete

Click this button to remove the selected configuration from the list.

		Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
	 or 	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the  buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

ItemDescription

Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. The shared run/debug configurations are kept in separate xml files under <code>.idea\runConfigurations</code> folder, while the local run/debug configurations are kept in the <code>.idea\workspace.xml</code> . This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, PyCharm checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

ItemKeyboardDescription shortcut

		Click this icon to add a task to the list. Select the task to be added: <ul style="list-style-type: none"> – Run External tool. Select this option to run an application which is external to PyCharm. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in PyCharm yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools. – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run File Watchers. Select this option to have PyCharm apply all the currently active file watchers, see Using File Watchers for details. – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the <code>gruntfile.js</code> where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the <code>grunt-cli</code> package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the <code>gulpfile.js</code> where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the <code>gulp</code> package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the <code>package.json</code> file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: <ul style="list-style-type: none"> – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support. – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details. – Upload files to Remote Host. Select this option to have the application files automatically uploaded to the server according to the default server access configuration. For more information, see Configuring Synchronization with a Web Server and Uploading and Downloading Files.
---	---	---

		Click this icon to remove the selected task from the list.
		Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
		Click this icon to move the selected task one line up in the list.
		Click this icon to move the selected task one line down in the list.
Show this page	<input type="checkbox"/>	Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window	<input type="checkbox"/>	Select this option if you want the Run/Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Protractor

This feature is supported in the Professional edition only.

In this dialog box, create configurations for running and debugging **AngularJS unit tests** using the **Protractor test runner**.

On this page:

- [Getting access to the Run/Debug Configuration: Protractor dialog](#)
- [Protractor-specific configuration settings](#)
- [Toolbar](#)
- [Common options](#)

Getting access to the Run/Debug Configuration: Protractor dialog

1. **Install** and **enable** the Node.js plugin. The plugin is not bundled with PyCharm, but it can be installed from the **JetBrains plugin repository** as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).
2. Download and install the [NodeJS](#) runtime environment that contains the [Node Package Manager\(npm\)](#).
3. Using the **Node Package Manager**, install the [Protractor test framework](#) as described in [Using AngularJS](#).
4. Make sure the **AngularJS** plugin is activated. The plugin is not bundled with PyCharm, but it can be installed from the **JetBrains plugin repository** as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).

Protractor-specific configuration settings

ItemDescription

Name	In this text box, specify the name of the run/debug configuration.
Share	Select this check box to make the run/debug configuration available to other team members. The shared run/debug configurations are kept in separate xml files under <code>.idea\runConfigurations</code> folder, while the local run/debug configurations are kept in the <code>.idea\workspace.xml</code> . This check box is not available when editing the run/debug configuration defaults.
Configuration file	In this field, specify the location of the Protractor configuration file . Normally, the file has the extensions <code>protractor.conf.js</code>
Node interpreter	In this field, specify the NodeJS installation home. Type the path to the NodeJS executable file manually, or click the  button and select the location in the dialog box, that opens. If you have appointed one of the installations as default , the field displays the path to its executable file.
Protractor package	In this field, specify the Protractor installation home <code>/npm/node_modules/protractor</code> . If you installed Protractor regularly through the Node Package Manager , PyCharm detects the Protractor installation home itself. Alternatively, type the path to executable file manually, or click the Browse button  and select the location in the dialog box, that opens.
Environment variables	In this field, specify the environment variables for the NodeJS executable file, if applicable. Click the Browse button  to the right of the field and configure a list of variables in the Environment Variables dialog box, that opens: <ul style="list-style-type: none">– To define a new variable, click the Add toolbar button  and specify the variable name and value.– To discard a variable definition, select it in the list and click the Delete toolbar button .– Click OK, when ready The definitions of variables are displayed in the Environment variables read-only field with semicolons as separators. The acceptable variables are: <ul style="list-style-type: none">– <code>NODE_PATH</code> : A <code>:</code>-separated list of directories prefixed to the module search path.– <code>NODE_MODULE_CONTEXTS</code> : Set to 1 to load modules in their own global contexts.– <code>NODE_DISABLE_COLORS</code> : Set to 1 to disable colors in the REPL.

Toolbar

ItemShortcutDescription

		Click this button to add a new configuration to the list.
		Click this button to remove the selected configuration from the list.
		Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
	 or 	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the  buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

ItemDescription

Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. The shared run/debug configurations are kept in separate xml files under <code>.idea\runConfigurations</code> folder, while the local run/debug configurations are kept in the <code>.idea\workspace.xml</code> . This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, PyCharm checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

ItemKeyboardDescription shortcut

		Click this icon to add a task to the list. Select the task to be added: <ul style="list-style-type: none">– Run External tool. Select this option to run an application which is external to PyCharm. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in PyCharm yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools.– Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project.– Run File Watchers. Select this option to have PyCharm apply all the currently active file watchers, see Using File Watchers for details.– Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the <code>gruntfile.js</code> where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the <code>grunt-cli</code> package.– Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the <code>gulpfile.js</code> where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the <code>gulp</code> package.– Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the <code>package.json</code> file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it.– Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:<ul style="list-style-type: none">– If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.– If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.– Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support.– Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.– Upload files to Remote Host. Select this option to have the application files automatically uploaded to the server according to the default server access configuration. For more information, see Configuring Synchronization with a Web Server and Uploading and Downloading Files.
		Click this icon to remove the selected task from the list.
		Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
		Click this icon to move the selected task one line up in the list.
		Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run/Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Python

On this page:

- [Prerequisites](#)
- [Configuration tab](#)
- [Logs tab](#)
- [Toolbar](#)
- [Common options](#)

Prerequisites

Before you start working with Python, make sure that Python plugin is [installed and enabled](#). The plugin is not bundled with PyCharm.

Also make sure that the following prerequisites are met:

- Python SDK is downloaded and installed on your machine.
- The required framework SDKs are downloaded and installed on your machine.

Refer to their respective download and installation pages for details:

- [Python](#)
- [Django](#)

Use this dialog box to create a run/debug configuration for Python scripts.

Configuration tab

ItemDescription

Script	In this text box, specify the name of the Python script to be executed.
Script parameters	<p>In this text box, specify parameters to be passed to the Python script. When specifying the script parameters, follow these rules:</p> <ul style="list-style-type: none">– Use spaces to separate individual script parameters.– Script parameters containing spaces should be delimited with double quotes, for example, <code>some "param"</code> or <code>"some param"</code>.– If script parameter includes double quotes, escape the double quotes with backslashes, for example: <pre>-s"main.snap_source_dirs=[\"pcomponents/src/main/python\"]" -s"http.cc_port=8189" -s"backdoor.port=9189" -s"main.metadata={\"location\": \"B\", \"language\": \"python\", \"platform\": \"unix\"}"</pre>
Project	Click this drop-down list to select one of the projects, opened in the same PyCharm window , where this run/debug configuration should be used. If there is only one open project, this field is not displayed.
Environment variable	<p>This field shows the list of environment variables. If the list contains several variables, they are delimited with semicolons.</p> <p>To fill in the list, click the browse button, or press <code>Shift+Enter</code> and specify the desired set of environment variables in the Environment Variables dialog box.</p> <p>To create a new variable, click <code>+</code>, and type the desired name and value.</p> <p>By default, the variable <code>PYTHONUNBUFFERED</code> is set to 1.</p>
Emulate terminal in output node	<p>On Linux and macOS systems, select this check box to emulate the terminal in the Run tool window.</p> <p>On Windows system, this option is not visible!</p>
Show command line afterwards	Select this check box to leave the console opened after a project run or a debug session, saving its context.
Python Interpreter	<p>Select one of the pre-configured Python interpreters from the drop-down list.</p> <p>Note that you can select a remote interpreter, as well as the local one. If a remote interpreter is selected, you have to specify path mappings in the corresponding field that appears.</p>
Interpreter options	In this field, specify the string to be passed to the interpreter. If necessary, click <code>+</code> , and type the string in the editor.
Working directory	<p>Specify a directory to be used by the running task.</p> <ul style="list-style-type: none">– When a default run/debug configuration is created by the keyboard shortcut <code>Ctrl+Shift+F10</code>, or by choosing Run on the context menu of a script, the working directory is the one that contains the executable script. This directory may differ from the project directory.– When this field is left blank, the <code>bin</code> directory of the PyCharm installation will be used.
Path mappings	<p>This field appears, if a remote interpreter has been selected in the field Python interpreter.</p> <p>Click the browse button <code>...</code> to define the required mappings between the local and remote paths. In the Edit Path Mappings dialog box, use <code>+</code>/<code>-</code> buttons to create new mappings, or delete the selected ones.</p>
Add content roots to PYTHONPATH	Select this check box to add all content roots of your project to the environment variable PYTHONPATH;
Add source roots	Select this check box to add all source roots of your project to the environment variable PYTHONPATH;

Warning! This field only appears when [Docker-based remote interpreter](#) has been selected for a project.

Note Speaking about the correspondence of settings with some options (`--net`, `--link`, etc.), note that these options come from [Docker command line arguments](#).

Click  to open the dialog and specify the following settings:

- **Disable networking:** select this check box to have the networking disabled. This corresponds to `--net="none"`, which means that inside a container the external network resources are not available.
- **Network mode:** corresponds to the other values of the option `--net`.
 - `bridge` is the default value. An IP address will be allocated for container on the bridge's network and traffic will be routed through this bridge to the container.

Containers can communicate via their IP addresses by default. To communicate by name, they must be linked.

- `host`: use the host's network stack inside the container.
- `container:<name|id>`: use the network stack of another container, specified via its `name` or `id`.

Refer to the [Network settings](#) documentation for details.

- **Links:** Use this section to link the container to be created with the other containers. This is applicable to `Network mode = bridge` and corresponds to the `--link` option.
- **Publish all ports:** If the check box is selected, all the container's exposed ports are bound to a random host port.

See e.g. [EXPOSE \(incoming ports\)](#) in [Docker run reference](#). This corresponds to the option `--publish-all`.

- **Port bindings:** Use this field to specify the Container port/protocol - Host IP address/port mappings.
E.g. `8080 tcp 127.0.0.1 18080` would bind the TCP port `8080` inside the container to port `18080` on the `localhost` or `127.0.0.1` interface on the host machine.

For more info, see e.g. [Connect using network port mapping](#) in [Legacy container links](#).

- **Extra hosts:** This corresponds to the `--add-host` option. Refer to the page [Managing/etc/hosts](#) for details.
- **Volume bindings:** Use this field to specify the bindings between the special folders-volumes and the folders of the computer, where the Docker daemon runs. This corresponds to the `-v` option.

See [Managing data in containers](#) for details.

- **Environment variables:** Use this field to specify the list of environment variables and their values. This corresponds to the `-e` option. Refer to the page [ENV \(environment variables\)](#) for details.

Click  to expand the tables. Click ,  or  to make up the lists.

Logs tab

Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the [Run](#) or [Debug tool window](#).

ItemDescription

Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window .
Log File Entry	The read-only fields in this column list the log files to show. The list can contain: <ul style="list-style-type: none"> – Full paths to specific files. – Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.
Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point to the desired location in the dialog that opens .
Show console when a message is printed to standard output stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.
Show console when a message is printed to standard error stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.
	Click this button to open the Edit Log Files Aliases dialog where you can select a new log entry and specify an alias for it.
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog .
	Click this button to remove the selected log entry from the list.
	Click this button to edit the select log file entry. The button is available only when an entry is selected.

Toolbar

ItemShortcutDescription

	 Alt+Insert	Click this button to add a new configuration to the list.
	 Alt+Delete	Click this button to remove the selected configuration from the list.
	 Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.



Alt+Up or
Alt+Down

Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug dropdown list on the main toolbar.



Move into new folder /
Create new folder

Use this button to [create a new folder](#).
If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created.

Move run/debug configurations to a folder using drag-and-drop, or the buttons.



Sort configurations

Click this button to sort configurations in alphabetical order.

Common options

ItemDescription

Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	<p>Select this check box to make the run/debug configuration available to other team members. The shared run/debug configurations are kept in separate xml files under <code>.idea\runConfigurations</code> folder, while the local run/debug configurations are kept in the <code>.idea\workspace.xml</code>.</p> <p>This check box is not available when editing the run/debug configuration defaults.</p>
Single instance only	<p>If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, PyCharm checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place.</p> <p>This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources.</p> <p>If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.</p>
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

ItemKeyboardDescription shortcut



Alt+Insert

- Click this icon to add a task to the list. Select the task to be added:
- Run External tool. Select this option to run an application which is external to PyCharm. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in PyCharm yet, add its definition. For more information, see [Configuring Third-Party Tools](#) and [External Tools](#).
 - Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project.
 - Run File Watchers. Select this option to have PyCharm apply all the currently active [file watchers](#), see [Using File Watchers](#) for details.
 - Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package.
 - Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package.
 - Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it.
 - Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:
 - If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.
 - If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.
 - Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see [CoffeeScript Support](#).
 - Run Remote External tool: Add a remote SSH external tool. Refer to the section [Remote SSH External Tools](#) for details.
 - Upload files to Remote Host. Select this option to have the application files automatically uploaded to the server according to the [default server access configuration](#). For more information, see [Configuring Synchronization with a Web Server](#) and [Uploading and Downloading Files](#).



Alt+Delete

Click this icon to remove the selected task from the list.



	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
 	Click this icon to move the selected task one line up in the list.
 	Click this icon to move the selected task one line down in the list.
Show this page	Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window	Select this option if you want the Run/Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Python Remote Debug

This feature is supported in the Professional edition only.

Use the remote debug configuration to launch the debug server. Refer to the [Remote Debugging](#) topic for additional information.

ItemDescription

Name	In this text box, specify the name of the current run/debug configuration.
Update your script	This section contains vital information required to prepare for remote debugging. Add <code>pycharm-debug.egg</code> from the <code>debug-eggs</code> directory under PyCharm installation to the Python path. Add the following import statement <pre>import pydevd</pre> Copy the import statement from this read-only field, and paste it in your local script: <pre>import pydevd</pre> Add the following command to connect to the debug server <pre>pydevd.settrace(<host name>, port=<port number>)</pre> When <code><host name></code> is taken from the Local host name field of this debug configuration. - <code><port number></code> is the number taken from the Port field of this debug configuration, or, if it has not been specified, some random number written to the console. Note that the parameters of this command depend on the settings specified in this page. The command with the default settings is: <pre>pydevd.pydevd.settrace('localhost', port=\$SERVER_PORT,stdoutToServer=True,stderrToServer=True)</pre> which corresponds to the host name 'localhost', port number 0, selected check boxes Redirect output to console and Suspend after connect.
Local host name	Specify the name of the local host, by which the IDE is accessible from the remote host. This host name will be automatically substituted to the command line. By default, <code>localhost</code> is used.
Port	Specify the port number, which will be automatically substituted to the command line. If the default port number (0) is used, then PyCharm substitutes an arbitrary number to the command line at each launch of this debug configuration; if you specify any other value, it will be used permanently.
Path mappings	Use this field to create mappings between the local and remote paths. Clicking the browse button  results in opening Edit Path Mappings dialog box, where you can add new path mappings, and delete the selected ones.
Redirect output to console	If this check box is selected, the output and error streams will be redirected to the PyCharm console, and the command line is complemented with the <code>stdoutToServer=True, stderrToServer=True</code>
Suspend after connect	If this check box is selected, the debugger will suspend immediately after connecting to the IDE, on the next line after the <code>settrace</code> call. If this check box is not selected, the debugger will only suspend upon hitting a breakpoint, or clicking  , and the command line is complemented with <code>suspend=False</code>

Toolbar

ItemShortcutDescription

	<code>Alt+Insert</code>	Click this button to add a new configuration to the list.
	<code>Alt+Delete</code>	Click this button to remove the selected configuration from the list.
	<code>Ctrl+D</code>	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
	<code>Alt+Up</code> or <code>Alt+Down</code>	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the  buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

ItemDescription

Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. The shared run/debug configurations are kept in separate xml files under <code>.idea\runConfigurations</code> folder, while the local run/debug configurations are kept in the <code>.idea\workspace.xml</code> . This check box is not available when editing the run/debug configuration defaults.

Single instance only	<p>If this check box is selected, this run/debug configuration cannot be launched more than once.</p> <p>Every time a new run/debug configuration is launched, PyCharm checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place.</p> <p>This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources.</p> <p>If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.</p>
----------------------	--

Before launch Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

**ItemKeyboardDescription
shortcut**

	 Click this icon to add a task to the list. Select the task to be added: <ul style="list-style-type: none"> – Run External tool. Select this option to run an application which is external to PyCharm. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in PyCharm yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools. – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run File Watchers. Select this option to have PyCharm apply all the currently active file watchers, see Using File Watchers for details. – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the <code>gruntfile.js</code> where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the <code>grunt-cli</code> package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the <code>gulpfile.js</code> where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the <code>gulp</code> package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the <code>package.json</code> file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: <ul style="list-style-type: none"> – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support. – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details. – Upload files to Remote Host. Select this option to have the application files automatically uploaded to the server according to the default server access configuration. For more information, see Configuring Synchronization with a Web Server and Uploading and Downloading Files.
---	---

	 Click this icon to remove the selected task from the list.
	 Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	 Click this icon to move the selected task one line up in the list.
	 Click this icon to move the selected task one line down in the list.
Show this page	Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window	Select this option if you want the Run/Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configurations: Python Docs

Use these pages to create run/debug configurations for the supported inline documentation frameworks.

Click [+](#) to select the stub run/debug configuration of one of the following types:

- [Run/Debug Configuration: DocUtil Task](#)
- [Run/Debug Configuration: Sphinx Task](#)

Run/Debug Configuration: DocUtil Task

Use this dialog box to create a run/debug configuration for a [DocUtils](#) task, which allows you to produce documentation in some reasonable format (for example, HTML), from a file in the [reStructuredText](#) format.

In this section:

- [Configuration tab](#)
- [Logs tab](#)
- [Toolbar](#)
- [Common options](#)

Configuration tab

ItemDescription

Command	Select the command that will be used to convert the specified *.rst file. The selected command stipulates the output format of the documentation.
Input	Specify the fully qualified path to the source *.rst file. Type the path manually, or click the browse button, and choose the desired *.rst file from the file chooser dialog. Alternatively, you can press <code>Shift+Enter</code> to open the file chooser dialog.
Output	Specify the fully qualified path to the generated file. Type the path manually, or click the browse button, and choose the desired file from the file chooser dialog. Alternatively, you can press <code>Shift+Enter</code> to open the file chooser dialog.
Option	In this text field, type the keys the script will be launched with.
Open output file in browser	If this check box is selected, PyCharm will automatically open the generated documentation in the default browser . Note that the check box is disabled, when it is irrelevant to the command selected in the Command drop-down list.
Environment variable	Click the browse button, or press <code>Shift+Enter</code> to specify the desired set of environment variables in the Environment Variables dialog box. To create a new variable, click <code>+</code> , and type the desired name and value.
Python Interpreter	Select one of the pre-configured Python interpreters from the drop-down list. Note that you can select a remote interpreter , as well as the local one. If a remote interpreter is selected, you have to specify path mappings in the corresponding field that appears .
Interpreter options	In this field, specify the string to be passed to the interpreter. If necessary, click <code>⌨</code> , and type the string in the editor.
Working directory	Specify a directory to be used by the running task. <ul style="list-style-type: none">- When a default run/debug configuration is created by the keyboard shortcut <code>Ctrl+Shift+F10</code>, or by choosing Run on the context menu of a script, the working directory is the one that contains the executable script. This directory may differ from the project directory.- When this field is left blank, the <code>bin</code> directory of the PyCharm installation will be used.
Path mappings	This field appears, if a remote interpreter has been selected in the field Python interpreter. Click the browse button <code>⌨</code> to define the required mappings between the local and remote paths. In the Edit Path Mappings dialog box, use <code>+</code> / <code>-</code> buttons to create new mappings, or delete the selected ones.
Add content roots to PYTHONPATH	Select this check box to add all content roots of your project to the environment variable PYTHONPATH;
Add source roots to PYTHONPATH	Select this check box to add all source roots of your project to the environment variable PYTHONPATH;
Docker container settings	<p>Warning! This field only appears when Docker-based remote interpreter has been selected for a project.</p> <p>Note Speaking about the correspondence of settings with some options (<code>--net</code>, <code>--link</code>, etc.), note that these options come from Docker command line arguments.</p>

Click `⌨` to open the dialog and specify the following settings:

- **Disable networking:** select this check box to have the networking disabled. This corresponds to `--net="none"`, which means that inside a container the external network resources are not available.
- **Network mode:** corresponds to the other values of the option `--net`.
 - `bridge` is the default value. An IP address will be allocated for container on the bridge's network and traffic will be routed through this bridge to the container.

Containers can communicate via their IP addresses by default. To communicate by name, they must be linked.

- `host`: use the host's network stack inside the container.
- `container:<name|id>`: use the network stack of another container, specified via its name or id.

Refer to the [Network settings](#) documentation for details.

- **Links:** Use this section to link the container to be created with the other containers. This is applicable to `Network mode = bridge` and corresponds to the `--link` option.
- **Publish all ports:** If the check box is selected, all the container's exposed ports are bound to a random host port.

See e.g. [EXPOSE \(incoming ports\)](#) in [Docker run reference](#). This corresponds to the option `--publish-all`.

- **Port bindings:** Use this field to specify the Container port/protocol - Host IP address/port mappings.
E.g. `8080 tcp 127.0.0.1 18080` would bind the TCP port `8080` inside the container to port `18080` on the `localhost` or `127.0.0.1` interface on the host machine.

For more info, see e.g. [Connect using network port mapping](#) in [Legacy container links](#).

- **Extra hosts:** This corresponds to the `--add-host` option. Refer to the page [Managing /etc/hosts](#) for details.
- **Volume bindings:** Use this field to specify the bindings between the special folders-volumes and the folders of the computer, where the Docker daemon runs. This corresponds to the `-v` option.

See [Managing data in containers](#) for details.

- Environment variables: Use this field to specify the list of environment variables and their values. This corresponds to the `-e` option. Refer to the page [ENV \(environment variables\)](#) for details.

Click  to expand the tables. Click ,  or  to make up the lists.

Logs tab

Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the [Run](#) or [Debug tool window](#).

ItemDescription

Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window .
Log File Entry	The read-only fields in this column list the log files to show. The list can contain: <ul style="list-style-type: none">– Full paths to specific files.– Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.
Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point to the desired location in the dialog that opens .
Show console when a message is printed to standard output stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.
Show console when a message is printed to standard error stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.
	Click this button to open the Edit Log Files Aliases dialog where you can select a new log entry and specify an alias for it.
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog .
	Click this button to remove the selected log entry from the list.
	Click this button to edit the select log file entry. The button is available only when an entry is selected.

Toolbar

ItemShortcutDescription

		Click this button to add a new configuration to the list.
		Click this button to remove the selected configuration from the list.
		Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
	 or 	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the  buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

ItemDescription

Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. The shared run/debug configurations are kept in separate xml files under <code>.idea\runConfigurations</code> folder, while the local run/debug configurations are kept in the <code>.idea\workspace.xml</code> . This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, PyCharm checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place.

This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources.

If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.

Before launch Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item
Keyboard shortcut
Description

		Click this icon to add a task to the list. Select the task to be added: <ul style="list-style-type: none">– Run External tool. Select this option to run an application which is external to PyCharm. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in PyCharm yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools.– Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project.– Run File Watchers. Select this option to have PyCharm apply all the currently active file watchers, see Using File Watchers for details.– Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the <code>Gruntfile.js</code> where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the <code>grunt-cli</code> package.– Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the <code>Gulpfile.js</code> where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the <code>gulp</code> package.– Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the <code>package.json</code> file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it.– Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:<ul style="list-style-type: none">– If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.– If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.– Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support.– Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.– Upload files to Remote Host. Select this option to have the application files automatically uploaded to the server according to the default server access configuration. For more information, see Configuring Synchronization with a Web Server and Uploading and Downloading Files.
		Click this icon to remove the selected task from the list.
		Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
		Click this icon to move the selected task one line up in the list.
		Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run/Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Sphinx Task

Use this dialog box to create a run/debug configuration for a [Sphinx](#) task, which allows you to produce documentation in some reasonable format (for example, HTML), from a file in the [reStructuredText](#) format.

In this section:

- [Configuration tab](#)
- [Logs tab](#)
- [Toolbar](#)
- [Common options](#)

Configuration tab

ItemDescription

Command	Select the command that will be used to convert the specified <code>*.rst</code> file. The selected command stipulates the output format of the documentation.
Input	Specify the fully qualified path to the source <code>*.rst</code> file. Type the path manually, or click the browse button, and choose the desired <code>*.rst</code> file from the file chooser dialog. Alternatively, you can press <code>Shift+Enter</code> to open the file chooser dialog.
Output	Specify the fully qualified path to the generated directory where the generated files will be placed. Type the path manually, or click the browse button, and choose the desired file from the file chooser dialog. Alternatively, you can press <code>Shift+Enter</code> to open the Select Path dialog.
Option	In this text field, type the keys the script will be launched with.
Environment variable	Click the browse button, or press <code>Shift+Enter</code> to specify the desired set of environment variables in the Environment Variables dialog box. To create a new variable, click <code>+</code> , and type the desired name and value.
Python Interpreter	Select one of the pre-configured Python interpreters from the drop-down list. Note that you can select a remote interpreter , as well as the local one. If a remote interpreter is selected, you have to specify path mappings in the corresponding field that appears .
Interpreter options	In this field, specify the string to be passed to the interpreter. If necessary, click <code>+</code> , and type the string in the editor.
Working directory	Specify a directory to be used by the running task. <ul style="list-style-type: none">- When a default run/debug configuration is created by the keyboard shortcut <code>Ctrl+Shift+F10</code>, or by choosing Run on the context menu of a script, the working directory is the one that contains the executable script. This directory may differ from the project directory.- When this field is left blank, the <code>bin</code> directory of the PyCharm installation will be used.
Path mappings	This field appears, if a remote interpreter has been selected in the field Python interpreter. Click the browse button <code>+</code> to define the required mappings between the local and remote paths. In the Edit Path Mappings dialog box, use <code>+</code> / <code>-</code> buttons to create new mappings, or delete the selected ones.
Add content roots to PYTHONPATH	Select this check box to add all content roots of your project to the environment variable PYTHONPATH;
Add source roots to PYTHONPATH	Select this check box to add all source roots of your project to the environment variable PYTHONPATH;
Docker container settings	<div style="background-color: #ffff00; padding: 5px;">Warning! This field only appears when Docker-based remote interpreter has been selected for a project.</div> <div style="background-color: #ffff00; padding: 5px;">Note Speaking about the correspondence of settings with some options (<code>--net</code>, <code>--link</code>, etc), note that these options come from Docker command line arguments.</div> <p>Click <code>+</code> to open the dialog and specify the following settings:</p> <ul style="list-style-type: none">- Disable networking: select this check box to have the networking disabled. This corresponds to <code>--net="none"</code>, which means that inside a container the external network resources are not available.- Network mode: corresponds to the other values of the option <code>--net</code>.<ul style="list-style-type: none">- <code>bridge</code> is the default value. An IP address will be allocated for container on the bridge's network and traffic will be routed though this bridge to the container. <p>Containers can communicate via their IP addresses by default. To communicate by name, they must be linked.</p> <ul style="list-style-type: none">- <code>host</code>: use the host's network stack inside the container.- <code>container:<name id></code>: use the network stack of another container, specified via its <code>name</code> or <code>id</code>. <p>Refer to the Network settings documentation for details.</p> <ul style="list-style-type: none">- Links: Use this section to link the container to be created with the other containers. This is applicable to <code>Network mode = bridge</code> and corresponds to the <code>--link</code> option.- Publish all ports: If the check box is selected, all the container's exposed ports are bound to a random host port. <p>See e.g. EXPOSE (incoming ports) in Docker run reference. This corresponds to the option <code>--publish-all</code>.</p> <ul style="list-style-type: none">- Port bindings: Use this field to specify the Container port/protocol - Host IP address/port mappings. E.g. <code>8080 tcp 127.0.0.1 18080</code> would bind the TCP port <code>8080</code> inside the container to port <code>18080</code> on the <code>localhost</code> or <code>127.0.0.1</code> interface on the host machine. <p>For more info, see e.g. Connect using network port mapping in Legacy container links.</p> <ul style="list-style-type: none">- Extra hosts: This corresponds to the <code>--add-host</code> option. Refer to the page Managing /etc/hosts for details.- Volume bindings: Use this field to specify the bindings between the special folders-volumes and the folders of the computer, where the Docker daemon runs. This corresponds to the <code>-v</code> option. <p>See Managing data in containers for details.</p>

- Environment variables: Use this field to specify the list of environment variables and their values. This corresponds to the `-e` option. Refer to the page [ENV \(environment variables\)](#) for details.

Click  to expand the tables. Click ,  or  to make up the lists.

Logs tab

Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the [Run](#) or [Debug tool window](#).

ItemDescription

Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window .
Log File Entry	The read-only fields in this column list the log files to show. The list can contain: <ul style="list-style-type: none"> – Full paths to specific files. – Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.
Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point to the desired location in the dialog that opens .
Show console when a message is printed to standard output stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.
Show console when a message is printed to standard error stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.
	Click this button to open the Edit Log Files Aliases dialog where you can select a new log entry and specify an alias for it.
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog .
	Click this button to remove the selected log entry from the list.
	Click this button to edit the select log file entry. The button is available only when an entry is selected.

Toolbar

ItemShortcutDescription

		Click this button to add a new configuration to the list.
		Click this button to remove the selected configuration from the list.
		Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
		Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the  buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

ItemDescription

Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. The shared run/debug configurations are kept in separate xml files under <code>.idea\runConfigurations</code> folder, while the local run/debug configurations are kept in the <code>.idea\workspace.xml</code> . This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, PyCharm checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes

too much of the CPU and memory resources.

If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.

Before launch Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
------	-------------------	-------------



Alt+Insert

Click this icon to add a task to the list. Select the task to be added:

- Run External tool. Select this option to run an application which is external to PyCharm. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in PyCharm yet, add its definition. For more information, see [Configuring Third-Party Tools](#) and [External Tools](#).
- Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run.
This option is available only if you have already at least one run/debug configuration in the current project.
- Run File Watchers. Select this option to have PyCharm apply all the currently active file watchers, see [Using File Watchers](#) for details.
- Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool.
Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package.
- Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool.
Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package.
- Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with.
Specify the location of the Node.js interpreter and the parameters to pass to it.
- Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:
 - If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.
 - If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.
- Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see [CoffeeScript Support](#).
- Run Remote External tool: Add a remote SSH external tool. Refer to the section [Remote SSH External Tools](#) for details.
- Upload files to Remote Host. Select this option to have the application files automatically uploaded to the server according to the default [server access configuration](#). For more information, see [Configuring Synchronization with a Web Server](#) and [Uploading and Downloading Files](#).



Alt+Delete

Click this icon to remove the selected task from the list.



Enter

Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.



Alt+Up

Click this icon to move the selected task one line up in the list.



Alt+Down

Click this icon to move the selected task one line down in the list.

Show this page

Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.

Active tool window

Select this option if you want the [Run/Debug](#) tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Pyramid Server

This feature is supported in the Professional edition only.

Use this dialog box to create a run/debug configuration for launching a Pyramid server.

This section provides descriptions of the [configuration-specific items](#), as well as the [toolbar](#) and [options](#) that are common for all run/debug configurations.

In this section:

- [Configuration tab](#)
- [Logs tab](#)
- [Toolbar](#)
- [Common options](#)

Configuration tab

ItemDescription

Config file	In this field, specify the name of the configuration file <code>development.ini</code> .
Additional options	In this field, type the additional options to be passed to the server. These are the options that <code>pserve</code> accepts. Use <code>pserve --help</code> to learn more about the additional options.
Run browser	Select this check box, if you want your Pyramid application to open in the default browser. In the text field below, enter the IP address where your application will be opened.
Project	Click this drop-down list to select one of the projects, opened in the same PyCharm window , where this run/debug configuration should be used. If there is only one open project, this field is not displayed.
Environment variable	This field shows the list of environment variables. If the list contains several variables, they are delimited with semicolons. To fill in the list, click the browse button, or press <code>Shift+Enter</code> and specify the desired set of environment variables in the Environment Variables dialog box. To create a new variable, click <code>+</code> , and type the desired name and value. By default, the variable <code>PYTHONUNBUFFERED</code> is set to 1.
Python Interpreter	Select one of the pre-configured Python interpreters from the drop-down list. Note that you can select a remote interpreter , as well as the local one. If a remote interpreter is selected, you have to specify path mappings in the corresponding field that appears .
Interpreter options	In this field, specify the string to be passed to the interpreter. If necessary, click <code>+</code> , and type the string in the editor.
Working directory	Specify a directory to be used by the running task. <ul style="list-style-type: none">– When a default run/debug configuration is created by the keyboard shortcut <code>Ctrl+Shift+F10</code>, or by choosing Run on the context menu of a script, the working directory is the one that contains the executable script. This directory may differ from the project directory.– When this field is left blank, the <code>bin</code> directory of the PyCharm installation will be used.
Path mappings	This field appears, if a remote interpreter has been selected in the field Python interpreter. Click the browse button <code>...</code> to define the required mappings between the local and remote paths. In the Edit Path Mappings dialog box, use <code>+</code> / <code>-</code> buttons to create new mappings, or delete the selected ones.
Add content roots to PYTHONPATH	Select this check box to add all content roots of your project to the environment variable PYTHONPATH;
Add source roots to PYTHONPATH	Select this check box to add all source roots of your project to the environment variable PYTHONPATH;

Logs tab

Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the [Run](#) or [Debug tool window](#).

ItemDescription

Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window .
Log File Entry	The read-only fields in this column list the log files to show. The list can contain: <ul style="list-style-type: none">– Full paths to specific files.– Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.
Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point to the desired location in the dialog that opens .
Show console when a message is printed to	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.

standard output stream

Show console when a message is printed to standard error stream Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.

+ Click this button to open the [Edit Log Files Aliases dialog](#) where you can select a new log entry and specify an alias for it.

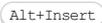
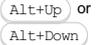
✎ Click this button to edit the properties of the selected log file entry in the [Edit Log Files Aliases dialog](#).

- Click this button to remove the selected log entry from the list.

⋮ Click this button to edit the selected log file entry. The button is available only when an entry is selected.

Toolbar

ItemShortcutDescription

		Click this button to add a new configuration to the list.
		Click this button to remove the selected configuration from the list.
		Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
		Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the  buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

ItemDescription

Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. The shared run/debug configurations are kept in separate xml files under <code>.idea\runConfigurations</code> folder, while the local run/debug configurations are kept in the <code>.idea\workspace.xml</code> . This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, PyCharm checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

ItemKeyboardDescription shortcut

		Click this icon to add a task to the list. Select the task to be added: <ul style="list-style-type: none">Run External tool. Select this option to run an application which is external to PyCharm. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in PyCharm yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools.Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project.Run File Watchers. Select this option to have PyCharm apply all the currently active file watchers, see Using File Watchers for details.Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the <code>gruntfile.js</code> where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the <code>grunt-cli</code> package.Run Gulp task. Select this option to run a Gulp task. In the Gulp task dialog box that opens, specify the <code>gulpfile.js</code> where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the <code>gulp</code> package.Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens,
---	---	--

specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with.

Specify the location of the Node.js interpreter and the parameters to pass to it.

- Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:
 - If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.
 - If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.
- Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see [CoffeeScript Support](#).
- Run Remote External tool: Add a remote SSH external tool. Refer to the section [Remote SSH External Tools](#) for details.
- Upload files to Remote Host. Select this option to have the application files automatically uploaded to the server according to the default [server access configuration](#). For more information, see [Configuring Synchronization with a Web Server](#) and [Uploading and Downloading Files](#).

		Click this icon to remove the selected task from the list.
		Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
		Click this icon to move the selected task one line up in the list.
		Click this icon to move the selected task one line down in the list.
Show this page	<input type="checkbox"/>	Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window	<input type="checkbox"/>	Select this option if you want the Run/Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Python Tests

This feature is supported in the Professional edition only.

Use these pages to create run/debug configurations for the supported testing frameworks.

Click [+](#) to select the stub run/debug configuration of one of the following types:

- [Run/Debug Configuration: Unittests](#)
- [Run/Debug Configuration: py.test](#)
- [Run/Debug Configuration: Nostests](#)
- [Run/Debug Configuration: Doctests](#)

Use this dialog box to create a run/debug configuration for [Python unit tests](#).

In this section:

- [Configuration tab](#)
- [Logs tab](#)
- [Toolbar](#)
- [Common options](#)

Configuration tab

ItemDescription

Unittests

Target	Click one of the radio-buttons to choose the possible target. The following choices are available: Python, Path, and Custom. The contents of the subsequent sections depend on the choice.	
AvailableItemDescription for		
Python		In this text field, specify the path to test(s). For example, run everything from the module <code>my_tests</code> , included in the package <code>tests</code> : <code>tests.my_tests</code> .
Python	Additional arguments	In this text field, specify the additional framework-specific arguments to be passed to the test as-is, for example <code>--some-argument=some-value</code> . Warning! This option is not recommended and should only be used by the experts, who want to launch something not supported by PyCharm.
Path		In this text field, specify the path to a file or folder to test everything in. For example, <code>C:/SampleProjects/py/MyPythonApp/Solver.py</code> . Note that when this target is selected, the browse button  appears to the right, allowing you to choose the path from the file system.
Path	Pattern	In this text field, specify the pattern that describes all the test in the required location.
Path	Additional arguments	In this text field, specify the additional framework-specific arguments to be passed to the test as-is, for example <code>--some-argument=some-value</code> . Warning! This option is not recommended and should only be used by the experts, who want to launch something not supported by PyCharm.
Custom	Additional arguments	In this text field, specify the additional framework-specific arguments to be passed to the test as-is, for example <code>--some-argument=some-value</code> . Warning! This option is not recommended and should only be used by the experts, who want to launch something not supported by PyCharm.
Environment variable	Click the browse button, or press <code>Shift+Enter</code> to specify the desired set of environment variables in the Environment Variables dialog box. To create a new variable, click  , and type the desired name and value.	
Python Interpreter	Select one of the pre-configured Python interpreters from the drop-down list. Note that you can select a remote interpreter , as well as the local one. If a remote interpreter is selected, you have to specify path mappings in the corresponding field that appears .	
Interpreter options	In this field, specify the string to be passed to the interpreter. If necessary, click  , and type the string in the editor.	
Working directory	Specify a directory to be used by the running task. – When a default run/debug configuration is created by the keyboard shortcut <code>Ctrl+Shift+F10</code> , or by choosing Run on the context menu of a script, the working directory is the one that contains the executable script. This directory may differ from the project directory. – When this field is left blank, the <code>bin</code> directory of the PyCharm installation will be used.	
Path mappings	This field appears, if a remote interpreter has been selected in the field Python interpreter. Click the browse button  to define the required mappings between the local and remote paths. In the Edit Path Mappings dialog box, use  buttons to create new mappings, or delete the selected ones.	
Add content roots to PYTHONPATH	Select this check box to add all content roots of your project to the environment variable PYTHONPATH;	
Add source roots	Select this check box to add all source roots of your project to the environment variable PYTHONPATH;	

Warning! This field only appears when [Docker-based remote interpreter](#) has been selected for a project.

Note Speaking about the correspondence of settings with some options (`--net`, `--link`, etc.), note that these options come from [Docker command line arguments](#).

Click  to open the dialog and specify the following settings:

- Disable networking: select this check box to have the networking disabled. This corresponds to `--net="none"`, which means that inside a container the external network resources are not available.
- Network mode: corresponds to the other values of the option `--net`.
 - `bridge` is the default value. An IP address will be allocated for container on the bridge's network and traffic will be routed though this bridge to the container.

Containers can communicate via their IP addresses by default. To communicate by name, they must be linked.

- `host`: use the host's network stack inside the container.
- `container:<name|id>`: use the network stack of another container, specified via its name or id.

Refer to the [Network settings](#) documentation for details.

- Links: Use this section to link the container to be created with the other containers. This is applicable to `Network mode = bridge` and corresponds to the `--link` option.
- Publish all ports: If the check box is selected, all the container's exposed ports are bound to a random host port.

See e.g. [EXPOSE \(incoming ports\)](#) in [Docker run reference](#). This corresponds to the option `--publish-all`.

- Port bindings: Use this field to specify the Container port/protocol - Host IP address/port mappings. E.g. `8080 tcp 127.0.0.1 18080` would bind the TCP port `8080` inside the container to port `18080` on the `localhost` or `127.0.0.1` interface on the host machine.

For more info, see e.g. [Connect using network port mapping](#) in [Legacy container links](#).

- Extra hosts: This corresponds to the `--add-host` option. Refer to the page [Managing /etc/hosts](#) for details.
- Volume bindings: Use this field to specify the bindings between the special folders-volumes and the folders of the computer, where the Docker daemon runs. This corresponds to the `-v` option.

See [Managing data in containers](#) for details.

- Environment variables: Use this field to specify the list of environment variables and their values. This corresponds to the `-e` option. Refer to the page [ENV \(environment variables\)](#) for details.

Click  to expand the tables. Click ,  or  to make up the lists.

Logs tab

Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the [Run](#) or [Debug tool window](#).

ItemDescription

Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window .
Log File Entry	The read-only fields in this column list the log files to show. The list can contain: <ul style="list-style-type: none"> - Full paths to specific files. - Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.
Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point to the desired location in the dialog that opens .
Show console when a message is printed to standard output stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.
Show console when a message is printed to standard error stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.
	Click this button to open the Edit Log Files Aliases dialog where you can select a new log entry and specify an alias for it.
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog .
	Click this button to remove the selected log entry from the list.
	Click this button to edit the select log file entry. The button is available only when an entry is selected.

Toolbar

ItemShortcutDescription

	 Alt+Insert	Click this button to add a new configuration to the list.
	 Alt+Delete	Click this button to remove the selected configuration from the list.
	 Ctrl+D	Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.



Alt+Up or
Alt+Down

Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug dropdown list on the main toolbar.



Move into new folder /
Create new folder

Use this button to [create a new folder](#).
If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created.

Move run/debug configurations to a folder using drag-and-drop, or the buttons.



Sort configurations

Click this button to sort configurations in alphabetical order.

Common options

ItemDescription

Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	<p>Select this check box to make the run/debug configuration available to other team members. The shared run/debug configurations are kept in separate xml files under <code>.idea\runConfigurations</code> folder, while the local run/debug configurations are kept in the <code>.idea\workspace.xml</code>.</p> <p>This check box is not available when editing the run/debug configuration defaults.</p>
Single instance only	<p>If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, PyCharm checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place.</p> <p>This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources.</p> <p>If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.</p>
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

ItemKeyboardDescription shortcut



Alt+Insert

Click this icon to add a task to the list. Select the task to be added:

- Run External tool. Select this option to run an application which is external to PyCharm. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in PyCharm yet, add its definition. For more information, see [Configuring Third-Party Tools](#) and [External Tools](#).
- Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project.
- Run File Watchers. Select this option to have PyCharm apply all the currently active [file watchers](#), see [Using File Watchers](#) for details.
- Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package.
- Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package.
- Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it.
- Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:
 - If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.
 - If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.
- Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see [CoffeeScript Support](#).
- Run Remote External tool: Add a remote SSH external tool. Refer to the section [Remote SSH External Tools](#) for details.
- Upload files to Remote Host. Select this option to have the application files automatically uploaded to the server according to the [default server access configuration](#). For more information, see [Configuring Synchronization with a Web Server](#) and [Uploading and Downloading Files](#).



Alt+Delete

Click this icon to remove the selected task from the list.



	Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
 	Click this icon to move the selected task one line up in the list.
 	Click this icon to move the selected task one line down in the list.
Show this page	Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window	Select this option if you want the Run/Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Warning! The plugin `coverage` for `py.test`, due to technical restrictions, breaks PyCharm's debugger.

Use this dialog box to create a run/debug configuration for `py.test`.

In this section:

- [Configuration tab](#)
- [Logs tab](#)
- [Toolbar](#)
- [Common options](#)

Configuration tab

ItemDescription

py.test		
Target		Click one of the radio-buttons to choose the possible target. The following choices are available: Python, Path, and Custom. The contents of the subsequent sections depend on the choice.
	AvailableItemDescription for	
Python		In this text field, specify the path to test(s). For example, run everything from the module <code>my_tests</code> , included in the package <code>tests</code> : <code>tests.my_tests</code> .
Python	Additional arguments	In this text field, specify the additional framework-specific arguments to be passed to the test as-is, for example <code>--some-argument=some-value</code> . Warning! This option is not recommended and should only be used by the experts, who want to launch something not supported by PyCharm.
Python	Keywords	In this text field, specify the keywords to search for the required tests. Refer to the test name search documentation for details.
Path		In this text field, specify the path to a file or folder to test everything in. For example, <code>C:/SampleProjects/py/MyPythonApp/Solver.py</code> . Note that when this target is selected, the browse button  appears to the right, allowing you to choose the path from the file system.
Path	Additional arguments	In this text field, specify the additional framework-specific arguments to be passed to the test as-is, for example <code>--some-argument=some-value</code> . Warning! This option is not recommended and should only be used by the experts, who want to launch something not supported by PyCharm.
Custom	Additional arguments	In this text field, specify the additional framework-specific arguments to be passed to the test as-is, for example <code>--some-argument=some-value</code> . Warning! This option is not recommended and should only be used by the experts, who want to launch something not supported by PyCharm.
Environment variable		Click the browse button, or press <code>Shift+Enter</code> to specify the desired set of environment variables in the Environment Variables dialog box. To create a new variable, click  , and type the desired name and value.
Python Interpreter		Select one of the pre-configured Python interpreters from the drop-down list. Note that you can select a remote interpreter , as well as the <code>local</code> one. If a remote interpreter is selected, you have to specify path mappings in the corresponding field that appears .
Interpreter options		In this field, specify the string to be passed to the interpreter. If necessary, click  , and type the string in the editor.
Working directory		Specify a directory to be used by the running task. – When a default run/debug configuration is created by the keyboard shortcut <code>Ctrl+Shift+F10</code> , or by choosing Run on the context menu of a script, the working directory is the one that contains the executable script. This directory may differ from the project directory. – When this field is left blank, the <code>bin</code> directory of the PyCharm installation will be used.
Path mappings		This field appears, if a remote interpreter has been selected in the field Python interpreter. Click the browse button  to define the required mappings between the local and remote paths. In the Edit Path Mappings dialog box, use  buttons to create new mappings, or delete the selected ones.
Add content		Select this check box to add all content roots of your project to the environment variable PYTHONPATH;

roots to
PYTHONPATH

Add source roots to PYTHONPATH Select this check box to add all [source roots](#) of your project to the environment variable PYTHONPATH;

Docker container settings

Warning! This field only appears when [Docker-based remote interpreter](#) has been selected for a project.

Note Speaking about the correspondence of settings with some options (`--net`, `--link`, etc.), note that these options come from [Docker command line arguments](#).

Click  to open the dialog and specify the following settings:

- **Disable networking:** select this check box to have the networking disabled. This corresponds to `--net="none"`, which means that inside a container the external network resources are not available.
- **Network mode:** corresponds to the other values of the option `--net`.
 - `bridge` is the default value. An IP address will be allocated for container on the bridge's network and traffic will be routed through this bridge to the container.

Containers can communicate via their IP addresses by default. To communicate by name, they must be linked.

- `host`: use the host's network stack inside the container.
- `container:<name|id>`: use the network stack of another container, specified via its [name](#) or [id](#).

Refer to the [Network settings](#) documentation for details.

- **Links:** Use this section to link the container to be created with the other containers. This is applicable to `Network mode = bridge` and corresponds to the `--link` option.
- **Publish all ports:** If the check box is selected, all the container's exposed ports are bound to a random host port.

See e.g. [EXPOSE \(incoming ports\)](#) in [Docker run reference](#). This corresponds to the option `--publish-all`.

- **Port bindings:** Use this field to specify the Container port/protocol - Host IP address/port mappings.
E.g. `8080 tcp 127.0.0.1 18080` would bind the TCP port `8080` inside the container to port `18080` on the `localhost` or `127.0.0.1` interface on the host machine.

For more info, see e.g. [Connect using network port mapping](#) in [Legacy container links](#).

- **Extra hosts:** This corresponds to the `--add-host` option. Refer to the page [Managing /etc/hosts](#) for details.
- **Volume bindings:** Use this field to specify the bindings between the special folders-volumes and the folders of the computer, where the Docker daemon runs. This corresponds to the `-v` option.

See [Managing data in containers](#) for details.

- **Environment variables:** Use this field to specify the list of environment variables and their values. This corresponds to the `-e` option. Refer to the page [ENV \(environment variables\)](#) for details.

Click  to expand the tables. Click ,  or  to make up the lists.

Logs tab

Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the [Run](#) or [Debug tool window](#).

ItemDescription

Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window .
Log File Entry	The read-only fields in this column list the log files to show. The list can contain: <ul style="list-style-type: none">- Full paths to specific files.- Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.
Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point to the desired location in the dialog that opens .
Show console when a message is printed to standard output stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.
Show console when a message is printed to standard error stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.
	Click this button to open the Edit Log Files Aliases dialog where you can select a new log entry and specify an alias for it.
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog .
	Click this button to remove the selected log entry from the list.
	Click this button to edit the select log file entry. The button is available only when an entry is selected.

Toolbar

ItemShortcutDescription

		Click this button to add a new configuration to the list.
		Click this button to remove the selected configuration from the list.

		Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
	 or 	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the  buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

ItemDescription

Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. The shared run/debug configurations are kept in separate xml files under <code>.idea\runConfigurations</code> folder, while the local run/debug configurations are kept in the <code>.idea\workspace.xml</code> . This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, PyCharm checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

ItemKeyboardDescription shortcut

		Click this icon to add a task to the list. Select the task to be added: <ul style="list-style-type: none"> – Run External tool. Select this option to run an application which is external to PyCharm. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in PyCharm yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools. – Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project. – Run File Watchers. Select this option to have PyCharm apply all the currently active file watchers, see Using File Watchers for details. – Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the <code>gruntfile.js</code> where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the <code>grunt-cli</code> package. – Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the <code>gulpfile.js</code> where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the <code>gulp</code> package. – Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the <code>package.json</code> file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it. – Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected: <ul style="list-style-type: none"> – If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start. – If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched. – Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support. – Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details. – Upload files to Remote Host. Select this option to have the application files automatically uploaded to the server according to the default server access configuration. For more information, see Configuring Synchronization with a Web Server and Uploading and Downloading Files.
---	---	---

	 Click this icon to remove the selected task from the list.
	 Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
	 Click this icon to move the selected task one line up in the list.
	 Click this icon to move the selected task one line down in the list.
Show this page	Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window	Select this option if you want the Run/Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Use this dialog box to create a run/debug configuration for [Nose tests](#).

In this section:

- [Configuration tab](#)
- [Logs tab](#)
- [Toolbar](#)
- [Common options](#)

Configuration tab

ItemDescription

Nostests

Target	Click one of the radio-buttons to choose the possible target. The following choices are available: Python, Path, and Custom. The contents of the subsequent sections depend on the choice.
	<p>AvailableItemDescription for</p> <hr/> <p>Python</p> <p>In this text field, specify the path to test(s). For example, run everything from the module <code>my_tests</code>, included in the package <code>tests</code>:</p> <pre>tests.my_tests .</pre> <hr/> <p>Python</p> <p>Additional In this text field, specify the additional framework-specific arguments to be passed to the test as-is, for example arguments <code>--some-argument=some-value</code>.</p> <p>Warning! This option is not recommended and should only be used by the experts, who want to launch something not supported by PyCharm.</p> <hr/> <p>Path</p> <p>In this text field, specify the path to a file or folder to test everything in. For example,</p> <pre>C:/SampleProjects/py/MyPythonApp/Solver.py .</pre> <p>Note that when this target is selected, the browse button  appears to the right, allowing you to choose the path from the file system.</p> <hr/> <p>Path</p> <p>Regex Pattern In this text field, specify the regex pattern that describes all the test in the required location. Files, directories, function names, and class names that match the specified regex pattern, are considered tests, for example <code>test.*</code></p> <hr/> <p>Path</p> <p>Additional In this text field, specify the additional framework-specific arguments to be passed to the test as-is, for example arguments <code>--some-argument=some-value</code>.</p> <p>Warning! This option is not recommended and should only be used by the experts, who want to launch something not supported by PyCharm.</p> <hr/> <p>Custom</p> <p>Additional In this text field, specify the additional framework-specific arguments to be passed to the test as-is, for example arguments <code>--some-argument=some-value</code>.</p> <p>Warning! This option is not recommended and should only be used by the experts, who want to launch something not supported by PyCharm.</p>
Environment variable	Click the browse button, or press <code>Shift+Enter</code> to specify the desired set of environment variables in the Environment Variables dialog box. To create a new variable, click + , and type the desired name and value.
Python Interpreter	Select one of the pre-configured Python interpreters from the drop-down list.
	Note that you can select a remote interpreter , as well as the local one. If a remote interpreter is selected, you have to specify path mappings in the corresponding field that appears .
Interpreter options	In this field, specify the string to be passed to the interpreter. If necessary, click  , and type the string in the editor.
Working directory	Specify a directory to be used by the running task. <ul style="list-style-type: none"> – When a default run/debug configuration is created by the keyboard shortcut <code>Ctrl+Shift+F10</code>, or by choosing Run on the context menu of a script, the working directory is the one that contains the executable script. This directory may differ from the project directory. – When this field is left blank, the <code>bin</code> directory of the PyCharm installation will be used.
Path mappings	This field appears, if a remote interpreter has been selected in the field Python interpreter.
	Click the browse button  to define the required mappings between the local and remote paths. In the Edit Path Mappings dialog box, use + / - buttons to create new mappings, or delete the selected ones.
Add content roots to PYTHONPATH	Select this check box to add all content roots of your project to the environment variable PYTHONPATH;
Add source roots to PYTHONPATH	Select this check box to add all source roots of your project to the environment variable PYTHONPATH;

Warning! This field only appears when [Docker-based remote interpreter](#) has been selected for a project.

Note Speaking about the correspondence of settings with some options (`--net`, `--link`, etc.), note that these options come from [Docker command line arguments](#).

Click  to open the dialog and specify the following settings:

- **Disable networking:** select this check box to have the networking disabled. This corresponds to `--net="none"`, which means that inside a container the external network resources are not available.
- **Network mode:** corresponds to the other values of the option `--net`.
 - `bridge` is the default value. An IP address will be allocated for container on the bridge's network and traffic will be routed through this bridge to the container.

Containers can communicate via their IP addresses by default. To communicate by name, they must be linked.

- `host`: use the host's network stack inside the container.
- `container:<name|id>`: use the network stack of another container, specified via its name or id.

Refer to the [Network settings](#) documentation for details.

- **Links:** Use this section to link the container to be created with the other containers. This is applicable to `Network mode = bridge` and corresponds to the `--link` option.
- **Publish all ports:** If the check box is selected, all the container's exposed ports are bound to a random host port.

See e.g. [EXPOSE \(incoming ports\)](#) in [Docker run reference](#). This corresponds to the option `--publish-all`.

- **Port bindings:** Use this field to specify the Container port/protocol - Host IP address/port mappings. E.g. `8080 tcp 127.0.0.1 18080` would bind the TCP port `8080` inside the container to port `18080` on the `localhost` or `127.0.0.1` interface on the host machine.

For more info, see e.g. [Connect using network port mapping](#) in [Legacy container links](#).

- **Extra hosts:** This corresponds to the `--add-host` option. Refer to the page [Managing /etc/hosts](#) for details.
- **Volume bindings:** Use this field to specify the bindings between the special folders-volumes and the folders of the computer, where the Docker daemon runs. This corresponds to the `-v` option.

See [Managing data in containers](#) for details.

- **Environment variables:** Use this field to specify the list of environment variables and their values. This corresponds to the `-e` option. Refer to the page [ENV \(environment variables\)](#) for details.

Click  to expand the tables. Click ,  or  to make up the lists.

Logs tab

Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the [Run](#) or [Debug tool window](#).

ItemDescription

Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window .
Log File Entry	The read-only fields in this column list the log files to show. The list can contain: <ul style="list-style-type: none"> - Full paths to specific files. - Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.
Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point to the desired location in the dialog that opens .
Show console when a message is printed to standard output stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.
Show console when a message is printed to standard error stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.
	Click this button to open the Edit Log Files Aliases dialog where you can select a new log entry and specify an alias for it.
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog .
	Click this button to remove the selected log entry from the list.
	Click this button to edit the select log file entry. The button is available only when an entry is selected.

Toolbar

ItemShortcutDescription

		Click this button to add a new configuration to the list.
		Click this button to remove the selected configuration from the list.
		Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
	 or 	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-

down list on the main toolbar.

	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the  buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

ItemDescription

Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. The shared run/debug configurations are kept in separate xml files under <code>.idea\runConfigurations</code> folder, while the local run/debug configurations are kept in the <code>.idea\workspace.xml</code> . This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, PyCharm checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place. This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources. If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.
Before launch	Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

ItemKeyboardDescription shortcut

		Click this icon to add a task to the list. Select the task to be added: <ul style="list-style-type: none">– Run External tool. Select this option to run an application which is external to PyCharm. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in PyCharm yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools.– Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project.– Run File Watchers. Select this option to have PyCharm apply all the currently active file watchers, see Using File Watchers for details.– Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the <code>gruntfile.js</code> where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the <code>grunt-cli</code> package.– Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the <code>gulpfile.js</code> where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the <code>gulp</code> package.– Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the <code>package.json</code> file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it.– Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:<ul style="list-style-type: none">– If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.– If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.– Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support.– Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.– Upload files to Remote Host. Select this option to have the application files automatically uploaded to the server according to the default server access configuration. For more information, see Configuring Synchronization with a Web Server and Uploading and Downloading Files.
		Click this icon to remove the selected task from the list.
		Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
		Click this icon to move the selected task one line up in the list.



Alt+Up
Alt+Down

Click this icon to move the selected task one line down in the list.

Show this page

Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.

Active tool window

Select this option if you want the [Run/Debug](#) tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Use this dialog box to create a run/debug configuration for [Doctests](#).

In this section:

- [Configuration tab](#)
- [Logs tab](#)
- [Toolbar](#)
- [Common options](#)

Configuration tab

ItemDescription

Tests	<p>Click one of the radio-buttons to define the testing scope (all tests in a folder, all tests in a script, a test class, a single test method or function.) Next, specify the location of the tests. The fields in this section are enabled depending on the test type selected in the Tests section.</p> <ul style="list-style-type: none"> – All tests in folder: for this testing scope, specify the following arguments: <ul style="list-style-type: none"> Folder: type the path to the directory that contains the tests to be executed, or click the browse button and select the desired directory in the Select Path dialog box. Pattern: type the one or more patterns the file names should match for the files to be considered tests. Use comma as the delimiter. The patterns should be Python regular expressions. – Tests in script: for this scope, specify the name of the script that contains the tests to be executed. – Tests class: for this scope, specify the name of the script and test class. – Tests method: for this scope, specify the name of the script, test class and test method. – Tests function: for this scope, specify the name of the script, and test method.
Environment variable	<p>Click the browse button, or press <code>Shift+Enter</code> to specify the desired set of environment variables in the Environment Variables dialog box. To create a new variable, click <code>+</code>, and type the desired name and value.</p>
Python Interpreter	<p>Select one of the pre-configured Python interpreters from the drop-down list.</p> <p>Note that you can select a remote interpreter, as well as the local one. If a remote interpreter is selected, you have to specify path mappings in the corresponding field that appears.</p>
Interpreter options	<p>In this field, specify the string to be passed to the interpreter. If necessary, click <code>⌨</code>, and type the string in the editor.</p>
Working directory	<p>Specify a directory to be used by the running task.</p> <ul style="list-style-type: none"> – When a default run/debug configuration is created by the keyboard shortcut <code>Ctrl+Shift+F10</code>, or by choosing Run on the context menu of a script, the working directory is the one that contains the executable script. This directory may differ from the project directory. – When this field is left blank, the <code>bin</code> directory of the PyCharm installation will be used.
Path mappings	<p>This field appears, if a remote interpreter has been selected in the field Python interpreter.</p> <p>Click the browse button <code>⌨</code> to define the required mappings between the local and remote paths. In the Edit Path Mappings dialog box, use <code>+</code>/<code>-</code> buttons to create new mappings, or delete the selected ones.</p>
Add content roots to PYTHONPATH	<p>Select this check box to add all content roots of your project to the environment variable PYTHONPATH;</p>
Add source roots to PYTHONPATH	<p>Select this check box to add all source roots of your project to the environment variable PYTHONPATH;</p>
Docker container settings	<p>Warning! This field only appears when Docker-based remote interpreter has been selected for a project.</p> <p>Note Speaking about the correspondence of settings with some options (<code>--net</code>, <code>--link</code>, etc.), note that these options come from Docker command line arguments.</p> <p>Click <code>⌨</code> to open the dialog and specify the following settings:</p> <ul style="list-style-type: none"> – Disable networking: select this check box to have the networking disabled. This corresponds to <code>--net="none"</code>, which means that inside a container the external network resources are not available. – Network mode: corresponds to the other values of the option <code>--net</code>. <ul style="list-style-type: none"> – <code>bridge</code> is the default value. An IP address will be allocated for container on the bridge's network and traffic will be routed though this bridge to the container. <p>Containers can communicate via their IP addresses by default. To communicate by name, they must be linked.</p> <ul style="list-style-type: none"> – <code>host</code>: use the host's network stack inside the container. – <code>container:<name id></code>: use the network stack of another container, specified via its name or id. <p>Refer to the Network settings documentation for details.</p> <ul style="list-style-type: none"> – Links: Use this section to link the container to be created with the other containers. This is applicable to <code>Network mode = bridge</code> and corresponds to the <code>--link</code> option. – Publish all ports: If the check box is selected, all the container's exposed ports are bound to a random host port. <p>See e.g. EXPOSE (incoming ports) in Docker run reference. This corresponds to the option <code>--publish-all</code>.</p> <ul style="list-style-type: none"> – Port bindings: Use this field to specify the Container port/protocol - Host IP address/port mappings. E.g. <code>8080 tcp 127.0.0.1 18080</code> would bind the TCP port <code>8080</code> inside the container to port <code>18080</code> on the <code>localhost</code> or <code>127.0.0.1</code> interface on the host machine. <p>For more info, see e.g. Connect using network port mapping in Legacy container links.</p> <ul style="list-style-type: none"> – Extra hosts: This corresponds to the <code>--add-host</code> option. Refer to the page Managing /etc/hosts for details. – Volume bindings: Use this field to specify the bindings between the special folders-volumes and the folders of the computer, where the Docker

daemon runs. This corresponds to the `-v` option.

See [Managing data in containers](#) for details.

- Environment variables: Use this field to specify the list of environment variables and their values. This corresponds to the `-e` option. Refer to the page [ENV \(environment variables\)](#) for details.

Click  to expand the tables. Click ,  or  to make up the lists.

Logs tab

Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the [Run](#) or [Debug tool window](#).

ItemDescription

Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window .
Log File Entry	The read-only fields in this column list the log files to show. The list can contain: <ul style="list-style-type: none">– Full paths to specific files.– Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.
Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point to the desired location in the dialog that opens .
Show console when a message is printed to standard output stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.
Show console when a message is printed to standard error stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.
	Click this button to open the Edit Log Files Aliases dialog where you can select a new log entry and specify an alias for it.
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog .
	Click this button to remove the selected log entry from the list.
	Click this button to edit the select log file entry. The button is available only when an entry is selected.

Toolbar

ItemShortcutDescription

		Click this button to add a new configuration to the list.
		Click this button to remove the selected configuration from the list.
		Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
	 or 	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the  buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

ItemDescription

Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. The shared run/debug configurations are kept in separate xml files under <code>.idea\runConfigurations</code> folder, while the local run/debug configurations are kept in the <code>.idea\workspace.xml</code> . This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, PyCharm checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place.

This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources.

If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.

Before launch Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
------	-------------------	-------------



Alt+Insert

Click this icon to add a task to the list. Select the task to be added:

- Run External tool. Select this option to run an application which is external to PyCharm. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in PyCharm yet, add its definition. For more information, see [Configuring Third-Party Tools](#) and [External Tools](#).
- Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project.
- Run File Watchers. Select this option to have PyCharm apply all the currently active [file watchers](#), see [Using File Watchers](#) for details.
- Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the `gruntfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `grunt-cli` package.
- Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the `gulpfile.js` where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the `gulp` package.
- Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the `package.json` file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it.
- Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:
 - If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.
 - If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.
- Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see [CoffeeScript Support](#).
- Run Remote External tool: Add a remote SSH external tool. Refer to the section [Remote SSH External Tools](#) for details.
- Upload files to Remote Host. Select this option to have the application files automatically uploaded to the server according to the [default server access configuration](#). For more information, see [Configuring Synchronization with a Web Server](#) and [Uploading and Downloading Files](#).



Alt+Delete

Click this icon to remove the selected task from the list.



Enter

Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.



Alt+Up

Click this icon to move the selected task one line up in the list.



Alt+Down

Click this icon to move the selected task one line down in the list.

Show this page

Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.

Active tool window

Select this option if you want the [Run/Debug](#) tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Run/Debug Configuration: Tox

The **Tox** run/debug configuration enables you running test with different Python versions and interpreters.

The dialog box consists of the following tabs:

- [Configuration tab](#)
- [Logs tab](#)

Click [here](#) for the description of the options that are common for all run/debug configurations.

Configuration tab

ItemDescription

Item	Description
Tox	
Arguments	Specify the arguments that are passed to the <code>tox.ini</code> script. So doing, the arguments are delimited with spaces, for example, <code>--some-arg --foo-arg</code> .
Run only environment	Specify here the Python environments/interpreters, where your project will be executed. So doing, the environments are delimited with commas, for example, <code>py27,py34</code> .
Environment	
Project	Click this drop-down list to select one of the projects, opened in the same PyCharm window , where this run/debug configuration should be used. If there is only one open project, this field is not displayed.
Environment variable	<p>This field shows the list of environment variables. If the list contains several variables, they are delimited with semicolons.</p> <p>To fill in the list, click the browse button, or press <code>Shift+Enter</code> and specify the desired set of environment variables in the Environment Variables dialog box.</p> <p>To create a new variable, click <code>+</code>, and type the desired name and value.</p>
Python Interpreter	<p>Select one of the pre-configured Python interpreters from the drop-down list.</p> <p>Note that you can select a remote interpreter, as well as the local one. If a remote interpreter is selected, you have to specify path mappings in the corresponding field that appears.</p>
Interpreter options	In this field, specify the string to be passed to the interpreter. If necessary, click <code>+</code> , and type the string in the editor.
Working directory	<p>Specify a directory to be used by the running task.</p> <ul style="list-style-type: none">– When a default run/debug configuration is created by the keyboard shortcut <code>Ctrl+Shift+F10</code>, or by choosing Run on the context menu of a script, the working directory is the one that contains the executable script. This directory may differ from the project directory.– When this field is left blank, the <code>bin</code> directory of the PyCharm installation will be used.
Path mappings	<p>This field appears, if a remote interpreter has been selected in the field Python interpreter.</p> <p>Click the browse button <code>+</code> to define the required mappings between the local and remote paths. In the Edit Path Mappings dialog box, use <code>+</code>/<code>-</code> buttons to create new mappings, or delete the selected ones.</p>
Add content roots to PYTHONPATH	Select this check box to add all content roots of your project to the environment variable PYTHONPATH;
Add source roots to PYTHONPATH	Select this check box to add all source roots of your project to the environment variable PYTHONPATH;
Docker container settings	<p>Warning! This field only appears when Docker-based remote interpreter has been selected for a project.</p> <p>Note Speaking about the correspondence of settings with some options (<code>--net</code>, <code>--link</code>, etc.), note that these options come from Docker command line arguments.</p> <p>Click <code>+</code> to open the dialog and specify the following settings:</p> <ul style="list-style-type: none">– Disable networking: select this check box to have the networking disabled. This corresponds to <code>--net="none"</code>, which means that inside a container the external network resources are not available.– Network mode: corresponds to the other values of the option <code>--net</code>.<ul style="list-style-type: none">– <code>bridge</code> is the default value. An IP address will be allocated for container on the bridge's network and traffic will be routed though this bridge to the container. <p>Containers can communicate via their IP addresses by default. To communicate by name, they must be linked.</p> <ul style="list-style-type: none">– <code>host</code>: use the host's network stack inside the container.– <code>container:<name id></code>: use the network stack of another container, specified via its name or id. <p>Refer to the Network settings documentation for details.</p> <ul style="list-style-type: none">– Links: Use this section to link the container to be created with the other containers. This is applicable to <code>Network mode = bridge</code> and corresponds to the <code>--link</code> option.– Publish all ports: if the check box is selected, all the container's exposed ports are bound to a random host port. <p>See e.g. EXPOSE (incoming ports) in Docker run reference. This corresponds to the option <code>--publish-all</code>.</p> <ul style="list-style-type: none">– Port bindings: Use this field to specify the Container port/protocol - Host IP address/port mappings. E.g. <code>8080 tcp 127.0.0.1 18080</code> would bind the TCP port <code>8080</code> inside the container to port <code>18080</code> on the <code>localhost</code> or <code>127.0.0.1</code> interface on the host machine. <p>For more info, see e.g. Connect using network port mapping in Legacy container links.</p> <ul style="list-style-type: none">– Extra hosts: This corresponds to the <code>--add-host</code> option. Refer to the page Managing /etc/hosts for details.– Volume bindings: Use this field to specify the bindings between the special folders-volumes and the folders of the computer, where the Docker

daemon runs. This corresponds to the `-v` option.

See [Managing data in containers](#) for details.

- Environment variables: Use this field to specify the list of environment variables and their values. This corresponds to the `-e` option. Refer to the page [ENV \(environment variables\)](#) for details.

Click  to expand the tables. Click ,  or  to make up the lists.

Logs tab

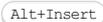
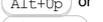
Use this tab to specify which log files generated while running or debugging should be displayed in the console, that is, on the dedicated tabs of the [Run](#) or [Debug tool window](#).

ItemDescription

Is Active	Select check boxes in this column to have the log entries displayed in the corresponding tabs in the Run tool window or Debug tool window .
Log File Entry	The read-only fields in this column list the log files to show. The list can contain: <ul style="list-style-type: none">– Full paths to specific files.– Aliases to substitute for full paths or patterns. These aliases are also displayed in the headers of the tabs where the corresponding log files are shown. If a log entry pattern defines more than one file, the tab header shows the name of the file instead of the log entry alias.
Skip Content	Select this check box to have the previous content of the selected log skipped.
Save console output to file	Select this check box to save the console output to the specified location. Type the path manually, or click the browse button and point to the desired location in the dialog that opens .
Show console when a message is printed to standard output stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.out.
Show console when a message is printed to standard error stream	Select this check box to activate the output console and bring it forward if an associated process writes to Standard.err.
	Click this button to open the Edit Log Files Aliases dialog where you can select a new log entry and specify an alias for it.
	Click this button to edit the properties of the selected log file entry in the Edit Log Files Aliases dialog .
	Click this button to remove the selected log entry from the list.
	Click this button to edit the select log file entry. The button is available only when an entry is selected.

Toolbar

ItemShortcutDescription

		Click this button to add a new configuration to the list.
		Click this button to remove the selected configuration from the list.
		Click this button to create a copy of the selected configuration.
	Edit defaults	Click this button to edit the default configuration templates. The defaults are used for newly created configurations.
	 or 	Use these buttons to move the selected configuration or folder up and down in the list. The order of configurations or folders in the list defines the order in which configurations appear in the Run/Debug drop-down list on the main toolbar.
	Move into new folder / Create new folder	Use this button to create a new folder . If one or more run/debug configurations are in focus, the selected run/debug configurations are automatically moved to the newly created folder. If only a category is in focus, an empty folder is created. Move run/debug configurations to a folder using drag-and-drop, or the  buttons.
	Sort configurations	Click this button to sort configurations in alphabetical order.

Common options

ItemDescription

Name	In this text box, specify the name of the current run/debug configuration. This field does not appear for the default run/debug configurations.
Defaults	This node in the left-hand pane of the dialog box contains the default run/debug configuration settings. Select the desired configuration to change its default settings in the right-hand pane. The defaults are applied to all newly created run/debug configurations.
Share	Select this check box to make the run/debug configuration available to other team members. The shared run/debug configurations are kept in separate xml files under <code>.idea\runConfigurations</code> folder, while the local run/debug configurations are kept in the <code>.idea\workspace.xml</code> . This check box is not available when editing the run/debug configuration defaults.
Single instance only	If this check box is selected, this run/debug configuration cannot be launched more than once. Every time a new run/debug configuration is launched, PyCharm checks the presence of the other instances of the same run/debug configuration, and displays a confirmation dialog box. If you click OK in the confirmation dialog box, the first instance of the runner will be stopped, and the next one will take its place.

This makes sense when the usage of certain resources can cause conflicts, or when launching two run/debug configurations of the same type consumes too much of the CPU and memory resources.

If this check box is not selected, it is possible to launch as many instances of the runner as required. So doing, each runner will start in its own tab of the Run tool window.

Before launch Specify which tasks must be performed before applying the run/debug configuration. The specified tasks are performed in the order they appear in the list.

Item	Keyboard shortcut	Description
------	-------------------	-------------

		Click this icon to add a task to the list. Select the task to be added: <ul style="list-style-type: none">– Run External tool. Select this option to run an application which is external to PyCharm. In the dialog that opens, select the application or applications that should be run. If the necessary application is not defined in PyCharm yet, add its definition. For more information, see Configuring Third-Party Tools and External Tools.– Run Another Configuration. Select this option to have another run/debug configuration executed. In the dialog that opens, select the configuration to run. This option is available only if you have already at least one run/debug configuration in the current project.– Run File Watchers. Select this option to have PyCharm apply all the currently active file watchers, see Using File Watchers for details.– Run Grunt task. Select this option to run a Grunt task. In the Grunt task dialog box that opens, specify the <code>gruntfile.js</code> where the required task is defined, select the task to execute, and specify the arguments to pass to the Grunt tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the <code>grunt-cli</code> package.– Run Gulp task. Select this option to run a Grunt task. In the Gulp task dialog box that opens, specify the <code>gulpfile.js</code> where the required task is defined, select the task to execute, and specify the arguments to pass to the Gulp tool. Specify the location of the Node.js interpreter, the parameters to pass to it, and the path to the <code>gulp</code> package.– Run npm Script. Select this check box to execute an npm script. In the NPM Script dialog box that opens, specify the <code>package.json</code> file where the required script is defined, select the script to execute, choose the command to apply to it, and specify the arguments to execute the script with. Specify the location of the Node.js interpreter and the parameters to pass to it.– Compile TypeScript. Select this option to run the built-in TypeScript compiler and thus make sure that all the changes you made to your TypeScript code are reflected in the generated JavaScript files. In the TypeScript Compile Settings dialog that opens, select or clear the Check errors check box to configure the behaviour of the compiler in case any errors are detected:<ul style="list-style-type: none">– If the Check errors check box is selected, the compiler will show all the errors and the run configuration will not start.– If the Check errors check box is cleared, the compiler will show all the detected errors but the run configuration still will be launched.– Generate CoffeeScript Source Maps. Select this option to have the source maps for your CoffeeScript sources generated. In the dialog that opens, specify where your CoffeeScript source files are located. For more information, see CoffeeScript Support.– Run Remote External tool: Add a remote SSH external tool. Refer to the section Remote SSH External Tools for details.– Upload files to Remote Host. Select this option to have the application files automatically uploaded to the server according to the default server access configuration. For more information, see Configuring Synchronization with a Web Server and Uploading and Downloading Files.
---	---	---

		Click this icon to remove the selected task from the list.
		Click this icon to edit the selected task. Make the necessary changes in the dialog that opens.
		Click this icon to move the selected task one line up in the list.
		Click this icon to move the selected task one line down in the list.
Show this page		Select this check box to have the run/debug configuration settings shown prior to actually starting the run/debug configuration.
Active tool window		Select this option if you want the Run/Debug tool windows to be activated automatically when you run/debug your application. This option is enabled by default.

Save File as Template Dialog

PyCharm helps you create the [file templates](#) from the existing files.

ItemDescription

Name	Specify here the name under which the new template will appear in the Files tab of the File and Code Templates settings. By default, the name of the current file is used.
Extension	Specify here the extension of the file to be created by this new template. By default, the extension of the current file is used.
Template text	<p>Edit the template contents. You can use:</p> <ul style="list-style-type: none">– Plain text.– <code>#parse</code> directives to work with template includes.– Custom variables. Their names can be defined right in the template through the <code>#set</code> directive or will be defined during the file creation.– Variables to be expanded into corresponding values in the <code>\${<variable_name>}</code> format. <p>The available predefined file template variables are:</p> <ul style="list-style-type: none">– <code>\${PROJECT_NAME}</code> - the name of the current project.– <code>\${NAME}</code> - the name of the new file which you specify in the New File dialog box during the file creation.– <code>\${USER}</code> - the login name of the current user.– <code>\${DATE}</code> - the current system date.– <code>\${TIME}</code> - the current system time.– <code>\${YEAR}</code> - the current year.– <code>\${MONTH}</code> - the current month.– <code>\${DAY}</code> - the current day of the month.– <code>\${HOUR}</code> - the current hour.– <code>\${MINUTE}</code> - the current minute.– <code>\${PRODUCT_NAME}</code> - the name of the IDE in which the file will be created.– <code>\${MONTH_NAME_SHORT}</code> - the first 3 letters of the month name. Example: Jan, Feb, etc.– <code>\${MONTH_NAME_FULL}</code> - full name of a month. Example: January, February, etc. <p>Treating dollar sign</p> <ul style="list-style-type: none">– You can prevent treating dollar characters (<code>\$</code>) in template variables as prefixes. If you need a dollar character (<code>\$</code>) inserted as is, use the <code>#{DS}</code> file template variable instead. When the template is applied, this variable evaluates to a plain dollar character (<code>\$</code>). <p>Examples:</p> <ul style="list-style-type: none">– To use some version control keywords (such as <code>\$Revision\$</code> , <code>\$Date\$</code> , etc.) in your default class template, write <code>#{DS}</code> instead of the dollar prefix (<code>\$</code>).– The template code <code>#{DS}this</code> will be rendered as <code>\$this</code> . <div style="background-color: #ffff00; padding: 5px; margin-top: 10px;">Note PyCharm doesn't prompt for the values of Velocity variables defined with <code>#set</code> .</div>
Reformat according to style	Select this check box, to have PyCharm reformat generated stub files according to the style defined on the Code Style page . This option is only available in the Files tab.
Enable Live Templates	Select this check box to use a live template inside a file template. So doing, one has to put the live template fragments into Velocity escape syntax. For example: <pre>#[[\$MY_VARIABLE\$ \$END\$]]</pre> Thus, one can specify the cursor position. Note that it is required to use the live template variables here!
Description	This read-only field provides information about the template, its predefined variables, and the way they work.

Select Path Dialog

Use this dialog to select the necessary files or folders.

The dialog name and the available functions depend on the task you are performing at the moment (inappropriate functions are normally disabled). For example, depending on the situation, you may be able to select only one item, or a number of items. There may be cases when you can select a folder or folders but cannot select a file or files, etc.

- [Main functions](#)
- [Path field](#)
- [Using drag-and-drop from a file browser](#)

Main functions

Most of the functions available in this dialog are accessed by means of the toolbar icons (shown in the **Icon** column). Alternatively, you can use context menu commands (accessed by right-clicking items in the tree; listed in the **Command** column) or keyboard shortcuts (the **Shortcut** column).

IconCommandShortcutDescription

Icon	Command	Shortcut	Description
	Home	Ctrl+I	Use this icon, command or shortcut to select your home directory. For example, on Windows, this may be <code>C:\Users\<your_name>< code="">.</your_name><></code>
	Desktop	Ctrl+D	Use this icon, command or shortcut to select the Desktop directory
	Project	Ctrl+2	Use this icon, command or shortcut to select your project root directory.
	New Folder	Alt+Insert	Use this icon, command or shortcut to create a new folder in the selected folder.
	Delete	Delete	Use this icon, command or shortcut to delete the selected file or folder.
	Refresh	Ctrl+Alt+Y	Use this icon, command or shortcut to synchronize the tree with the current state of the file system. (Under certain circumstances, PyCharm may not be aware of the changes made externally unless you use this command.)
	Show or Hide Hidden Files and Folders		Use this icon or command to turn showing hidden files and folders on or off.
	Hide or Show path	Ctrl+P	Use this command or shortcut to hide or show the path field . (The command is located on the toolbar in the right-hand part of the dialog and is shown as a hyperlink.)

Path field

The path field (if not hidden) is located underneath the toolbar. This field shows the path to the item selected in the tree.

By using the path auto-completion feature available in this field, you can quickly navigate through the file system to select the necessary file or folder.

To activate path auto-completion, place the cursor in the field and press `Ctrl+Space`. Start typing. A pop-up will appear showing the contents of the current directory. Select an item in the pop-up. Continue typing and selecting until the necessary item is selected.

Use the  button to the right to show the history list of recent entries.

Using drag-and-drop from a file browser

You can quickly locate and select the necessary file or folder if you drag the corresponding item from your file browser (Explorer, Finder, etc.) into the area where the tree is shown.

Specify Code Cleanup Scope Dialog

Code | Code Cleanup

Specify the scope for code cleanup. (Code cleanup means finding potentially problematic code fragments and automatically fixing them right away.)

ItemDescription

Whole project	Select this option if you want to perform code cleanup for the whole project.
Uncommitted files	This option is only available for projects under version control. Select this option if you only want to perform code cleanup for the files that have not yet been committed to a version control system, and choose a changelist from the drop-down list.
File <file path>	Select this option to perform code cleanup for the file that is open in the editor or selected in the Project tool window.
Module <module name>	Select this option to perform code cleanup for the project that is currently selected in the Project tool window. This option is only available, when several projects are opened in one PyCharm window .
Directory <directory path>	Select this option to perform code cleanup for the directory currently selected in the Project tool window.
Selected files	Select this option to perform code cleanup for the files selected in the Project tool window.
Custom scope	Select this option to specify a custom scope. Select one of the predefined scopes from the drop-down list, or click  and define the scope in the Scopes dialog that opens. For instructions on how to define a scope, refer to Scope Language Syntax Reference .
Inspection profile	Select the inspection profile to be used. Choose a pre-defined profile from the drop-down list, or click  and configure a profile in the Inspections dialog that opens. You can open the Inspections dialog to check which fixes will be applied to the selected scope when you run code cleanup.

Specify Code Duplication Analysis Scope

Code | Locate Duplicates

Use this dialog box to launch the search for duplicated code fragments in the specified scope.

ItemDescription

Whole project	Select this option to perform the analysis for the whole project.
File <name>	Select this option to analyze the file that is currently selected in the Project tool window or opened in the editor.
Selected files	Select this option to analyze the files that are currently selected in the Project tool window.
Uncommitted files	This scope is only available for the projects under version control. Select this option to have PyCharm analyze only files that have not been committed to the version control system. Use the drop-down list to further limit the analysis scope. The available options are: – All - select this option to have files from all changelists analyzed. – Default - select this option to have PyCharm analyze only files from the Default changelist.
Custom scope	Select this option to use a custom scope. Select a pre-defined scope from the drop-down list, or click  and define the scope in the Scopes dialog .

Tip Use a special [language](#) to define a scope.

Code Duplication Analysis Settings

Code | Locate Duplicates - OK

Use this dialog to define the sensitivity of search, and set limitation that will help you avoid reporting about every similar code construct. Your preferences are specified in a language-specific context.

ItemDescription

CSS - Do not show duplicates containing less than <number> CSS properties : Set the size of duplicated language constructs that are shown in the results window.

CoffeeScript - Anonymize Variables: when this check box is selected, two identical functions that use different variable names are considered duplicates, for example:

```
var test01 = function(a,b){
  return (a*b)
}

var test01 = function(a,b){
  return (a*b)
}
```

- Anonymize Functions
- Anonymize Literals

- Do not show duplicates simpler than: Set the size of duplicated language constructs that are shown in the result window. By default, the constructs less than 10 units are not included (and this limitation cannot be changed).
- Anonymize uncommon subexpressions simpler than: Set the value of the subelements within language constructs that can be considered similar, to show the construct as duplicate in the result window. The larger is the number, the larger constructs are taken as similar by PyCharm. The values are set as arbitrary weights based on the element size calculated with additive algorithm. The larger is the element, the higher is the calculated value.

ECMA Script level 4 - Anonymize Variables: when this check box is selected, two identical functions that use different variable names are considered duplicates, for example:

```
var test01 = function(a,b){
  return (a*b)
}

var test01 = function(a,b){
  return (a*b)
}
```

- Anonymize Functions
- Anonymize Literals

- Do not show duplicates simpler than: Set the size of duplicated language constructs that are shown in the result window. By default, the constructs less than 10 units are not included (and this limitation cannot be changed).
- Anonymize uncommon subexpressions simpler than: Set the value of the subelements within language constructs that can be considered similar, to show the construct as duplicate in the result window. The larger is the number, the larger constructs are taken as similar by PyCharm. The values are set as arbitrary weights based on the element size calculated with additive algorithm. The larger is the element, the higher is the calculated value.

HTML - Do not show duplicates simpler than: Set the size of duplicated language constructs that are shown in the result window. By default, the constructs less than 10 units are not included (and this limitation cannot be changed).

JavaScript - Anonymize Variables: when this check box is selected, two identical functions that use different variable names are considered duplicates, for example:

```
var test01 = function(a,b){
  return (a*b)
}

var test01 = function(a,b){
  return (a*b)
}
```

- Anonymize Functions
- Anonymize Literals

- Do not show duplicates simpler than: Set the size of duplicated language constructs that are shown in the result window. By default, the constructs less than 10 units are not included (and this limitation cannot be changed).
- Anonymize uncommon subexpressions simpler than: Set the value of the subelements within language constructs that can be considered similar, to show the construct as duplicate in the result window. The larger is the number, the larger constructs are taken as similar by PyCharm. The values are set as arbitrary weights based on the element size calculated with additive algorithm. The larger is the element, the higher is the calculated value.

JSON - Anonymize Variables: when this check box is selected, two identical functions that use different variable names are considered duplicates, for example:

```
var test01 = function(a,b){
return (a*b)
}

var test01 = function(a,b){
return (a*b)
}
```

- Anonymize Functions
- Anonymize Literals

- Do not show duplicates simpler than: Set the size of duplicated language constructs that are shown in the result window. By default, the constructs less than 10 units are not included (and this limitation cannot be changed).

- Anonymize uncommon subexpressions simpler than: Set the value of the subelements within language constructs that can be considered similar, to show the construct as duplicate in the result window. The larger is the number, the larger constructs are taken as similar by PyCharm.

Python

On this page, configure your preferences of search in Python constructs. The values are set as arbitrary weights based on the element size calculated with additive algorithm. The larger is the element, the higher is the calculated value.

- Selecting each of the check boxes defines which tokens should be anonymized.

- Do not show duplicates containing less than <number> tokens : Set the size of duplicated language constructs that are shown in the results window.

TypeScript

- Anonymize Variables: when this check box is selected, two identical functions that use different variable names are considered duplicates, for example:

```
var test01 = function(a,b){
return (a*b)
}

var test01 = function(a,b){
return (a*b)
}
```

- Anonymize Functions
- Anonymize Literals

- Do not show duplicates simpler than: Set the size of duplicated language constructs that are shown in the result window. By default, the constructs less than 10 units are not included (and this limitation cannot be changed).

- Anonymize uncommon subexpressions simpler than: Set the value of the subelements within language constructs that can be considered similar, to show the construct as duplicate in the result window. The larger is the number, the larger constructs are taken as similar by PyCharm.

The values are set as arbitrary weights based on the element size calculated with additive algorithm. The larger is the element, the higher is the calculated value.

XHTML

- Do not show duplicates containing less than <number> tags : Set the size of duplicated language constructs that are shown in the results window.

- Anonymize values of tags and attributes

XML

- Do not show duplicates containing less than <number> tags : Set the size of duplicated language constructs that are shown in the results window.

- Anonymize values of tags and attributes

Specify Inspection Scope Dialog

Code | Inspect Code

Use this dialog box to define the scope to apply inspection to and the profile against which the source code should be inspected.

Note that the list of scopes varies depending on the project type.

ItemDescription

Whole project	Select this option to perform the analysis for the whole project.
File <name>	Select this option to analyze the file that is currently selected in the Project tool window or opened in the editor.
Selected files	Select this option to analyze the files that are currently selected in the Project tool window.
Uncommitted files	<p>This scope is only available for the projects under version control.</p> <p>Select this option to have PyCharm analyze only files that have not been committed to the version control system. Use the drop-down list to further limit the analysis scope. The available options are:</p> <ul style="list-style-type: none">– All - select this option to have files from all changelists analyzed.– Default - select this option to have PyCharm analyze only files from the Default changelist.
Custom scope	<p>Select this option to use a custom scope. Select a pre-defined scope from the drop-down list, or click  and define the scope in the Scopes dialog.</p> <p>Tip Use a special language to define a scope.</p>
Inspection profile	<p>Select a profile to inspect the specified scope against.</p> <p>A profile is selected from the drop-down list. If the desired profile is not in the list, click the ellipsis button and configure the desired profile on the Inspections page of the Settings dialog.</p>

Structural Search and Replace Dialogs

Edit | Find | Search Structurally

Edit | Find | Replace Structurally

Use these dialog boxes to find and replace fragments of code that structurally match the suggested [search template](#).

Tip To learn more about the possible usages, refer to the section [Structural Search and Replace](#).

Item	Description	Available in
Search template	Use this text area to specify the template to be sought for. You can type the template code in the field or click the Copy existing template button to use one of the existing templates.	Both
Replacement template	Use this text area to specify the template to be substituted. You can type the template code in the field or click the Copy existing template button to use one of the existing templates.	Structural Replace
Save template	Click this button to open the Save Template dialog box, where you have to specify the name of the new template. Note that the new template is stored under the User Defined node of the existing templates tree view.	Both
Edit variables	Calls the Edit variables dialog box to set constraints for template variables.	Both
History	Click this button to open the History dialog box, that shows up to 25 last invoked templates.	Both
Copy existing template	Click this button to open the Existing Templates dialog box, where you can select one of the pre-defined or custom templates. Selected template is displayed in the Preview field. Clicking OK in the Existing Templates dialog box inserts the source code of the template into the Search template or Replace template field.	Both
Recursive matching	If this check box is selected, the search is performed recursively in the results.	Structural Search
Case sensitive	If this check box is selected, the search discerns lower and upper case letters.	Both
File type	Select file type from the drop-down list.	Both
Shorten fully qualified names	This option makes sense in case the template text contains fully qualified class names. If the check box is selected, PyCharm automatically reduces these names in the template. Otherwise, fully qualified class names are used.	Structural Replace
Reformat according to style	Check this option, if you want PyCharm to automatically reformat the expanded code fragment according to your code style settings (for details, refer to the Code Style dialog box). If the option is not checked, PyCharm will only indent the whole template according to the position in code at which it is expanded, leaving its formatting as is.	Structural Replace
Use static import if possible	Check this option, if you want PyCharm to shorten any references to static elements in the replaced code. PyCharm will insert a static import for those elements. The elements are then referenced by their short name. If there are no references to static elements in the replaced code, the option will be ignored.	Structural Replace
Scope	Select one of the existing scopes from the drop-down list or click the ellipsis button (alternatively, press <code>Shift+Enter</code>), and create your own scope in the Scopes dialog box.	Both
Open in new tab	If this check box is selected, the results of the new search display in a new tab in the Find results tool window. Otherwise, the search results update the existing tab.	Both

Structural Search and Replace. Edit Variable Dialog

Edit | Find | Search Structurally | Edit variables

Use this dialog to define constraints for the variables of a [search template](#).

Tip The contents of the dialog box depend on the selected variable type.

ItemDescription

Variables	This area shows a list of variables used in the current search template.
Text constraints	<p>In this area define the following constraints of the selected variable regarding text:</p> <ul style="list-style-type: none">- Text/regular expression - in this text box, type a perl-like expression or a class name to be used as a variable constraint. Basic code completion is available for class names.- Invert condition - select this check box to have the text pattern inverted.- Apply constraint within type hierarchy - select this check box to have the search according to the pattern performed both in type names and in parents (within the hierarchy).- Whole words only - when this check box is selected, only whole words within text are matched. This option recognizes string literals and comments.
Occurrences count	<p>In this area, define how pattern hits will be counted.</p> <ul style="list-style-type: none">- Minimum count - in this text box, type the minimum number of elements in the list.- Maximum count - in this text box, type the maximum number of elements in the list.- Unlimited - select this check box to allow unlimited number of elements in the list.
Expression constraints	<p>In this area, define how expressions should be processed.</p> <ul style="list-style-type: none">- Value is read - if this check box is selected, the matching variable is to be read.- Value is written - if this check box is selected, the matching variable is to be written.- Expression type (regexp) - if the calculated variable is an expression, this constraint checks its type. For instance, for the <code>foo(\$a\$)</code> expression the type of the method parameter would be checked.- Expected type of expression (regexp) - if the calculated variable is matched to any expected type of an expression, this constraint checks the expression type anywhere the expression was used. For instance, correspondence between the method parameter type (e.g. <code>\$a\$</code>) in method calls will be checked for methods like <code>foo(\$a\$)</code>.- Apply constraint within type hierarchy - select this check box to have the search according to the pattern performed both in type names and in parents (within the hierarchy).- Invert condition - select this check box to have the value of the corresponding check box changed to the opposite one.
Script constraints	<p>In this area, define a variable constraint via a script. Specify the script in the text box or click the  button to open the Edit Groovy Script Constraint dialog box. The constraint is applied after the initial matching process is finished.</p>
This variable is target of the search	If this check box is selected, the search results will show not the entire expression but the selected variable(s) only.

Edit Variables. Complete Match Dialog

Edit | Find | Search Structurally | Edit variables | Complete Match

Use the Complete Match dialog to define constraints for the entire pattern that you have specified in the [search template](#).

ItemDescription

Text constraints	<p>In this area define the constraints of the selected variable regarding text. In case of Complete Match this area might not be useful. The constraints are as follows:</p> <ul style="list-style-type: none">- Text/regular expression - in this text box, type a perl-like expression or a class name to be used as a variable constraint. Basic code completion is available for class names.- Invert condition - select this check box to have the text pattern inverted.- Apply constraint within type hierarchy - select this check box to have the search according to the pattern performed both in type names and in parents (within the hierarchy).- Whole words only - when this check box is selected, only whole words within text are matched. This option recognizes string literals and comments.
Contained in constraints	<p>In this area, define additional constraint pattern inside the already defined search template. Specify the pattern in the text box or click the  button to open the Existing Templates dialog box. Invert condition - select this check box to have the value of the corresponding field changed to the opposite one.</p>
Script constraints	<p>In this area, define a variable constraint via a script. Specify the script in the text box or click the  button to open the Edit Groovy Script Constraint dialog box. The constraint is applied after the initial matching process is finished.</p>
This variable is target of the search	<p>If this check box is selected, the search results will show not the entire expression but the selected variable(s) only.</p>

Settings / Preferences Dialog

File | Settings for Windows and Linux

PyCharm | Preferences for macOS

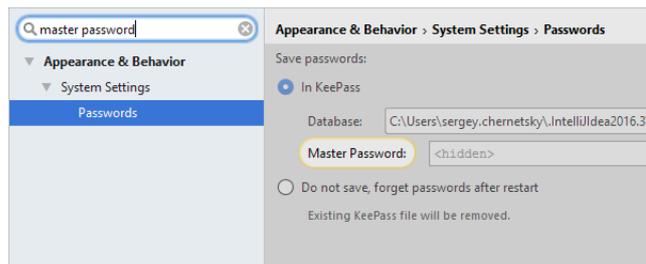
Ctrl+Alt+S



Note The settings that pertain to the current project, are marked with the  icon.

The Settings dialog lets you control every aspect of the PyCharm behavior and appearance.

Use the search box in the upper-left part of the dialog to find the options of interest. Alternatively, you can browse the settings using the hierarchical list of categories (groups of settings) underneath the search box.



On this page find the descriptions of the main [controls](#) of the dialog.

ItemDescription

Search	Enter a search keyword in the text area. While typing the search string, the list of options in the dialog reduces to the matching occurrences.
	Click this button to clear the search area.
OK	Apply changes and close the dialog box.
Cancel	Discard changes and close the dialog box.
Apply	Apply changes and leave the dialog box opened.
Help	Show reference page.

- [Appearance and Behavior](#)
- [Keymap](#)
- [Editor](#)
- [Plugins](#)
- [Version Control](#)
- [Current Project](#)
- [Build, Execution, Deployment](#)
- [Languages and Frameworks](#)
- [Tools](#)

Appearance and Behavior

File | Settings | Appearance and Behavior for Windows and Linux

PyCharm | Preferences | Appearance and Behavior for macOS Ctrl+Alt+S



When you select the Appearance and Behavior category in the left-hand pane, its main subcategories are listed in the right-hand part of the dialog.

- [Appearance](#)
- [Menus and Toolbars](#)
- [System Settings](#)
- [File Colors](#)
- [Scopes](#)
- [Notifications](#)
- [Quick Lists](#)

Appearance

File | Settings | Appearance and Behavior | Appearance for Windows and Linux

PyCharm | Preferences | Appearance and Behavior | Appearance for macOS

Ctrl+Alt+S



Use this page to change the overall look and feel of your IDE.

- [UI Options](#)
- [Antialiasing](#)
- [Window Options](#)
- [Presentation Mode](#)

UI Options

OptionDescription

Theme	Use this drop-down list to select the desired theme from the list. In particular, Darcula theme is available. Changing to/from this theme requires PyCharm restart. In the Community edition of PyCharm, the new default scheme with the name <code>IntelliJ</code> is used.
Adjust colors for red-green vision deficiency	Select this option to adjust the IDE colors (code highlighting in the editor, text notifications, etc.) for people with the red-green color deficiency.
Override default fonts by (not recommended)	Select this check box to enable specifying font family and size to be used instead of the default one. When first installed, PyCharm takes Windows default font size and style.
Cyclic scrolling in list	Select this check box to enable scrolling through a list by jumping from the last item to the first one and vice versa.
Show icons in quick navigation	Select this check box to have icons shown in the quick navigation pop-up menu (<code>Ctrl</code> / <code>Ctrl+Shift</code> / <code>Ctrl+Shift+Alt+N</code>).
Automatically position mouse cursor on default button	Select this check box to have the mouse pointer placed at the default button when a dialog box opens. If the check box is not selected, the pointer location does not change.
Hide navigation popups on focus loss	If this check box is selected, the navigation pop-up frames (go to class/file/symbol) close, when any other PyCharm component gets the focus. If this check box is not selected, the navigation pop-up frames persist on changing the focus, and the only way to close such pop-up lays with pressing <code>Escape</code> .
Drag-n-Drop with ALT pressed only	If this check box is not selected (by default), PyCharm allows moving editor tabs, tool window buttons, files and folders in the Project tool window , using drag-n-drop. Select this check box to avoid accidental moving of a file or folder, or a UI component. Thus drag-n-drop only works while ALT key is pressed.
Tooltip initial delay (ms)	Use this slider to specify the time to pass between the moment you hover the mouse over an item in the editor and the moment when the tooltip with its value appears. This settings is especially important during debugging. If the delay is too short using the mouse becomes inconvenient because every mouse move across the screen brings forward a number of tooltips with the values of all the variables.

Antialiasing

OptionDescription

IDE	From this drop-down list, select which antialiasing mode you want to apply to the IDE (including menus, tool windows, etc.): <ul style="list-style-type: none">- Subpixel: this option is recommended for LCD displays and takes advantage of the fact that each pixel on a colour LCD is composed of red, green and blue sub-pixels. This allows smoothing text and rendering it with greater detail.- Greyscale: this option is recommended for non-LCD displays or displays positioned vertically. It deals with text at the pixel level.- No antialiasing: this option can be used for displays with high resolution, where non-antialiased fonts are rendered faster and may look better.
Editor	From this drop-down list, select which antialiasing mode you want to apply to the Editor : <ul style="list-style-type: none">- Subpixel: this option is recommended for LCD displays and takes advantage of the fact that each pixel on a colour LCD is composed of red, green and blue sub-pixels. This allows smoothing text and rendering it with greater detail.- Greyscale: this option is recommended for non-LCD displays or displays positioned vertically. It deals with text at the pixel level.- No antialiasing: this option can be used for displays with high resolution, where non-antialiased fonts are rendered faster and may look better.

Window Options

OptionDescription

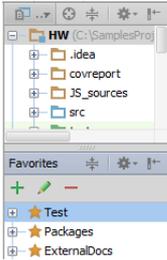
Animate windows	Select this check box to have undocked tool windows slide with the animation effect. This option applies only when a tool window is undocked.
Show memory indicator	Select this check box to show the Memory Indicator on the Status Bar .

Disable mnemonics in menu
Select this check box to hide underlining of hot keys in the PyCharm menus.

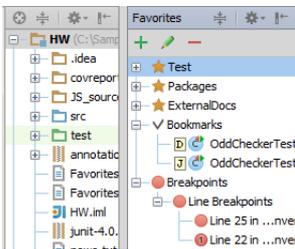
Disable mnemonics in controls
Select this check box to hide underlining of hot keys in the PyCharm controls.

Display icons in menu items
If this check box is selected (by default), the icons are displayed to the left of the menu commands.
If this check box is not selected, the menu commands are displayed without icons.

Side by side layout on the left/right
When these check boxes are selected, the way the tool windows are positioned is optimized for a wide-screen display.
Side-by-side layout is OFF:



Side by side layout is ON:



Toggle layout by `Ctrl+MouseClicked` on splitter between the tool windows.

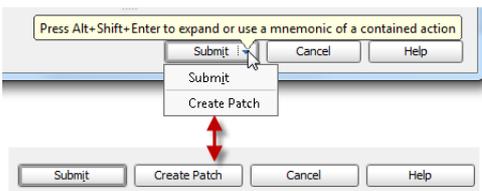
This only applies to the tool windows located on the left and right sides, but not at the top and bottom of the PyCharm window.

Show tool window bars
Select this check box to display tool window bars.

Show tool window numbers
Select this check box to show tool window quick access numbers on the tool window buttons.
You can use the `Alt+number` shortcuts regardless of this setting and change the shortcuts on the [Keymap page](#).

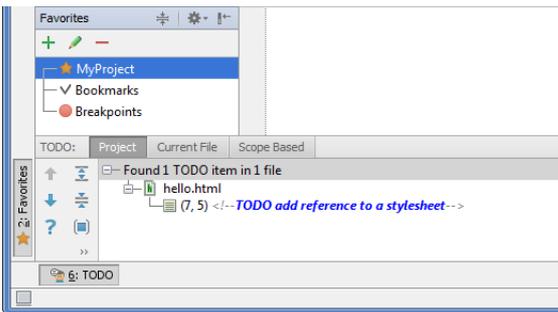
Note that the tool window mnemonics show up only when the corresponding keybindings have the format `Alt+n`, where `n` is an integer number in the range from 1 to 9. In the case of a different keyboard shortcut, the mnemonics are not displayed.

Allow merging buttons on dialogs
If this check box is selected, the multiple commands in a dialog box are grouped under a single button with a down arrow. You can view all merged commands by clicking the drop-down list, or pressing `Shift+Alt+Enter`.
If this check box is not selected, the buttons will be shown in a row. Compare:

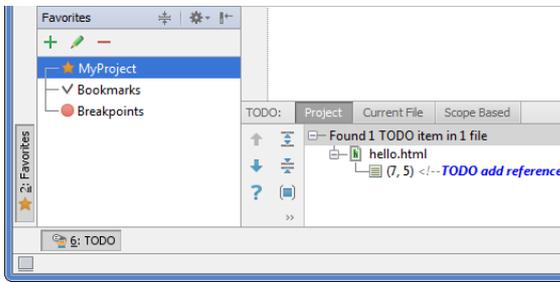


Small labels in editor tabs
If this check box is selected, the font size on the editor tabs is set to the smaller value.
If this check box is not selected, the font size on the editor tabs is set to the default value, as in the project tree view.

Widescreen tool window layout
If this check box is selected, the way the tool windows are positioned is optimized for a wide-screen display.
Widescreen tool window layout is OFF:



Widescreen tool window layout is ON:



Presentation Mode

ItemDescription

Font size Use the drop-down list to select the font size for the [presentation mode](#).

Menus and Toolbars

File | Settings | Appearance and Behavior | Appearance | Menus and Toolbars for Windows and Linux

PyCharm | Preferences | Appearance and Behavior | Appearance | Menus and Toolbars for macOS

Ctrl+Alt+S



Use this page to configure the PyCharm [menus and toolbars](#).

In this topic:

- [Menus and Items List](#)
- [Controls](#)

Menus and Items List

The list shows the items for the menus and toolbars. The items are grouped according to the areas of their use.

To configure an item, expand the corresponding node and select the desired item. After that, the buttons in the right-hand part of the page become available.

Controls

ItemDescription

Add After	Click this button to add a new action to the menu after the selected one. In the Choose Actions to Add dialog box that opens choose the desired action and optionally assign an icon to it.
Add Separator	Click this button to have a separator added to the menu after the selected item.
Edit Action Icon	Click this button to associate an icon with the selected menu item. In the Choose Action Icon Path dialog box that opens specify the path to the desired image. 
Remove	Click this button to delete the selected item from the list.
Move Up	Click this button to move the selected item one position up.
Move Down	Click this button to move the selected item one position down.
Restore All Defaults	Click this button to abandon all the changes made to the all items and return to the default settings.
Restore Default	Click this button to abandon all the changes made to the selected item and return to the default settings.

Choose Actions to Add Dialog

The dialog box opens when you select an item in the Menus and Items List and click the Add After button.

In the dialog box, choose the desired action to be added to the menu or toolbar and optionally assign an icon to it.

ItemDescription

Action List The list shows all the actions available in PyCharm. The actions are grouped below nodes according to the areas of their use.

Icon Path In this text box, specify the location of the file with the icon you want to assign to the selected action. If necessary, use the Browse button  to select the file in the [corresponding dialog](#).

Tip – The image file should have `.png` extension.
– The size of the toolbar icons should be 16x16.

Set Icon Click this button to associate the selected action with the icon specified in the Icon Path dialog box.

System Settings

File | Settings | Appearance and Behavior | System Settings for Windows and Linux

PyCharm | Preferences | Appearance and Behavior | System Settings for macOS

Use this page to configure general behavior of PyCharm.

On this page:

- [Startup/Shutdown](#)
- [Project opening](#)
- [Synchronization](#)
- [Accessibility](#)
- [On Closing Tool Windows with Running Process](#)

Startup/Shutdown

ItemDescription

Reopen last project on startup	Select this check box to have PyCharm re-open the last opened project on startup.
Confirm application exit	Select this check box to have a warning message displayed when you attempt to close PyCharm.

Project opening

ItemDescription

Open project in a new window	Click this radio button to always open a new project in a new window. If this option is selected, the command Attach project appears in the File menu, which allows you to select the folder of the project to be attached.
Open project in the same window	Click this radio button to always close the current project, and reuse the same window.
Confirm window to open project in	Click this radio button to have PyCharm ask you whether you want to open a new project in the same frame, or in a new one.

Synchronization

ItemDescription

Synchronize files on frame or editor tab activation	If this check box is selected, all the files that were changed externally are reloaded from disk when you switch to PyCharm from a different application, or when you switch to their editor tab.
Save files on frame deactivation	If this check box is selected, all modified files are auto saved when you switch from PyCharm to a different application. Note that you cannot disable autosave completely by turning off this and the following option. See Saving and Reverting Changes .
Save files automatically if application is idle for N seconds	If this check box is selected, all modified files are auto saved at regular time intervals. See also, Saving and Reverting Changes .
Use "safe write" (save changes to a temporary file first)	<p>If this check box is selected, a changed file is first saved in a temporary file. If the save operation succeeds, the file being saved is replaced with the saved file. (Technically, the original file is deleted and the temporary file is renamed.)</p> <p>Also, the ownership of such file changes.</p> <p>If this check box is not selected, the ownership of a file does not change, but all the advantages of safe write will be lost.</p>

Accessibility

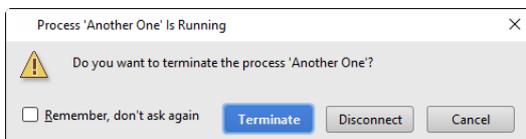
ItemDescription

Support screen readers (requires restart)

On Closing Tool Windows with Running Process

ItemDescription

Terminate	If this option is selected, the running process disconnects and terminates silently.
Disconnect (if available)	If this option is selected, the running process is disconnected.
Ask	If this option is selected, the dialog box shows up:



Ctrl+Alt+S



If to access the Internet PyCharm should use an HTTP proxy, specify the proxy settings on this page.

ItemDescription

No proxy	Click this radio button to connect to the Internet without a proxy.
Auto-detect proxy settings	Click this radio button to enable using an auto-configuration URL to configure the web proxy settings. When this option is selected, the following controls become enabled: ItemDescription Automatic proxy configuration URL Select this check box to manually specify the location of the proxy settings file, in case PyCharm does not find it automatically. Clear passwords Click this button to clear the passwords to the specified proxy.
Manual proxy configuration	Click this radio button to enable manual proxy configuration. When this option is selected, the following controls become enabled: ItemDescription HTTP Click this radio button if you want PyCharm to use an HTTP proxy when accessing the Internet. SOCKS Click this radio button if you want PyCharm to use the Socket Secure protocol when accessing the Internet. Host name Specify the proxy hostname or IP address. Port number Specify the proxy port number. No proxy for Specify here the patterns for the URLs or IP addresses, for which proxy should not be specified. Proxy authentication Select this check box if your proxy requires authentication. Login Specify the name of the user on whose behalf PyCharm will connect to the proxy. Password Specify the password associated with the user name (login). Remember password Select this check box if you want PyCharm to remember the password. Otherwise, you will be asked to provide the password every time PyCharm connects to the proxy.

Ctrl+Alt+S



Use this page to share the statistics of your PyCharm usage with JetBrains.

ItemDescription

Allow to send usages statistics to JetBrains

Select this check box to allow JetBrains to collect your anonymous statistics.

Daily, Weekly or Monthly

Select one of these options to define how often your usage statistics should be sent to JetBrains.

Ctrl+Alt+S



Use this page to:

- Enable automatic update of PyCharm and specify to which kind of release you want it updated.
- Obtain information about the current PyCharm version and availability of a newer version.

ItemDescription

Check for updates for Select this check box to enable the automatic update function, and select the desired update channel (for example, stable version).

- Channel **Early Access Program**: this channel gets patch from the previous EAP/release version. This is not recommended for production development.
More details about the Early Access Program, or EAP, are available at <http://eap.jetbrains.com/>.
- Channel **Beta Releases or Public Previews**: this channel includes release candidates (RC).
- Channel **Stable Releases**: this channel includes all PyCharm releases, for example, PyCharm X.Y.Z

Note that the list is only available for the **stable versions**. For the various EAPs, it is enforced to **Early Access Program**.

Use secure connection

- If this check box is selected, the secure connection protocol (HTTPS) is used.
- If this check box is cleared, the HTTP protocol is used. Note that the HTTP protocol may be blocked due to security reasons.

By default, the check box is selected.

Check Now Click this button to check for updates immediately.

Tip You can alternatively choose Help | Check for Updates (for Windows or *NIX) or PyCharm | Check for Updates (for macOS) on the main menu.

View/edit ignored updates Follow this link to show/change the builds which were ignored on PyCharm update. These build numbers are included in the list of ignored updates and not suggested any more.

Ctrl+Alt+S 

Specify whether PyCharm should save your passwords - ones you use to access password-protected resources such as version control repositories and databases.

When you use [KeePass password manager](#), the master password will be used to access a file that stores individual passwords. Once PyCharm remembers your passwords, it will not ask for the passwords again including the master password unless you need to access the password database.

Under Save passwords, you can configure password settings.

ItemDescription

In native Keychain	PyCharm displays this option for macOS and Linux only. Select this option to use native Keychain for storing your passwords.
In KeePass	Select this option to use the KeePass password manager for storing your passwords.
Database	This field displays the location of your current <code>c.kdbx</code> file. If you need to select another location, click  and in the dialog that opens, choose the appropriate one. If you want to import another <code>c.kdbx</code> file, click  icon and from the drop-down list select Import. If you want to remove the existing passwords in the <code>c.kdbx</code> file, select Clear.
Master Password	Use this field to enter a password that you want to use for accessing <code>c.kdbx</code> file. The first time PyCharm generates a password automatically.
Do not save, forget password after restart	Select this option if you want to remove the <code>c.kdbx</code> file containing individual passwords after the restart.

Password Manager Database Updated

This dialog opens only once when you launch a new PyCharm version with configurations from the previous PyCharm version.

PyCharm lets you convert the current saved password database into a new one.

ItemDescription

Password	Use this field to enter a master password that you have used in the previous version of PyCharm.
Convert	Click this button to store your saved passwords in a new password database.
Clear Password	If you click this button nothing will be converted and your old passwords will not be saved. The same action will be taken if you close this dialog without entering the master password.

File Colors

File | Settings | Appearance and Behavior | File Colors for Windows and Linux

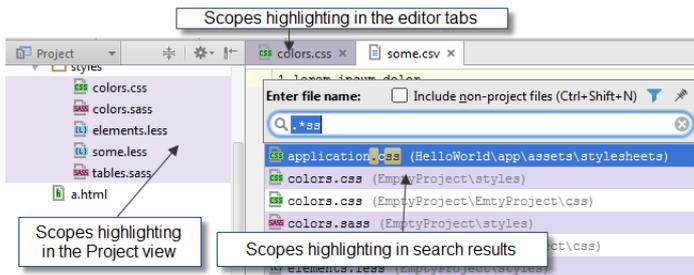
PyCharm | Preferences | Appearance and Behavior | File Colors for macOS

Ctrl+Alt+S



Use this page to set different background colors for distinguishing between project files, folders, and packages of specific **scopes**. The settings apply to the following UI elements:

- The headers of editor tabs.
- [Navigation lists](#) when one searches for files or classes by their names
- [Project view](#)



On this page:

- [Common Options](#)
- [Local Colors](#)
- [Shared Colors](#)

Common Options

ItemDescription

Enable File Colors	Select this check box to apply background color settings to navigation lists .
Use in Editor Tabs	Select this check box to apply background color settings to the headers of editor tabs.
Use in Project View	Select this check box to apply background color settings to the Project view.
Manage Scopes	Click this button to open the Scopes dialog in which you can define custom scopes for various actions.

Local Colors

In this area, configure the color-scope associations to be applied locally.

Once defined, a color-scope association cannot be changed. To re-assign a color to a scope, remove the existing association and define a new one.

ItemTooltip	Description
Scope	This read-only field shows the scope to apply the color setting to.
Color	This read-only field shows the color to be applied to the corresponding scope.
+	Add Click this button to open the Add Color Label dialog in which you can configure a new color-scope association.
-	Remove Click this button to remove the selected color-scope association.
↑ or ↓	Move up or Move down Use these buttons to resort the color-scope associations and thus determine the order in which they are applied.
👁	Share Click this button to have the selected scope-color association shared among the members of the team. The selected association will be accordingly moved to the list in the Shared Colors area.

Shared Colors

Use the controls in this area to configure the color-scope associations to be shared among all the members of the team.

Once defined, a color-scope association cannot be changed. To re-assign a color to a scope, remove the existing association and define a new one.

ItemTooltip	Description
Scope	This read-only field shows the scope to apply the color setting to.
Color	This read-only field shows the color to be applied to the corresponding scope.
+	Add Click this button to open the Add Color Label dialog in which you can configure a new color-scope association.
-	Remove Click this button to remove the selected color-scope association.



Move up or
Move down

Use these buttons to resort the color-scope associations and thus determine the order in which they are applied.



Unshare

Click this button to have the selected scope-color association applied only locally.
The selected association will be accordingly moved to the list in the [Local Colors](#) area.

Scopes

File | Settings | Appearance and Behavior | Scopes for Windows and Linux

PyCharm | Preferences | Appearance and Behavior | Scopes for macOS

Ctrl+Alt+S



A [scope](#) is a set of files to which various operations apply. Using this dialog, you can define scopes for the various PyCharm actions, for example, [Find Usages](#), or [Code Inspections](#).

In this section:

- [Main toolbar](#)
- [Scope configuration controls](#)
- [Examples](#)
- [Scope toolbar](#)

Main toolbar

ItemTooltipDescription

	Add scope	Click this button to add a new local or shared scope.
	Delete	Click this button to delete the selected scope from the list.
	Copy configuration	Click this button to create a copy of the selected scope.
	Save as	Click this button to have the selected local scope saved as shared or a selected shared scope as local.
	Move Up/Move Down	Use these buttons to move the scopes up and down in the list. If some file is included into several scopes, the order of the scopes becomes important: PyCharm uses the color of the uppermost scope (shown in the Scopes settings page) to highlight such file. Of course, you can change the order of the scopes, and thus the resulted highlighting.

Scope configuration controls

ItemDescription

Name	In this text box, specify the scope name.
Pattern	In this text box, specify the pattern that defines the current scope. The following elements and structures can be used: <ol style="list-style-type: none">The <code>file:</code> modifier. The element is mandatory.The <code>*</code> asterisk to denote any symbol in a file name or file extension.Logical operators <code>AND (&&)</code>, <code>OR ()</code>, and <code>NOT (!)</code>.

For more information, see [Scope Language Syntax Reference](#).

Do one of the following:

- Type or edit the pattern manually in the text field Pattern.
- Click or press to type or edit in the Pattern dialog box.
- Choose the desired files in the [Project Tree View](#) and use the [buttons described below](#) to make PyCharm generate the corresponding pattern automatically.

Warning! Storing empty or incorrect patterns is not allowed. In such cases, you will be prompted with the `SyntaxError` warning.

Examples

- `file:*.js||file:*.coffee` - include all JavaScript and CoffeeScript files.
- `file:*.js&&!file:*.min.*` - include all JavaScript files except those that were generated through [minification](#), which is indicated by the `min` extension.

Include	Click this button to have the selected element included in the scope. The corresponding expression is automatically generated and added to the expression in the Pattern text box.
---------	--

Tip If the current element is a folder, the nested subfolders are ignored.

Include Recursively	Click this button to have the selected folder included in the scope, together with the nested subfolders. The corresponding expression is automatically generated and added to the expression in the Pattern text box.
---------------------	--

Exclude	Click this button to have the selected element excluded from the scope. The corresponding expression is automatically added to the Pattern. If the current element is a folder, the nested subfolders are ignored.
---------	--

Exclude Recursively

Click this button to have the selected folder excluded from the scope, together with the nested subfolders. The corresponding expression is automatically added to the Pattern field.

Scope toolbar

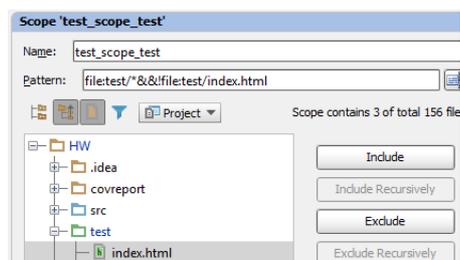
ItemTooltipDescription

Project tree view The tree view contains all the files available in your project. In the view, select the desired files to be included in the current scope and have the scope definition pattern generated automatically. The message on the toolbar shows the total number of available files and the number of files included in the scope. See also the [color legend](#) below. Use the toolbar buttons described below to change the view presentation.

 **Show Modules** When the button is pressed, items in the tree-view are shown below the corresponding module nodes. Otherwise, the project items are shown below the corresponding package (like a source path with packages).

 **Show Module Groups** When the button is pressed, the package structure of a scope is displayed.

 **Show Files** If this button is pressed, source files are displayed explicitly in the tree view. When the files are shown, they can be selected for exclusion/inclusion into a pattern.



If this button is not pressed, the files are hidden.

 **Show Included Only** When the button is pressed, the tree shows only the elements that are included in the scope.

Legend of the project tree view

ItemDescription

Green Folders and files included in scope.

Black Folders and files excluded from scope.

Blue Folders that contain both excluded and included files and subfolders.

Notifications

File | Settings | Appearance and Behavior | Notifications for Windows and Linux

PyCharm | Preferences | Appearance and Behavior | Notifications for macOS

Ctrl+Alt+S



Use this page to enable and disable notifications about certain events, change their presentation, and optionally enable their logging.

ItemDescription

Display balloon notifications Select this check box to enable event notifications for PyCharm. (The notifications, generally, are shown in the balloons that appear on the screen when the corresponding events take place.)

Enable system notifications Select this check box to allow showing system notification.

Warning! This option is not available on platforms where system notifications are not supported (Windows and some Unixes).

Group This column lists groups of events that you may be notified of and/or that may be logged.

Popup If the Display balloon notification check box is selected, the settings in this column specify how the notifications for the corresponding group of events are shown.
The available display options are:

- Balloon: The balloons with the notification messages appear on the screen for a short period of time and then disappear automatically. The notifications are also shown in the Status bar, and added to the list of notifications.
- Sticky balloon: The notification balloons stay on the screen unless you close them.
- Tool window balloon: The notification balloons are shown only if an appropriate tool window is open.
- No popup: The notifications for the corresponding group of events are not shown.

Log If the check box for a group of events is selected, the corresponding events are logged and can be seen in the [Event Log tool window](#).

Quick Lists

File | Settings | Appearance and Behavior | Quick Lists for Windows and Linux

PyCharm | Preferences | Appearance and Behavior | Quick Lists for macOS

Ctrl+Alt+S



Use this page to configure quick lists. A Quick List is a pop-up menu of PyCharm commands, configured by the user and associated with a keyboard or mouse shortcut. You can create as many quick lists, as necessary. Each command, included in a quick list, is identified by a sequential number. Numbering starts from the numerals (0 to 9), and then proceeds with the letters in alphabetical order.

Item ShortcutDescription

+	Alt+Insert	Create a new Quick List.
-	Alt+Delete	Delete the selected Quick List.
Display name		Edit the name of the selected Quick List.
Description		Edit the description of the selected Quick List. (The description is optional.)
+	Alt+Insert	Use this button to add actions to the Quick List. Select the actions in the Add Actions to Quick List dialog that opens.
-----		Use this button to add a separator at the end of the Quick List. (Separators help you organize menu commands in logical groups.)
-	Alt+Delete	Remove selected actions from the Quick List.
↑	Alt+U	Use this button to move the selected item one line up in the list.
↓	Alt+D	Use this button to move the selected item one line down in the list.

Keymap

Use this page to create, edit, and remove custom keymaps for specific environments, and change shortcuts associated with actions.

Note that default keymaps are not editable. To re-configure shortcut associations, create a child keymap based on the desired default one and edit it as required.

On the other hand, as soon as you try to change a keyboard shortcut associated with an action in one of the default keymaps, a copy of the corresponding keymap is automatically created.

- [Keymap Management Buttons](#)
- [Keymap Toolbar](#)
- [Actions](#)

Keymap Management Buttons

ItemDescription

Keymaps	From this drop-down list, select the desired keymap.
Copy	Click this button to create a child keymap on the basis of the keymap selected in the Keymaps drop-down list.
Reset	Click this button to abandon all the changes made to a custom keymap and restore the configuration of the parent keymap.
Delete	Click this button to remove the selected custom keymap from the list.
<language> layout support	This check box appears when a non-English keyboard layout has been detected. When you type any character in the editor, the keyboard layout is recognized and a notification appears.

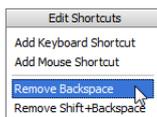
Warning! This option is available for Windows and Linux only. If you don't see this option, it means either you are using macOS or PyCharm didn't detect any of the supported input locales.

Based on keymap This read-only field shows the name of the parent keymap.

Keymap Toolbar

ItemTooltip Description and shortcut

	Expand All	Click this button to expand all nodes in the content pane of actions. Ctrl+NumPad Plus
	Collapse All	Click this button to collapse all nodes in the content pane of actions. Ctrl+NumPad -
	Edit Shortcut	Click this button to change shortcuts for the selected action. It is possible to remove existing shortcuts, and add new ones. Choose the desired change from the drop-down menu: Enter



- Select the option Add Keyboard Shortcut to open the [Enter Keyboard Shortcut](#) dialog box, where you can specify the combination of keystrokes to be assigned to the selected action in the current keymap.
- Select the option Add Mouse Shortcut to open the [Enter Mouse Shortcut](#) dialog box, where you can specify the combination of mouse clicks and buttons to be assigned to the selected action in the current keymap.
- Select the options Remove <shortcut> to delete the selected shortcut from the selected action.

These commands are duplicated on the context menus of the actions in the Actions content pane.



Use this text box to search through the content pane of actions. As you type a search string, the actions that match the search pattern are displayed.

The previously used search patterns are stored in the search history list. To add the search string to the history list, press [Enter](#).

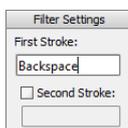
Click [Q](#) to reveal the history list of the previous searches.

Click [C](#) to clear the current search pattern from the text box.



Find Action by Shortcut

Click this button to open the Filter Settings dialog box for filtering out the desired actions by keystrokes. Refer to the section [Configuring Keyboard Shortcuts](#) to learn how to specify keyboard shortcuts.



The actions with shortcuts that match the specified criteria are shown in the content pane of actions.



Clear Filtering

Click this button to restore the initial list of actions in the content pane.

Actions

ItemDescription

All Actions	<p>This content pane shows all actions currently available in PyCharm. The actions are grouped below nodes according to the areas of their use.</p> <ul style="list-style-type: none">- Note that default keymaps are not editable. As soon as you try to change a keyboard shortcut associated with an action, a copy of the corresponding keymap is automatically created.- If some of the actions have no mapped keyboard shortcuts, they still can be invoked by Go to Action.
Shortcuts	<p>This read-only field shows the list of shortcuts associated with the selected action in the current keymap.</p> <div><p>Tip The shortcuts are represented depending on the platform. However, some keys are missing on certain keyboard layouts. For example,  /  keys are not available on notebooks. That's why one should use  plus arrow keys.</p></div>
Context menu of an action	
Add Keyboard Shortcut	Choose this command on the context menu of an action to open the Enter Keyboard Shortcut dialog box, where you can specify the combination of keystrokes to be assigned to the selected action in the current keymap.
Add Mouse Shortcut	Choose this command on the context menu of an action to open the Enter Mouse Shortcut dialog box, where you can specify the combination of mouse clicks and buttons to be assigned to the selected action in the current keymap.
Add abbreviation	Choose this command to add an abbreviation that can be used in Search Everywhere .
Remove <shortcut>/<abbreviation>	Choose this command on the context menu of an action to delete the selected shortcut or abbreviation.

Enter Mouse Shortcut Dialog

The dialog box opens when you select an action and click the Add Mouse Shortcut button. Use this dialog box to bind the selected action with a new mouse shortcut, which may be a single or a double clicking one of the mouse buttons or the wheel button.

The resulting mouse shortcut is marked with the  icon in the Shortcuts list.

ItemDescription

Click Count	In this area, specify the type of mouse click to be assigned to the selected action. The available options are: <ul style="list-style-type: none">– Single Click– Double Click
-------------	---

Click Pad	Click the desired mouse button anywhere in this area.
-----------	---

 The number of clicks in this area does not affect the shortcut configuration. No matter how many times you click a button, the click type chosen in the Click Count area will be assigned.

Shortcut Preview	This read-only field shows the newly defined shortcut.
------------------	--

Conflicts	This read-only field shows messages about conflicts that arise if a suggested mouse shortcut is already in use.
-----------	---

 You can ignore a conflict and assign a shortcut to several actions. However it is strongly recommended that you avoid binding two actions with the same shortcut, because the priority of these actions is not defined.

Enter Keyboard Shortcut Dialog

The dialog box opens when you select an action and click the Add Keyboard Shortcut button. Use this dialog box to bind the selected action with a new keyboard shortcut, which may consist of one or two keystrokes. The resulting keyboard shortcut is marked with the  icon in the Shortcuts list.

Warning! Use your mouse pointer to click buttons in the dialog box. Any key stroke is interpreted as a shortcut!

ItemDescription

First Stroke	Use this text box to define the primary shortcut by pressing keyboard keys and key combinations.
Enable	Select this check box to allow an optional second shortcut.
Second Stroke	Use this text box to define an optional shortcut by pressing keyboard keys and key combinations. This field is available after the Enable check box is selected.
Shortcut Preview	In this read-only field, view the newly defined shortcuts.
Conflicts	This read-only field shows messages about conflicts that arise if a suggested keystroke is already in use.

Warning! You can ignore a conflict and assign a shortcut to several actions. However it is strongly recommended that you avoid binding two actions with the same shortcut, because the priority of these actions is not defined.

Editor

File | Settings | Editor for Windows and Linux

PyCharm | Preferences | Editor for macOS

Ctrl+Alt+S



When you select the Editor category in the left-hand pane, its main subcategories are listed in the right-hand part of the dialog.

- [General](#)
- [Colors and Fonts](#)
- [Code Style](#)
- [Inspections](#)
- [File and Code Templates](#)
- [File Encodings](#)
- [Live Templates](#)
- [File Types](#)
- [Emmet](#)
- [Images](#)
- [Intentions](#)
- [Language Injections](#)
- [Spelling](#)
- [TextMate Bundles](#)
- [TODO](#)

General

File | Settings | Editor | General for Windows and Linux

PyCharm | Preferences | Editor | General for macOS

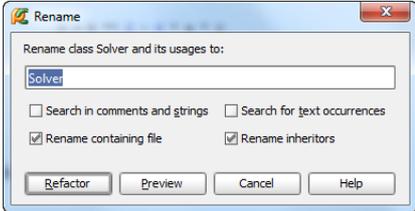
Ctrl+Alt+S



Use the General page of the Settings/Preferences dialog to configure the editor behaviour and customize its view.

ItemDescription

Mouse	
Honor "CamelHumps" word settings when selecting using double click	Select this check box to have PyCharm invoke the CamelHumps selection when words are selected by double-clicking. This feature works only if the Use 'CamelHumps' words option is enabled.
Change font size (Zoom) with Ctrl/Command+Mouse Wheel	If this check box is selected, a particular editor font size can be changed by rolling the mouse wheel while holding the  key. This check box also affects font size in quick documentation lookup . If this option is unchecked, rolling the mouse wheel while holding the  key scrolls the editor.
Enable Drag'n'Drop functionality in editor	If this check box is selected, you can drag-n-drop code fragments in the editor. Refer to Using Drag-and-Drop in the Editor .
Soft Wraps	
Use soft wraps in editor	If this check box is selected, soft wraps (or word wraps) are used in the editor. The horizontal scroll bar is not normally shown when this option is enabled. However, in certain cases, when a line cannot be "soft-wrapped", the horizontal scroll bar still appears (for example, if a line consists of a single string that is wider than the visible area.)
Use original line's indent for wrapped parts	Select this check box to use custom indentation for soft wraps on resizing the editor or console. Specify the indent value in the Additional shift text field on the right.
Show soft wrap indicators for current line only	If this check box is selected, the soft wrap characters <code>↵</code> will be shown in the active logical line only. Otherwise, soft wraps characters will be shown at the end of each line, and at the beginning of each next line.
Virtual Space	
Allow placement of caret after end of line	If this check box is cleared, the caret never rests after the last symbol in a line.
Allow placement of caret inside tabs	Select this check box to allow placing the caret inside tab characters. The reason is that each tab character shows in the editor as a set of 'virtual' space characters.
Show virtual space at file bottom	If this check box is selected, the currently edited line (even if it is the final line) can be scrolled to the top of the screen. PyCharm adds the necessary amount of virtual lines.
Other	
Strip trailing spaces on Save	From this drop-down list, select the mode in which PyCharm will handle trailing spaces in the end of lines on file saving: <ul style="list-style-type: none">– Modified lines: Strips trailing spaces only in the end of the changed lines.– All: Strips trailing spaces in all lines.– None: Does not strip trailing spaces.
<div style="background-color: #ffff00; padding: 5px;">Note Significant trailing spaces which affect an output of a program, are not removed where applicable. For example, trailing spaces in the multiline strings in Groovy are not removed etc.</div>	
Ensure line feed at file end on Save	Select this check box to have PyCharm automatically add an empty line in the end of a file during the save procedure.
Show quick documentation on mouse move	Select this check box to show quick documentation for the symbol at caret. The quick documentation pop-up window appears after the specified delay.
Highlight modified lines in gutter	Select this check box if you want added/modified lines to be highlighted with a color stripe in the left editor gutter.
Different color for lines with whitespace-only modifications	This option only becomes available if the Highlight modified lines in gutter option is enabled. Select this check box if you want lines where only whitespaces were added/removed to be highlighted with a different color from lines with more significant modifications.
Highlight on Caret Movement	
Highlight matched brace	Select this check box to have PyCharm highlight pairs of opening/closing braces when you position the caret right before the opening or right after the closing one. It also works for HTML and XML tags.
Highlight current scope	Select this check box to have PyCharm highlight the available scope for the code typed in the current caret location.
Highlight usages of element at caret	Select this check box to have PyCharm highlight all usages of the element at which the caret is currently positioned.
Formatting	
Show notification after reformat code action	Select this check box to show a notification with changes in your code and a shortcut to the Reformat Code dialog every time you try to reformat the code. Otherwise, PyCharm will reformat code silently.

Show notification after optimize imports action	Select this check box to show notification with changes in your code. Otherwise, PyCharm will optimize imports silently.
Scrolling	
Smooth scrolling	Select this check box to enable smooth scrolling in the editor.
Prefer scrolling editor canvas to keep caret line centered	Click this option to choose scrolling editor canvas and keeping the caret in place. Keeping the caret in place and scrolling the editor canvas can be helpful in course of debugging session . As you step through the lines of code, the editor canvas scrolls, while the line at caret is always in the center of the screen.
Prefer moving caret line to minimize editor scrolling	Click this option to choose moving the caret. When you step through the lines of code during the debugging session , the caret moves down, and the editor canvas doesn't scroll until the caret line reaches the bottom of the screen.
Refactorings	
Enable in-place mode	Select or clear this check box to enable or disable in-place refactorings for Python. The in-place in connection with the refactorings means specifying all or most of the information necessary for the refactoring by typing, right in the editor. All the affected code fragments are highlighted and change as you type. If appropriate, additional refactoring options are selected in corresponding option boxes. The in-place refactoring mode is available for the following refactorings: <ul style="list-style-type: none"> - Extract Constant - Extract Field - Extract Variable - Rename If this check box is not selected, the refactoring settings for all of the refactorings are specified in the corresponding dialogs.
Preselect old name	If this check box is selected, the old name of a symbol is selected when the Rename refactoring is invoked for that symbol.  If check box is not selected, the symbol being renamed is not selected.
Show inline dialog for local variables	Select this check box if you want to display a confirmation dialog for the "Inline local variable" refactoring.
Limits	
Maximum number of contents to keep in clipboard	In this text box, specify how many code blocks can be kept in clipboard.
Recent file limit	In this text box, specify how many file names can be included in the list of recent files.
Rich-text copy	
Copy as rich text by default	Select this check box to copy a rich text from the editor to any other editor that recognizes RTF. Note that you can override this option if you select Copy as Plain Text from the context menu in your editor and vice versa, using the Copy as Rich Text option from the context menu overrides the unselected check box in the editor settings.
Color scheme	Use this drop-down list to select a color scheme for the text copy. You can select from the following options: <ul style="list-style-type: none"> - Default - Active scheme - Darcula
Error highlighting	
Error stripe mark min height (pixels)	In this text box, specify the minimum size of the error and warning stripes.
Autoparse delay (ms)	In this text box, specify the time period after which PyCharm starts reparsing the entered text.
'Next Error' action goes to high priority problems only	Select this check box to have PyCharm pass through the highest priority problems only (for example, errors), when executing Navigate Next/Previous Highlighted Error command (F2 / Shift+F2). Clear this check box to have PyCharm pass through all the existing problems (for example, errors and warnings) sequentially.

Ctrl+Alt+S



- XML

- TypeScript

- Python

XML

Show import pop-up Automatically display an import pop-up dialog box when typing the name of an unbound namespace.

TypeScript

Show import pop-up Automatically display import pop-up dialog box when typing the name of a symbol that lacks import statement.

Python

Show import pop-up Automatically display an import pop-up dialog box when typing the name of a class that lacks an import statement.

Preferred import style Select the style of import statement to be generated. The possible options are:

- `from <module> import <name>`
- `import <module>.<name>`

Ctrl+Alt+S



Use this page to customize the appearance of the Editor.

ItemDescription

Caret blinking (ms)	Select this check box to make the caret blink with the specified period (in milliseconds).
Use block caret	Select this check box to have the block caret applied in the Insert mode and the usual caret applied in the Overwrite mode. Clear this check box to have the usual caret applied in the Insert mode and the block caret applied in the Overwrite mode.
Show right margin (configured in Code Style options)	Select this check box to have a thin vertical line at the right margin of the editor displayed. Refer to the description of the Code Style settings .
Show line numbers	Select this check box to have line numbering shown in the left gutter area.
Show method separators	Select this check box to have thin lines displayed in classes, scripts or tests to separate methods.
Show whitespaces	Select this check box to have PyCharm display white spaces or tabs (depending on the Code Style settings).

- You can select the following options:
- Leader - select this check box to add white spaces before your code line.
 - Inner - select this check box to display white spaces inside the line of your code.
 - Trailing - select this check box to display white spaces after the code line.

Show vertical indent guides	Select this check box to have PyCharm display vertical lines in the editor to indicate positions of indents and thus facilitate typing, manual formatting, reading, and maintaining code.
-----------------------------	---

Show code lens on scrollbar hover	Select this check box to enable lens mode .
-----------------------------------	---

Show breadcrumbs	Select this check box to show a breadcrumb trail on top of the editor tab for an HTML or an XML file. Reopen the editor for the changes to take effect.
------------------	---

```
topic content table conditional tr td
```

(XML)

```
html body table tr td
```

(HTML)

Show parameter name hints	If the check box is selected, the parameter name hints appear in the editor for SQL. E.g. the column name hints may be shown for SQL INSERT statements.
---------------------------	---

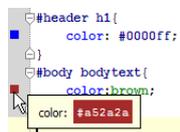
```
1 INSERT INTO family (member_id, name, relation) VALUES
2 ( member_id: 1, name: 'Chloe', relation: 'mother');
```

Here is how the same statement is shown when this check box is not selected.

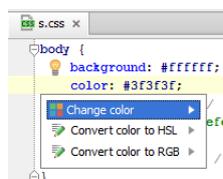
```
1 INSERT INTO family (member_id, name, relation) VALUES
2 (1, 'Chloe', 'mother');
```

Click Configure to change the contents of the Blacklist.

Show CSS color preview icon in gutter	Select this check box to show color preview icons for the color values. See Changing Color Values in Style Sheets .
---------------------------------------	--



If this check box is not selected, it is still possible to invoke the color picker and change color values, by choosing the Change color intention action.



Show CSS color preview as background	If this check box is selected, the background of the color value shows the color preview.
--------------------------------------	---

```
1 body{
2   background: #010203;
3   color: blue;
4 }
```

Enable XML/HTML tag tree highlighting

Select this check box to show the hierarchy of tags highlighted with different colors. If this option is enabled, you can define the following options:

- Levels to highlight: specify the depth of hierarchy to be highlighted.
- Opacity: specify brightness of highlighting



Highlighting is activated when there is more than one tag with the same name in the hierarchy.

Ctrl+Alt+S



Use this page to configure the [code completion](#), and [parameter information](#) settings.

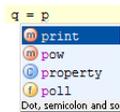
ItemDescription

Code Completion

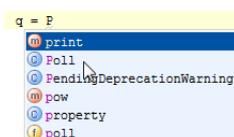
Case sensitive completion

From this drop-down list, select the degree to which you want PyCharm to take into consideration the case sensitivity when suggesting matches for code completion. The available options are:

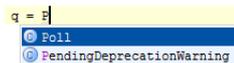
- All: The lookup list includes only those items that match the case of all typed letters. This option is most restrictive.



- None: The lookup list includes all matches regardless of their case.



- First letter: The lookup list includes only the items with the first letter matching.



Auto-insert when only one choice on:

When the check boxes in this section are selected, PyCharm doesn't show a lookup list for the corresponding completion type in cases when only one variant of code completion is available, and completes code automatically.

Sort lookup items lexicographically

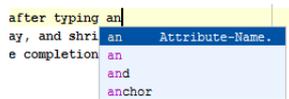
If this check box is selected, the entries in the suggestion list will be sorted according to their lexical order.

If this check box is not selected, the entries in the suggestion list will be sorted by relevance.

Note that the check box defines the default behavior. You can change it any time by clicking the **A** or **π** icons in the suggestion list. Refer to section [Auto-Completing Code](#) for details.

Autopopup code completion

Select this check box, if you want suggestion list to appear after typing anything.



If the check box is not selected, PyCharm will not suggest code completion automatically.

Insert selected variant by typing dot, space, etc.

If this check box is selected, code is completed by pressing certain character: comma, colon, semicolon, opening parentheses of the various kinds, equality sign, asterisk. This option is turned off by default.

Autopopup documentation (ms)

Select this check box to have PyCharm automatically show a pop-up window with the documentation for the class, method, or field currently highlighted in the lookup list.

In the text field to the right, specify the delay (in milliseconds), after which the pop-up window should appear.

For explicitly invoked completion

If this check box is not selected, use **Ctrl+Q** to show quick documentation for the element at caret.

Quick documentation window will automatically pop up with the specified delay in those cases only, when code completion has been invoked explicitly. For the automatic code completion list, documentation window will only show up on pressing **Ctrl+Q**

Parameter Info

Autopopup in (ms)

Select this check box to have PyCharm automatically show a pop-up window with all available method signatures, when an opening bracket is typed in the editor, or a method is selected from the lookup list.

In the text field to the right, specify the delay (in milliseconds) after when the pop-up window should appear.

If this check box is not selected, use **Ctrl+P** to show the parameter info.

Show full signatures

If this check box is selected, the parameter info displays full signatures, including the method name and returned type.

Code Folding

File | Settings | Editor | General | Code Folding for Windows and Linux

PyCharm | Preferences | Editor | General | Code Folding for macOS

Ctrl+Alt+S



Use this page to specify your [code folding](#) preferences.

ItemDescription

Show code folding outline Select this check box if you want the code folding toggles (☰, ☱ and ☲) to be shown in the editor. Clear the check box to hide the toggles.

Collapse by default Select the code fragments which should be folded by default, that is, when a file is first opened in the editor.

Ctrl+Alt+S



Use this page to configure the appearance of the editor tabs and tab headers, specify their positioning on the screen, and define the tab closing policy.

ItemDescription

Tab Appearance

Placement	<p>Use this drop-down list to define the location of the editor tab. The available options are:</p> <ul style="list-style-type: none"> – Top - the default setting. – Bottom – Right – Left – None - select this option to have single editor without any tabs displayed. <p>Refer to the section Changing Placement of the Editor Tab Headers for details.</p>
Show tabs in single row	<p>Select this check box to have headers of currently opened editor tabs displayed in a single row. As a result, some tab headers may become invisible. To cope with this problem, use the command Window Editor Tabs Show All Tabs. Refer to the section Navigating Between Editor Tabs.</p> <p>If this check box is selected, the sorting in alphabetical order mode becomes available for the top and bottom placement of the editor tabs.</p> <p>If this check box is not selected, headers of all the currently opened tabs are displayed, possibly, in several rows.</p>
Hide tabs if there is no space	<p>If this check box is selected, PyCharm shows as many tabs as fits into the current PyCharm frame; the rest of the tabs are hidden under the  drop-down:</p>  <p>If this check box is not selected, all the editor tabs are shown; so doing, each tab's size reduces:</p>  <p>This check box becomes enabled when Show tabs in single row check box is selected.</p>
Hide file extensions in editor tabs	<p>Select this check box to have only file names displayed in editor tab headers.</p>
Show directory in editor tabs for non-unique file names	<p>If this check box is selected, the editor tabs will show the file name together with the parent directory name;</p> <p>if this check box is not selected, only the file name will be included in the editor tab.</p>
Show "close" button on editor tabs	<p>Select this check box to have the Close Active Editor button  displayed in editor tab headers.</p>
Mark modified tabs with asterisk	<p>If this check box is selected, changed but yet unsaved files have an asterisk  on their editor tabs.</p>
Show tabs tooltips	<p>If this check box is selected, a tooltip with the complete path to a file displays on hovering the mouse pointer over a tab.</p> <p>If this check box is not selected, a tooltip is not shown.</p>
Tab Closing Policy	
Tab limit	<p>In this text box, specify the maximum number of the editor tabs to display.</p>
Navigation from non-modified tab will reuse it	<p>Use this option to specify the PyCharm behaviour on  (see section Viewing Definition for details). This option allows you to avoid cluttering of the editor space.</p> <p>If this check box is selected, then, if a file in an editor tab has not been modified and the users has navigated from this file, the target file opens in the same tab. If a file has been modified, then the target file opens in a new tab.</p> <p>If this check box is not selected, the target file always opens in a new tab.</p> <p>Note that a file is considered modified if its VCS status has changed.</p>
When number of opened editors exceeds tab limit	<p>In this area, specify which editor tab should be closed when the tab limit is reached and the user attempts to open a new file. The available options are:</p> <ul style="list-style-type: none"> – Close non-modified files first - if this option is selected, PyCharm examines the tabs in the order they were opened and closes the first tab with content that has not been modified. – Close less frequently used files - if this option is selected, PyCharm closes the tab with the less frequently modified content.
When closing active editor	<p>In this area, specify which editor tab to activate when closing the currently active tab. The available options are:</p> <ul style="list-style-type: none"> – Activate left neighbouring tab - if this option is selected, PyCharm activates the closest tab to the left from the tab being closed. – Activate right neighbouring tab - if this option is selected, PyCharm activates the closest tab to the right from the tab being closed. – Activate most recently opened tab - if this option is selected, PyCharm activates the tab with the file which was opened last.

Ctrl+Alt+S



Use this page to hide or show the icons in the gutter area that invoke actions related to the basic, PyCharm-wide features or to framework- and technology-specific features for all the newly created editors.

The right-hand pane shows all the gutter icons available in PyCharm. The basic, PyCharm-wide features, such as,  (Run), are displayed at the top of the list under the Common title. Other features are grouped by the frameworks and technologies to which they are related .

Note that a group of technology-related features is displayed only if the corresponding plugin is installed and activated, see [Enabling and Disabling Plugins](#) and [Installing, Updating and Uninstalling Repository Plugins](#) for details.

- To have an icon displayed in the gutter area, find the icon or the corresponding action in the list and then select the check box next to it.
- To have an icon hidden, clear the check box next to it.

Ctrl+Alt+S

[Overview](#)[Controls](#)

Overview

Postfix code completion lets you transform an already typed expression to another one based on the postfix you type after a dot, the type of the expression, and its context. This transformation is performed by expanding the postfix-specific predefined template.

For example, the `.if` postfix applied to an expression wraps it with an `if` statement.

BeforeAfter

```
function m(arg) {
  arg.if
}
```

```
function m(arg) {
  if (arg) {
  }
}
```

See more at: [Postfix Code Completion](#).

On this page, enable and disable postfix templates and appoint the key to activate the template expansion.

Controls

ItemDescription

Enable postfix completion – Select this check box to have PyCharm transform expressions with postfixes into other expressions by expanding postfix-specific templates. When the check box is selected, choose the postfixes to apply transformations to by selecting the check boxes next to the desired postfixes in the list below.
– When this check box is cleared, no template expansion is applied.

Expand template with From this drop-down box, choose the key that will invoke template expansion. The available options are: `Tab`, `Space`, and `Enter`.

Table of available postfix templates The table below shows the list of available postfix templates. To enable or disable a template, select or clear the check box next to it. When you select a template, the right-hand pane shows its description and illustrates how it works by displaying the expression before and after the selected template is applied.

Ctrl+Alt+S



Use this page to enable or disable specific smart keys and to define which actions you want to be invoked automatically.

Item	Description
Home	When this check box is selected, on pressing <code>Home</code> , the caret is positioned at the first non-space character of the current line. Pressing <code>Home</code> subsequently moves the caret from the <i>Smart Home position</i> to the first column and back.
End (on blank line)	When this check box is selected, on pressing <code>End</code> in an empty line, the caret is positioned with the indent, which PyCharm assumes to be reasonable in the current code point (indentation is based on the current Code Style Settings).
Insert pair bracket	Select this check box to have PyCharm automatically add a closing round or square bracket for each typed opening round or square bracket, respectively.
Insert pair quote	Select this check box to have PyCharm automatically add a closing single or double quote for each typed opening single or double quote, respectively. See page Creating Documentation Comments . Tip Generation of docstrings on pressing <code>Space</code> after typing opening triple quotes only works when the check box <code>Insert pair quote</code> is cleared in the page Smart Keys of the editor settings.
Reformat block on typing "]"	If this check box is selected, then, on typing the closing curly brace, the enclosed code block is reformatted automatically, if the formatting of this code block does not match the selected code style.
Use 'CamelHumps' words	Select this check box to have PyCharm discern separate words within CamelHump names. Words within a name should start with a capital letter or an underscore. This option impacts some editor actions, for example: <ul style="list-style-type: none"> - Caret Move (<code>Ctrl+Right</code> / <code>Ctrl+Left</code>) - Caret Move with Selection (<code>Ctrl+Shift+Right</code> / <code>Ctrl+Shift+Left</code>) - Select Word at Caret (<code>Ctrl+W</code>) - Delete to Word Start/End (<code>Ctrl+Backspace</code> and <code>Ctrl+Delete</code> respectively) - Double-clicking
Surround selection on typing quote or brace	If this check box is selected, the selected text on typing a quote, double-quote or brace, will be surrounded with these characters. If this check box is not selected, then the typed quotes, double-quotes or braces will replace the selection.
Add multiple carets on double <code>Ctrl</code> / <code>⌘</code> with arrow keys	If this check box is selected, then: <ul style="list-style-type: none"> - pressing <code>Ctrl</code> (for Windows or *NIX) or <code>⌘</code> (for Mac OS) twice plus up/down arrow keys leads to creating multiple carets. - pressing <code>Ctrl</code> (for Windows or *NIX) or <code>⌘</code> (for Mac OS) twice plus left/right arrow keys or Home/End leads to creating a selection.
Enter	Use this area to define the actions to be invoked by pressing <code>Enter</code> . <ul style="list-style-type: none"> - Smart Indent - select this check box to have PyCharm add a new line and position the caret in it, with the indent that PyCharm assumes to be reasonable in the current point of code (indentation is based on the current Code Style settings). If the check box is cleared, upon pressing <code>Enter</code> in a blank line, PyCharm adds a new line and positions the caret at the current non-space character column. - Insert pair '}' - select this check box to have PyCharm automatically position a closing brace <code>}</code> at the proper column when <code>Enter</code> is pressed in an empty line. In this case PyCharm seeks backward for the nearest unclosed opening brace <code>{</code> and places the closing one at the corresponding indentation level. - Insert documentation comment stub - this check box defines the behavior on pressing <code>Enter</code> after an opening documentation tag. <ul style="list-style-type: none"> - If this check box is selected, PyCharm generates a documentation comment stub. For the function comments, this stub contains the required tags (<code>@param</code> tags for each parameter declared in the signature, and <code>@return</code>). Refer to Creating Documentation Comments, Creating JSDoc Comments for details. - If this check box is not selected, only the closing tag is generated. Warning! Note that this check box refers to JavaScript, and the other languages that have special beginning of documentation comments. This check box does not refer to Python.
Backspace	Use this drop-down list to define the actions to be invoked by pressing <code>Backspace</code> key. The available options are: <ul style="list-style-type: none"> - Disabled - pressing <code>Backspace</code> returns the caret by one position at a time. - To nearest indent position - To proper indentation
Reformat on paste	Use this drop-down list to specify how to place pasted code blocks. The available options are: <ul style="list-style-type: none"> - None - The pasted code is inserted at the caret location as plain text without any reformatting or indenting. - Indent Block - The pasted code block is positioned at the proper indentation level, according to the current Code Style Settings, but its inner structure is not changed. - Indent Each Line - Each line of the pasted code block is positioned at the proper indentation level, according to the current Code Style Settings. - Reformat Block - The pasted code block is reformatted according to the current Code Style Settings. Tip This feature is applicable to lines that contain the trailing line feed characters.

XML/HTML

In this area, define the actions to be invoked automatically when editing XML or HTML code.

- **Insert closing tag on tag completion:** select this check box to have PyCharm automatically insert a closing XML or HTML tag upon entering the corresponding opening one.
- **Insert required attributes on tag completion:** select this check box to have PyCharm display a template with all mandatory attributes of the typed tag.
- **Insert required subtags on tag completion:** select this check box to have PyCharm display a template with all mandatory subtags.
- **Start attribute on tag completion:** select this check box to have PyCharm display a template with the first mandatory attribute of the typed tag.
- **Add quotes for attribute value on typing '=':** select this check box to have PyCharm automatically add quotes for the value of the attribute that you are currently typing.
- **Auto-close tag on typing '</':** select this check box to automatically add a closing tag after entering </>. Clear this check box to turn off such auto-completion.
- **Simultaneous <tag></tag> editing:**
 - When this check box is selected and you edit an opening tag the corresponding closing tag is automatically changed accordingly.
 - If this check box is cleared, editing the opening tag does not affect the closing tag which remains unchanged. As a result, the opening and closing tags do not match and the entire construct is underlined as erroneous.

This check box controls the behaviour of PyCharm in the following contexts:

- HTML files
- HTML injections within JavaScript code
- HTML with templates **Handlebars/Mustache** templates
- Handlebars/Mustache template files with the extension `.hbs`
- XML, XHTML files
- DTD files
- JSX files

CSS

In this area, define the selection of CSS identifiers/classes:

- **Select whole CSS identifier on double-click:** If this check box is selected, double-click on a CSS identifier or class name selects the entire name up to the prefix:

```
.ic-arrow-left
```

If this check box is not selected, double-click on a CSS identifier or class name selects a portion of a name up to the nearest hyphens:

```
.ic-arrow-left
```

Smart indent pasted lines

If this check box is selected, the pasted lines are indented relative to the current caret location. Otherwise, the pasted lines are indented relative to the first column.

Insert backslash when pressing Enter inside the statement

If this check box is selected, the continuation character will be inserted automatically on pressing `Enter`, preserving the correct syntax.

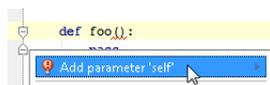
If this check box is not selected, the line will be broken, and syntax error will be reported by on-the-fly code inspection.

Insert 'self' when defining a method

If this check box is selected, the `self` parameter is inserted automatically after typing the opening brace, together with the closing brace and colon.

If this check box is cleared, only closing brace is generated automatically.

When necessary, use the suggested quick fix to insert `self`:



Insert type placeholders in the documentation comment stub

If this check box is selected, then the placeholders for parameters, types and return types will be generated:

```
"""  
  
:param a:  
:type a:  
:return:  
:rtype:  
"""
```

If this check box is not selected, then only the placeholders for parameters and return will be generated:

```
"""  
  
:param a:  
:return:  
"""
```

Insert 'type' and 'rtype' to the documentation comment stub

If this check box is selected, the documentation comment stub will contain `type` tag for each parameter, and `rtype` tag for the `return` statement.

Auto-insert closing `}}` and `%}` in Django templates

If this check box is selected, the closing characters will be automatically inserted after typing the opening ones.

AngularJS

Use this area to define the behavior of AngularJS:

- Auto-insert white space in the interpolation: If this check box is selected, a white space is automatically inserted between the braces: `{{ }}` .

If this check box is not selected, the white space is not inserted: `{{}}` .

SQL

Insert string concatenation on Enter. You may want to turn this option off, if the DBMS you are working with supports multiline string literals: Say, there is the following fragment for PostgreSQL `text` value `notes` :

```
SET notes = 'Lightest element'
```

and the cursor is in front of the word `element` .

If the option is on, and you press `Enter` , the fragment will change to:

```
SET notes = 'Lightest ' ||
            'element'
```

Otherwise, the fragment will change to:

```
SET notes = 'Lightest
element'
```

Qualify object on code completion. The selected option defines how the name of an object is inserted in the editor when using the code completion suggestion box.

- Always. The qualified object names are always used, i.e. `<schema_name>.<object_name>` .
- On collisions. The qualified object name is used only if the short name is ambiguous, e.g. when there is the object with the same name in more than one schema.
- Never. The unqualified object names are always used.

Colors and Fonts

File | Settings | Editor | Colors and Fonts for Windows and Linux

PyCharm | Preferences | Editor | Colors and Fonts for macOS

Ctrl+Alt+S



On this page:

- [Scheme](#)
- [Scheme settings](#)

Scheme

Use this section to select the Colors and Fonts scheme. PyCharm suggests several pre-defined schemes, one of them being the default.

ItemDescription

Scheme	From this drop-down list, select the Colors & Fonts scheme to use in your workspace.
	Click this button to reveal the submenu of the following choices: <ul style="list-style-type: none">- Duplicate: choose this command to save the currently selected Colors & Fonts settings as a new scheme.- Reset: choose this command to reset the selected color scheme to the initial defaults shipped with PyCharm.- Delete: choose this command to delete the current scheme. Note that the default color schemes cannot be deleted, and this command is available for copies only!

Scheme settings

Use the corresponding pages to change font type, colors and highlighting for the [supported languages](#), consoles, debugger, version control, differences viewer, file statuses, and scopes. Note that besides color settings, highlighting is determined by your [Inspection Profile](#). For example, if you do not want unused symbols to be highlighted, turn off the Unused Symbol inspection in the profile being used.

Choose specific nested page, and configure as required.

PageDescription

Font	Use this page to define the family of the font to be used in the editor, its size, and line spacing.
General	Use this page to customize the font type and colors for the editor textual components, specified in the list.
Language defaults	Use this page to customize the default settings that are common to all the supported languages, for the selected textual components, specified in the list, semantic highlighting that helps you tell the parameters at a glance, etc. Refer to these examples .
Console Colors	Use this page to define the colors to be used in the run or debug console (see Run Tool Window , Debug Tool Window , Console).
Console Font	Use this page to define the family of the font to be used in the run or debug console, its size, and line spacing (see Run Tool Window , Debug Tool Window , Console).
Custom	Use this page to customize the font type and colors in user-defined file types , in particular, for custom keywords highlighting .
VCS	Use this page to customize the colors of the left gutter markers and VCS annotations . To change color of any VCS component, click the corresponding color swatch to show the Color Picker dialog, and select the desired color.
File Status	Use this page to customize font type and colors to visually denote the current file status with regards to a VCS repository .
By Scope	Use this page to customize colors and highlighting for different scopes. To edit a scope, click the Edit Scopes button below the pane with a list of existing scopes. The Scopes dialog box opens.
Languages and PyCharm components	Use these pages to customize font type and colors for the corresponding file types and the PyCharm components.

Code Style

File | Settings | Editor | Code Style for Windows and Linux

PyCharm | Preferences | Editor | Code Style for macOS

Ctrl+Alt+S



On this page:

- [Scheme](#)
- [Line Separators](#)
- [Indents Detection](#)
- [Formatter Control](#)
 - [Formatting markers usage example](#)
- [EditorConfig](#)

Scheme

In this area, choose the code style scheme and change it as required. Code style scheme settings are automatically applied every time PyCharm generates, refactors, or reformats your code.

Code styles are defined at the project level and at the IDE level (global).

- At the **Project** level, settings are grouped under the Project scheme, which is predefined and is marked in bold. The **Project** style scheme is applied to the current project only.
You can copy the Project scheme to the IDE level, using the Copy to IDE... command.
- At the **IDE** level, settings are grouped under the predefined Default scheme (marked in bold), and any other scheme created by the user by the Duplicate command (marked as plain text). Global settings are used when the user doesn't want to keep code style settings with the project and share them.
You can copy the IDE scheme to the current project, using the Copy to Project... command.

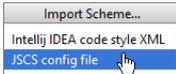
ItemDescription

Scheme From this drop-down list, select the scheme to be used. The predefined schemes are shown bold. The custom schemes, ones created as copies of the predefined schemes, are in plain text. The location where the scheme is stored is written next to each scheme, for example, the Default scheme is stored in the IDE, the Project scheme is stored in the project.



Click this button to invoke the drop-down list of commands to manage the schemes:

ItemDescriptionAvailable for

Copy to IDE...	Choose this command to copy the scheme settings to the IDE.	Project
Export...	Choose this command to export the selected scheme to an <code>xml</code> file in the selected location: 	Project and IDE
Import Scheme...	Choose this command to import the scheme of the selected type from the specified location: 	Project and IDE
Copy to Project...	Choose this command to copy the scheme settings to be stored with a project.	IDE
Duplicate...	Choose this command to create a copy of the selected scheme.	IDE
Reset	Choose this command to reset the default or bundled color scheme to the initial defaults shipped with PyCharm. This command becomes available only if some changes have been done.	IDE
Rename	Choose this command to change the name of the selected custom scheme. Press <code>Enter</code> to save changes, or <code>Escape</code> to cancel.	Custom schemes

Line Separators

PyCharm lets you configure line separator and indentation options for various languages. When [reformatting source code](#), PyCharm will apply the specified indentation behavior and skip the sections denoted with the special formatting off/on markers.

ItemDescription

Line Separator (for new files) Use this drop-down list to specify which line separator is to be used in files created by PyCharm. The available options are:

- System dependent - choose this option to use the default selection.
- Unix and macOS (`\n`) - choose this option to use the Unix and macOS line separator.
- Windows (`\r\n`) - choose this option to use the Windows line separator.
- Classic Mac (`\r`) - choose this option to use the Classic Mac line separator.

Refer to section [Configuring Line Separators](#).

Right Margin (columns)

In this text box, specify the number of columns to be used to display pages in the editor.

Wrap when typing reaches right margin Select this check box to ensure that edited text always fits in the specified right margin.

Indents Detection

Use this area to specify the default options for indentation.

ItemDescription

Detect and use existing file indents for editing	Select this check box for PyCharm to detect the existing indents in a file and use them for editing instead of the indents specified in the Code Style settings for the specific language.
Show notifications about detected indents	Select this check box to show a notification if PyCharm detects indents that are different from the ones specified in the Code Style settings for the specific language.

Formatter Control

In this area, specify the markers to limit code fragments that you want to exclude from reformatting, see more in [Reformatting Source Code](#). In the source code, formatting markers are written inside line comments, see [Commenting and Uncommenting Blocks of Code](#).

ItemDescription

Enable formatter markers in comments	<ul style="list-style-type: none">- If this check box is selected, fragments of code between line comments with the formatting markers will not be reformatted and will preserve the original formatting. After you select this check box, the fields below become available and you can specify the character strings to be treated as formatting markers.- If the check box is cleared, the formatting markers will be ignored and the code between the line comments with markers will be reformatted.
--------------------------------------	--

Markers

Formatter off:	In this text box, specify the character string that will indicate the beginning of a code fragment which you want to exclude from reformatting. Type a character string with the @ symbol in preposition or leave the predefined value @formatter:off .
Formatter on:	In this text box, specify the character string that will indicate the end of a code fragment which you want to exclude from reformatting. Type a character string with the @ symbol in preposition or leave the predefined value @formatter:on .
Regular expressions	Select this check box to use regular expressions instead of specifying the formatting markers explicitly. PyCharm matches formatter on/off markers using the regular expression specified instead of the exact string.

Formatting markers usage example

The original source code The code after reformatting

```
//@formatter:off
"scripts": {
  "post-install-cmd": [
    "php artisan optimize"
  ],
  "post-update-cmd": [
    "php artisan clear-compiled",
    "php artisan optimize"
  ],
  "post-create-project-cmd": [
    "php artisan key:generate"
  ]
},
//@formatter:on
```

When the formatting markers are disabled, the original formatting is broken:

```
//@formatter:off
"scripts": {
  "post-install-cmd": [
    "php artisan optimize"
  ],
  "post-update-cmd": [
    "php artisan clear-compiled",
    "php artisan optimize"
  ],
  "post-create-project-cmd": [
    "php artisan key:generate"
  ]
},
//@formatter:on
```

When the formatting markers are enabled, the original formatting is preserved:

```
//@formatter:off
"scripts": {
  "post-install-cmd": [
    "php artisan optimize"
  ],
  "post-update-cmd": [
    "php artisan clear-compiled",
    "php artisan optimize"
  ],
  "post-create-project-cmd": [
    "php artisan key:generate"
  ]
},
//@formatter:on
```

EditorConfig

In this area enable the support of the EditorConfig plugin.

ItemDescription

Enable EditorConfig support	Select this check box to enable the EditorConfig plugin support. In this case you can specify your own code style settings that override the IDE settings. However, if you decide to use IDE settings after creating the EditorConfig settings file then you need clear the Enable EditorConfig support check box. See also Configuring Code Style procedure.
Export	Click this button if you want to export the current IDE code style settings into the .editconfig file.



Use this dialog box to manage the set of code style schemes.

ItemDescription

Save As...	Click this button to create a copy of the currently selected scheme. This new scheme can be used for copying a scheme to project, and for export .
Delete	Click this button to remove the currently selected scheme. Note that the predefined schemes cannot be deleted.
Copy to Project	Click this button to copy the settings of the currently selected scheme to project. When the settings are copied, PyCharm suggests to switch to this scheme.
Import	Click this button to import PyCharm XML code style settings, or JSCS config file.
Export	Click this button to export to PyCharm XML code style settings file. The resulting XML file is exported to the specified location and has the specified name. This XML file is used by the Command Line Formatter .

Close	Click this button to close the Code Style Schemes dialog box.
-------	---

Note that any changes to the set of schemes (creating copies, or deleting unnecessary schemes) only take place on clicking Apply or OK buttons in the [Settings/Preferences dialog](#).

Use this page to configure formatting options for Python files. View the result in the Preview pane on the right.

On this page:

- [Tabs and Indents](#)
- [Spaces](#)
- [Wrapping and braces](#)
 - [Right Margin \(columns\)](#)
 - [Wrap on typing](#)
 - [Keep when reformatting](#)
- [Blank lines](#)
- [Imports](#)
- [Other](#)
- [Set from...](#)

Tip When a piece of code is selected in the editor, PyCharm suggests the quick fix [Adjust code style settings](#).
 – You can change the maximum line length for Python sources in [Code Style | Python | Wrapping and Braces | Right margin](#) of the editor settings.

Tabs and Indents

Item Description

Use tab character	<ul style="list-style-type: none"> – If this check box is selected, tab characters are used: <ul style="list-style-type: none"> – On pressing the <code>Tab</code> key – For indentation – For code reformatting – When the check box is cleared, PyCharm uses spaces instead of tabs.
Smart tabs	<ul style="list-style-type: none"> – If this check box is selected, the part of indentation defined by the nesting of code blocks, is made of the tabs and (if necessary) spaces, while the part of indentation defined by the alignment is made only of spaces. – If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment. <p>The Smart Tabs check box is available if the Use Tab Character check box is selected.</p>
Tab size	In this text box, specify the number of spaces included in a tab.
Indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.
Continuation indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted between the elements of an array, in expressions, method declarations and method calls.
Keep indents on empty lines	<p>If this check box is selected, then PyCharm will keep indents on the empty lines as if they contained some code.</p> <p>If this check box is not selected, PyCharm will delete the tab characters and spaces.</p>

Spaces

Use this tab to specify where you want spaces in your code. To have PyCharm automatically insert a space at a location, select the check box next to this location in the list. The results are displayed in the Preview pane.

Wrapping and braces

In this tab, customize the code style options, which PyCharm will apply on [reformatting the source code](#). The left-hand pane contains the list of exceptions (Keep when reformatting), and placement and alignment options for the various code constructs (lists, statements, operations, annotations, etc.) The right-hand pane shows preview.

Alignment takes precedence over indentation options.

Right Margin (columns)

Use Right Margin field to specify a margin space required on the right side of an element. If you select Default option then a value of the right margin from the [global settings](#) is used.

Wrap on typing

Use Wrap on typing settings to specify how the edited text is fitted in the specified Right margin. You can select one the following options:

- Default - in this case PyCharm uses the Wrap on typing option that is specified in the [global settings](#).
- Yes - in this case PyCharm uses the value specified in the Right Margin field.
- No - in this case this option is switched off and a line can exceed the value specified in the right margin.

Keep when reformatting

Use the check boxes to configure exceptions that PyCharm will make when reformatting the source code. For example, by default, the *Line breaks* check box is selected. If your code contains lines that are shorter than a standard convention, you can convert them by disabling the Line breaks check box before you [reformat the source code](#).

Blank lines

Use this tab to define where and how many blank lines you want PyCharm to retain and insert in your code after reformatting. For each type of location, specify the number of blank lines to be inserted. The results are displayed in the Preview pane.

ItemDescription

Keep Maximum Blank Lines In this area, specify the number of blank lines to be kept after reformatting in the specified locations.

Minimum Blank Lines In the text boxes in this area, specify the number of blank lines to be present in the specified locations.

Warning! These settings do not influence the number of blank lines before the first and after the last item.

Imports

This table lists actions to be performed when [imports are optimized](#).

ItemDescription

Sort import statements Select or clear this check box to enable or disable sorting imports within [individual import groups according to PEP 8](#). The following two check boxes affect the sorting order.

Sort imported names in "from" imports If this check box is selected, the imports in the `from ... import ...` statements are sorted alphabetically.

Not Selected
selected

```
from sys import version, path, modules
```

```
from sys import modules, path, version
```

Sort plain and 'from' imports separately within a group If this check box is not selected, the imports from the same module, regardless of their type, are grouped together, but so that `import` statements go first, and `from ... import ...` statements go next. If this check box is selected, the imports are at first sorted by their type (first `import`, next `from ... import ...`), and then alphabetically.

Not Selected
selected

```
import os
from os import getenv
import sys
from sys import path
```

```
import os
import sys
from os import getenv
from sys import path
```

Note The option Sort plain and 'from' imports separately within a group corresponds to the option `force_alphabetical_sort` of the utility `isort`.

Also, this sorting order corresponds to [Google Python Style Guide](#).

Join "from" imports with the same source If this check box is selected, the "from" imports of the same source are combined.

Not Selected
selected

```
from os.path import join
from os.path import relpath, abspath
from os.path import dirname, basename
```

```
from os.path import join, relpath, abspath, dirname, basename
```

Note The option Join "from" imports with the same source corresponds to the `combine_as` option of the utility `isort`.

Other

ItemDescription

Dict alignment From the drop-down list, select the type of `dict` alignment:

- Do not align: the `dict`'s elements in sequential lines will be not aligned.
- Align on colon: the `dict`'s elements in sequential lines will be aligned against the colon.
- Align on value: the `dict`'s elements in sequential lines will be aligned against the value.

Add line feed at the end of file Select this check box to add line feed character at the end of file.

Use continuation indent for arguments

Select this check box to use continuation indent (defined in the Tabs and Indents tab) for list of arguments. If this check box is not selected, then the indent value is used.

Set from...

Click this link to reveal the list of languages to be used as the base for the current language code style. So doing, only the settings that are applicable to the current language are taken. All the other settings are not affected.

This link appears in the upper-right corner of the language-specific code style page, when applicable.

Click Reset to discard changes and return to the initial set of code style settings.

Use this page to configure formatting options for CoffeeScript files. View the result in the Preview pane on the right.

On this page:

- [Tabs and Indents](#)
- [Spaces](#)
- [Wrapping and braces](#)
 - [Right Margin \(columns\)](#)
 - [Wrap on typing](#)
 - [Keep when reformatting](#)
 - [Wrapping options](#)
 - [Alignment options](#)
 - [Braces placement options](#)
- [Blank lines](#)
- [Other](#)
- [Set from...](#)

Tabs and Indents

ItemDescription

Use tab character	<ul style="list-style-type: none"> – If this check box is selected, tab characters are used: <ul style="list-style-type: none"> – On pressing the <code>Tab</code> key – For indentation – For code reformatting – When the check box is cleared, PyCharm uses spaces instead of tabs.
Smart tabs	<ul style="list-style-type: none"> – If this check box is selected, the part of indentation defined by the nesting of code blocks, is made of the tabs and (if necessary) spaces, while the part of indentation defined by the alignment is made only of spaces. – If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment. <p>The Smart Tabs check box is available if the Use Tab Character check box is selected.</p>
Tab size	In this text box, specify the number of spaces included in a tab.
Indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.
Continuation indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted between the elements of an array, in expressions, method declarations and method calls.
Keep indents on empty lines	<ul style="list-style-type: none"> If this check box is selected, then PyCharm will keep indents on the empty lines as if they contained some code. If this check box is not selected, PyCharm will delete the tab characters and spaces.

Spaces

Use this tab to specify where you want spaces in your code. To have PyCharm automatically insert a space at a location, select the check box next to this location in the list. The results are displayed in the Preview pane.

Wrapping and braces

In this tab, customize the code style options, which PyCharm will apply on [reformatting the source code](#). The left-hand pane contains the list of exceptions (Keep when reformatting), and placement and alignment options for the various code constructs (lists, statements, operations, annotations, etc.) The right-hand pane shows preview.

Alignment takes precedence over indentation options.

Right Margin (columns)

Use Right Margin field to specify a margin space required on the right side of an element. If you select Default option then a value of the right margin from the [global settings](#) is used.

Wrap on typing

Use Wrap on typing settings to specify how the edited text is fitted in the specified Right margin. You can select one the following options:

- Default - in this case PyCharm uses the Wrap on typing option that is specified in the [global settings](#).
- Yes - in this case PyCharm uses the value specified in the Right Margin field.
- No - in this case this option is switched off and a line can exceed the value specified in the right margin.

Keep when reformatting

Use the check boxes to configure exceptions that PyCharm will make when reformatting the source code. For example, by default, the *Line breaks* check box is selected. If your code contains lines that are shorter than a standard convention, you can convert them by disabling the Line breaks check box before you [reformat the source code](#).

Wrapping options

The wrapping style applies to the various code constructs, specified in the left-hand pane (for example, method call arguments, or assignment statements).

ItemDescription

Wrapping style	From this drop-down list, select the desired wrapping style: <ul style="list-style-type: none">– Do not wrap - when this option is selected, no special wrapping style is applied. With this option selected, the nested alignment and braces settings are ignored.– Wrap if long - select this option to have lines going beyond the right margin wrapped with proper indentation.– Wrap always - select this option to have all elements in lists wrapped so that there is one element per line with proper indentation.– Chop down if long - select this option to have elements in lists that go beyond the right margin wrapped so that there is one element per line with proper indentation.
----------------	--

Alignment options

ItemDescription

Align when multiline	If this check box is selected, a code construct starts at the same column on each next line. Otherwise, the position of a code construct is determined by the current indentation level.
<code><character(s)></code> on next line	Select this check box to have the specified character or characters moved to the next line when the lines are wrapped.
'else' on new line	Use this check box to have the corresponding statements or characters moved to the next line.
New line after <code><character></code>	Select this check box to have the code after the specified character moved to a new line.
Special else if treatment	If this check box is selected, <code>else if</code> statements are located in the same line. Otherwise, <code>else if</code> statements are moved to the next line to the corresponding indent level.
Indent case branches	If this check box is selected, the <code>case</code> statement is located at the corresponding indent level. Otherwise, <code>case</code> statement is placed at the same indent level with <code>switch</code> .

Braces placement options

ItemDescription

Braces placement style	Use this drop-down list to specify the position of the opening brace in class declarations, method declarations, and other types of declarations. The available options are: <ul style="list-style-type: none">– End of line - select this option to have the opening brace placed at the declaration line end.– Next line if wrapped - select this option to have the opening brace placed at the beginning of the line after the multiline declaration line.– Next line - select this option to have the opening brace placed at the beginning of the line after the declaration line.– Next line shifted - select this option to have the opening brace placed at the line after the declaration line being shifted to the corresponding indent level.– Next line each shifted - select this option to have the opening brace placed at the line after the declaration line being shifted to the corresponding indent level, and have the next line shifted to the next indent level as well.
Force braces	From this drop-down list, choose the braces introduction method for <code>if</code> , <code>for</code> , <code>while</code> , and <code>do () while</code> statements. The available options are: <ul style="list-style-type: none">– Do not force - select this option to suppress introducing braces automatically.– When multiline - select this option to have braces introduced automatically, if a statement occupies more than one line. Note that PyCharm analyzes the number of lines in the entire statement but not only its condition.– Always - select this check box to have braces always introduced automatically.

Blank lines

Use this tab to define where and how many blank lines you want PyCharm to retain and insert in your code after reformatting. For each type of location, specify the number of blank lines to be inserted. The results are displayed in the Preview pane.

ItemDescription

Keep Maximum Blank Lines	In this area, specify the number of blank lines to be kept after reformatting in the specified locations.
In code	Use this field to set the number of the blank lines.

Other

Item Description

Align object properties	From the drop-down list, select the type of objects' alignment: <ul style="list-style-type: none">– Do not align: the attributes in sequential lines will be not aligned.– On colon: the attributes in sequential lines will be aligned against the colon.– On value: the attributes in sequential lines will be aligned against the value.
Line comments at first column	Select this check box to place a line comment in the first column.

Set from...

Click this link to reveal the list of languages to be used as the base for the current language code style. So doing, only the settings that are applicable to the current language are taken. All the other settings are not affected.

This link appears in the upper-right corner of the language-specific code style page, when applicable.

Click Reset to discard changes and return to the initial set of code style settings.

Use this page to configure formatting options for CSS files. View the result in the Preview pane on the right.

On this page:

- [Tabs and Indents](#)
- [Other](#)
- [Set from...](#)

Tabs and Indents

ItemDescription

Use tab character

- If this check box is selected, tab characters are used:
 - On pressing the `Tab` key
 - For indentation
 - For code reformatting
- When the check box is cleared, PyCharm uses spaces instead of tabs.

Smart tabs

- If this check box is selected, the part of indentation defined by the nesting of code blocks, is made of the tabs and (if necessary) spaces, while the part of indentation defined by the alignment is made only of spaces.
- If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment.

The Smart Tabs check box is available if the Use Tab Character check box is selected.

Tab size In this text box, specify the number of spaces included in a tab.

Indent In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.

Continuation indent In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted between the elements of an array, in expressions, method declarations and method calls.

Keep indents on empty lines If this check box is selected, then PyCharm will keep indents on the empty lines as if they contained some code.

If this check box is not selected, PyCharm will delete the tab characters and spaces.

Other

In this tab, specify the alignment, braces and spaces options to be applied on reformatting.

ItemDescription

Braces placement Use this drop-down list to specify where PyCharm should place the opening braces of selectors. The available options are:

- At the end of line
- Next line

Align values Use this drop-down list to specify how PyCharm should align attributes and values. The available options are:

- Do not align: select this option to specify alignment on the first character of an attribute name.
- On value: select this option to specify alignment on the first character of the value of an attribute.
- On colon

Blank lines between blocks In this text box, specify the minimum number of sequential blank lines to be retained after reformatting.

Align closing brace with properties If this check box is selected, the closing brace of the selector will be placed under the list of properties.

If this check box is not selected, the closing brace of the selector will be placed under the selector.

Keep single-line blocks If this check box is selected, the blocks with a single property will be confined to one line.

If this check box is not selected, each property will be placed to its own line.

Spaces Select the check boxes in this area to add a space after the colon delimiting key and value, and before the opening brace of the selector.

HEX Colors Use this area to configure the hex color syntax. You can select from the following check options:

- Convert hex colors to - select this check box to configure the hex color letter case. You can choose Lower case or Upper case.
- Convert hex colors format to - select this check box to configure the hex color format length. You can choose Long format or Short format.

View changes in the Preview pane.

Set from...

Click this link to reveal the list of languages to be used as the base for the current language code style. So doing, only the settings that are applicable to the current language are taken. All the other settings are not affected.

This link appears in the upper-right corner of the language-specific code style page, when applicable.

Click [Reset](#) to discard changes and return to the initial set of code style settings.

Use this page to configure formatting options for Gherkin files. View the result in the Preview pane on the right.

On this page:

- [Tabs and Indents](#)
- [Set from...](#)

Tabs and Indents

ItemDescription

Use tab character

- If this check box is selected, tab characters are used:
 - On pressing the  key
 - For indentation
 - For code reformatting
- When the check box is cleared, PyCharm uses spaces instead of tabs.

Smart tabs

- If this check box is selected, the part of indentation defined by the nesting of code blocks, is made of the tabs and (if necessary) spaces, while the part of indentation defined by the alignment is made only of spaces.
- If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment.

The Smart Tabs check box is available if the Use Tab Character check box is selected.

Tab size In this text box, specify the number of spaces included in a tab.

Indent In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.

Continuation indent In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted between the elements of an array, in expressions, method declarations and method calls.

Keep indents on empty lines

- If this check box is selected, then PyCharm will keep indents on the empty lines as if they contained some code.
- If this check box is not selected, PyCharm will delete the tab characters and spaces.

Set from...

Click this link to reveal the list of languages to be used as the base for the current language code style. So doing, only the settings that are applicable to the current language are taken. All the other settings are not affected.

This link appears in the upper-right corner of the language-specific code style page, when applicable.

Click Reset to discard changes and return to the initial set of code style settings.

Use this page to configure formatting options for Haml files. View the result in the Preview pane on the right.

On this page:

- [Tabs and Indents](#)
- [Set from...](#)

Tabs and Indents

ItemDescription

Use tab character	<ul style="list-style-type: none">– If this check box is selected, tab characters are used:<ul style="list-style-type: none">– On pressing the <code>Tab</code> key– For indentation– For code reformatting– When the check box is cleared, PyCharm uses spaces instead of tabs.
Smart tabs	<ul style="list-style-type: none">– If this check box is selected, the part of indentation defined by the nesting of code blocks, is made of the tabs and (if necessary) spaces, while the part of indentation defined by the alignment is made only of spaces.– If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment. <p>The Smart Tabs check box is available if the Use Tab Character check box is selected.</p>
Tab size	In this text box, specify the number of spaces included in a tab.
Indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.
Continuation indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted between the elements of an array, in expressions, method declarations and method calls.
Keep indents on empty lines	<p>If this check box is selected, then PyCharm will keep indents on the empty lines as if they contained some code.</p> <p>If this check box is not selected, PyCharm will delete the tab characters and spaces.</p>

Set from...

Click this link to reveal the list of languages to be used as the base for the current language code style. So doing, only the settings that are applicable to the current language are taken. All the other settings are not affected.

This link appears in the upper-right corner of the language-specific code style page, when applicable.

Click Reset to discard changes and return to the initial set of code style settings.

Use this page to configure formatting options for HTML files. View the result in the Preview pane on the right.

On this page:

- [Tabs and Indents](#)
- [Other](#)
- [Set from...](#)

Tabs and Indents

ItemDescription

Use tab character	<ul style="list-style-type: none"> – If this check box is selected, tab characters are used: <ul style="list-style-type: none"> – On pressing the <code>Tab</code> key – For indentation – For code reformatting – When the check box is cleared, PyCharm uses spaces instead of tabs.
-------------------	--

Smart tabs	<ul style="list-style-type: none"> – If this check box is selected, the part of indentation defined by the nesting of code blocks, is made of the tabs and (if necessary) spaces, while the part of indentation defined by the alignment is made only of spaces. – If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment. <p>The Smart Tabs check box is available if the Use Tab Character check box is selected.</p>
------------	---

Tab size	In this text box, specify the number of spaces included in a tab.
----------	---

Indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.
--------	---

Continuation indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted between the elements of an array, in expressions, method declarations and method calls.
---------------------	--

Keep indents on empty lines	<p>If this check box is selected, then PyCharm will keep indents on the empty lines as if they contained some code.</p> <p>If this check box is not selected, PyCharm will delete the tab characters and spaces.</p>
-----------------------------	--

Other

ItemDescription

Right Margin	Use these settings to specify a margin space required on the right side of an element. If you select Default option then a value of the right margin from the global settings will be used.
--------------	---

Wrap on typing	<p>Use these settings to specify how the edited text is fitted in the specified Right margin. You can select one of the following options:</p> <ul style="list-style-type: none"> – Default - in this case PyCharm uses the Wrap on typing option that is specified in the global settings. – Yes - in this case the value in the specified right margin is used. – No - in this case this option is switched off and a line can exceed the number that is specified in the right margin.
----------------	--

Keep line breaks	Select this check box to have PyCharm honor line breaks when reviewing HTML files in the editor.
------------------	--

Keep line breaks in text	Select this check box to have PyCharm honor line breaks in attributes (for example, lengthy descriptions) when reviewing HTML files in the editor.
--------------------------	--

Keep blank lines	In this text box, specify the minimum number of sequential blank lines to be retained after reformatting.
------------------	---

Wrap attributes	<p>Use this drop-down list to determine how attribute lines should be wrapped. The available options are:</p> <ul style="list-style-type: none"> – Do not wrap - if this option is selected, no special wrapping style is applied to the code. – Wrap if long - select this option to have lines going beyond the right margin wrapped with proper indentation. – Chop down if long - select this option to have elements in lists that go beyond the right margin wrapped to give one element per line with proper indentation. – Wrap always - select this option to have all elements in lists wrapped to give one element per line with proper indentation.
-----------------	---

Wrap text	Select this check box to have long lines wrapped according to the code style settings.
-----------	--

Align attributes	Select this check box to have attributes in sequential lines aligned.
------------------	---

Align text	Select this check box to have PyCharm align the text that occupies several lines within a tag.
------------	--

Keep white spaces	Select this check box to suppress replacing actual white spaces with tabs.
-------------------	--

Spaces	<p>In this area, define the use of spaces for attributes and tag names.</p> <ul style="list-style-type: none"> – Around "=" in attribute - select this check box to have spaces added around the "=" symbol in attributes. – After tag name - select this check box to have spaces added after tag names.
--------	---

– In empty tag - select this check box to have spaces added in empty tags.

Insert new line before	This display field shows a list of tags before which a new line should be inserted. Use the button  next to the field or press Shift+Enter to open the Insert New Line Before Tags dialog box, where you can edit the list of tags.
Remove new line before	This display field shows a list of tags before which a break line should be removed. Use the button  next to the field or press Shift+Enter to open the Remove Line Breaks Before Tags dialog box, where you can edit the list of tags.
Do not indent children of	This display field shows a list of tags whose children should not be indented. Use the button  next to the field or press Shift+Enter to open the Do Not Indent Children Of dialog box, where you can edit the list of tags.
Or if tag size more than	In this text box, specify the minimum length of a tag in lines starting from which its children are not indented.
Inline elements	This display field shows a list of tags that are presented in the source code in the same line with the other tags. If a tag is removed from the list, the editor automatically moves it to a new line, when you add such tag to the source code. Use the button  next to the field or press Shift+Enter to open the Inline Elements dialog box, where you can edit the list of tags.
Keep white spaces inside	This display field shows a list of tags inside which you want the editor to preserve white spaces <i>as is</i> , without any changes. Use the button  next to the field or press Shift+Enter to open the Keep Whitespaces Inside dialog box, where you can edit the list of tags.
Don't break if inline content	This display field shows a list of tags that are not to be wrapped if their content is inlined. Use the button  next to the field or press Shift+Enter to open the Don't Wrap If Inline Content Only dialog box, where you can edit the list of tags.
Generated quote marks	Choose the style of the quote marks (double, single, or none) to be automatically inserted around HTML attributes on typing <input type="text"/> . This is important when HTML is inserted dynamically using JavaScript and you want to consistently use double-quote pairs for JavaScript strings and single-quote pairs for HTML to prevent problems, for example, when copying and pasting.
Enforce on format	If this check box is selected, then on code reformatting the previously generated quote marks will be replaced (for example, double quotes with single quotes).

Set from...

Click this link to reveal the list of languages to be used as the base for the current language code style. So doing, only the settings that are applicable to the current language are taken. All the other settings are not affected.

This link appears in the upper-right corner of the language-specific code style page, when applicable.

Click Reset to discard changes and return to the initial set of code style settings.

Use this page to configure formatting options for JSON files. View the result in the Preview pane on the right.

On this page:

- [Tabs and Indents](#)
- [Spaces](#)
- [Blank lines](#)
- [Wrapping and braces](#)
 - [Right Margin \(columns\)](#)
 - [Wrap on typing](#)
 - [Keep when reformatting](#)
 - [Wrapping options](#)
 - [Alignment options](#)
- [Other](#)

Tabs and Indents

ItemDescription

Use tab character	<ul style="list-style-type: none"> – If this check box is selected, tab characters are used: <ul style="list-style-type: none"> – On pressing the <code>Tab</code> key – For indentation – For code reformatting – When the check box is cleared, PyCharm uses spaces instead of tabs.
Smart tabs	<ul style="list-style-type: none"> – If this check box is selected, the part of indentation defined by the nesting of code blocks, is made of the tabs and (if necessary) spaces, while the part of indentation defined by the alignment is made only of spaces. – If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment. <p>The Smart Tabs check box is available if the Use Tab Character check box is selected.</p>
Tab size	In this text box, specify the number of spaces included in a tab.
Indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.
Continuation indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted between the elements of an array, in expressions, method declarations and method calls.
Keep indents on empty lines	<p>If this check box is selected, then PyCharm will keep indents on the empty lines as if they contained some code.</p> <p>If this check box is not selected, PyCharm will delete the tab characters and spaces.</p>
Indent chained methods	<p>In declarations of functions, the second and further methods in a chain are displayed on a separate line.</p> <ul style="list-style-type: none"> – When the check box is selected, the second and further methods in a chain are aligned with the first call. – When the check box is cleared, the second and further methods in a chain are aligned with the object on which they are invoked.
Indent all chained calls in a group	The check box is available only when the Indent chained methods check box is selected.

Spaces

Use this tab to specify where you want spaces in your code. To have PyCharm automatically insert a space at a location, select the check box next to this location in the list. The results are displayed in the Preview pane.

Blank lines

Use this tab to define where and how many blank lines you want PyCharm to retain and insert in your code after reformatting. For each type of location, specify the number of blank lines to be inserted. The results are displayed in the Preview pane.

ItemDescription

Keep Maximum Blank Lines	In this area, specify the number of blank lines to be kept after reformatting in the specified locations.
In code	Use this field to set the number of the blank lines.

Wrapping and braces

In this tab, customize the code style options, which PyCharm will apply on [reformatting the source code](#). The left-hand pane contains the list of exceptions (Keep when reformatting), and placement and alignment options for the various code constructs (lists, statements, operations, annotations, etc.) The right-hand pane shows preview.

Alignment takes precedence over indentation options.

Right Margin (columns)

Use Right Margin field to specify a margin space required on the right side of an element. If you select Default option then a value of the right margin from the [global settings](#) is used.

Wrap on typing

Use Wrap on typing settings to specify how the edited text is fitted in the specified Right margin. You can select one the following options:

- Default - in this case PyCharm uses the Wrap on typing option that is specified in the [global settings](#).
- Yes - in this case PyCharm uses the value specified in the Right Margin field.
- No - in this case this option is switched off and a line can exceed the value specified in the right margin.

Keep when reformatting

Use the check boxes to configure exceptions that PyCharm will make when reformatting the source code. For example, by default, the *Line breaks* check box is selected. If your code contains lines that are shorter than a standard convention, you can convert them by disabling the Line breaks check box before you [reformat the source code](#).

Wrapping options

The wrapping style applies to the various code constructs, specified in the left-hand pane (for example, method call arguments, or assignment statements).

ItemDescription

Wrapping style	<p>From this drop-down list, select the desired wrapping style:</p> <ul style="list-style-type: none">- Do not wrap - when this option is selected, no special wrapping style is applied. With this option selected, the nested alignment and braces settings are ignored.- Wrap if long - select this option to have lines going beyond the right margin wrapped with proper indentation.- Wrap always - select this option to have all elements in lists wrapped so that there is one element per line with proper indentation.- Chop down if long - select this option to have elements in lists that go beyond the right margin wrapped so that there is one element per line with proper indentation.
----------------	---

Ensure right margin is not exceeded	If this check box is selected, the formatter will do its best to avoid having document lines exceeding the right margin. This option takes precedence over the Do not wrap wrapping style.
-------------------------------------	--

Alignment options

ItemDescription

Align when multiline	If this check box is selected, a code construct starts at the same column on each next line. Otherwise, the position of a code construct is determined by the current indentation level.
----------------------	--

<character(s)> on next line	Select this check box to have the specified character or characters moved to the next line when the lines are wrapped.
-----------------------------	--

'else' on new line	Use this check box to have the corresponding statements or characters moved to the next line.
--------------------	---

New line after <character>	Select this check box to have the code after the specified character moved to a new line.
----------------------------	---

Special else if treatment	If this check box is selected, <code>else if</code> statements are located in the same line. Otherwise, <code>else if</code> statements are moved to the next line to the corresponding indent level.
---------------------------	---

Indent case branches	If this check box is selected, the <code>case</code> statement is located at the corresponding indent level. Otherwise, <code>case</code> statement is placed at the same indent level with <code>switch</code> .
----------------------	---

Other

Item Description

Align object properties	<p>From the drop-down list, select the type of objects' alignment:</p> <ul style="list-style-type: none">- Do not align: the attributes in sequential lines will be not aligned.- On colon: the attributes in sequential lines will be aligned against the colon.- On value: the attributes in sequential lines will be aligned against the value.
-------------------------	--

Use this page to configure formatting options for Less files. View the result in the Preview pane on the right.

On this page:

- [Tabs and Indents](#)
- [Set from...](#)

Tabs and Indents

ItemDescription

Use tab character	<ul style="list-style-type: none">– If this check box is selected, tab characters are used:<ul style="list-style-type: none">– On pressing the <code>Tab</code> key– For indentation– For code reformatting– When the check box is cleared, PyCharm uses spaces instead of tabs.
Smart tabs	<ul style="list-style-type: none">– If this check box is selected, the part of indentation defined by the nesting of code blocks, is made of the tabs and (if necessary) spaces, while the part of indentation defined by the alignment is made only of spaces.– If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment. <p>The Smart Tabs check box is available if the Use Tab Character check box is selected.</p>
Tab size	In this text box, specify the number of spaces included in a tab.
Indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.
Continuation indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted between the elements of an array, in expressions, method declarations and method calls.
Keep indents on empty lines	<p>If this check box is selected, then PyCharm will keep indents on the empty lines as if they contained some code.</p> <p>If this check box is not selected, PyCharm will delete the tab characters and spaces.</p>

Set from...

Click this link to reveal the list of languages to be used as the base for the current language code style. So doing, only the settings that are applicable to the current language are taken. All the other settings are not affected.

This link appears in the upper-right corner of the language-specific code style page, when applicable.

Click Reset to discard changes and return to the initial set of code style settings.

Use this page to configure formatting options for PostCSS files. View the result in the Preview pane on the right.

On this page:

- [Tabs and Indents](#)
- [Set from...](#)

The page is available only when the [PostCSS](#) repository plugin is installed and enabled. The plugin is not bundled with PyCharm, but it can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).

Tabs and Indents

ItemDescription

Use tab character

- If this check box is selected, tab characters are used:
 - On pressing the  key
 - For indentation
 - For code reformatting
- When the check box is cleared, PyCharm uses spaces instead of tabs.

Smart tabs

- If this check box is selected, the part of indentation defined by the nesting of code blocks, is made of the tabs and (if necessary) spaces, while the part of indentation defined by the alignment is made only of spaces.
- If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment.

The Smart Tabs check box is available if the Use Tab Character check box is selected.

Tab size In this text box, specify the number of spaces included in a tab.

Indent In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.

Continuation indent In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted between the elements of an array, in expressions, method declarations and method calls.

Keep indents on empty lines

If this check box is selected, then PyCharm will keep indents on the empty lines as if they contained some code.

If this check box is not selected, PyCharm will delete the tab characters and spaces.

Set from...

Click this link to reveal the list of languages to be used as the base for the current language code style. So doing, only the settings that are applicable to the current language are taken. All the other settings are not affected.

This link appears in the upper-right corner of the language-specific code style page, when applicable.

Click [Reset](#) to discard changes and return to the initial set of code style settings.

Use this page to configure formatting options for JavaScript files. View the result in the Preview pane on the right.

On this page:

- [Tabs and Indents](#)
- [Spaces](#)
- [Wrapping and braces](#)
 - [Right Margin \(columns\)](#)
 - [Wrap on typing](#)
 - [Keep when reformatting](#)
 - [Wrapping options](#)
 - [Alignment options](#)
 - [Braces placement options](#)
- [Blank lines](#)
- [Punctuation](#)
- [Other](#)
- [Imports](#)
- [Set from...](#)

Tabs and Indents

ItemDescription

Use tab character	<ul style="list-style-type: none"> – If this check box is selected, tab characters are used: <ul style="list-style-type: none"> – On pressing the <code>Tab</code> key – For indentation – For code reformatting – When the check box is cleared, PyCharm uses spaces instead of tabs.
Smart tabs	<ul style="list-style-type: none"> – If this check box is selected, the part of indentation defined by the nesting of code blocks, is made of the tabs and (if necessary) spaces, while the part of indentation defined by the alignment is made only of spaces. – If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment. <p>The Smart Tabs check box is available if the Use Tab Character check box is selected.</p>
Tab size	In this text box, specify the number of spaces included in a tab.
Indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.
Continuation indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted between the elements of an array, in expressions, method declarations and method calls.
Indent chained methods	<p>In declarations of functions, the second and further methods in a chain are displayed on a separate line.</p> <ul style="list-style-type: none"> – When the check box is selected, the second and further methods in a chain are aligned with the first call. – When the check box is cleared, the second and further methods in a chain are aligned with the object on which they are invoked.
Indent all chained calls in a group	The check box is available only when the Indent chained methods check box is selected.

Spaces

Use this tab to specify where you want spaces in your code. To have PyCharm automatically insert a space at a location, select the check box next to this location in the list. The results are displayed in the Preview pane.

Wrapping and braces

In this tab, customize the code style options, which PyCharm will apply on [reformatting the source code](#). The left-hand pane contains the list of exceptions (Keep when reformatting), and placement and alignment options for the various code constructs (lists, statements, operations, annotations, etc.) The right-hand pane shows preview.

Alignment takes precedence over indentation options.

Right Margin (columns)

Use Right Margin field to specify a margin space required on the right side of an element. If you select Default option then a value of the right margin from the [global settings](#) is used.

Wrap on typing

Use Wrap on typing settings to specify how the edited text is fitted in the specified Right margin. You can select one the following options:

Default - in this case PyCharm uses the Wrap on typing option that is specified in the [global settings](#)

– Default - in this case PyCharm uses the wrap on typing option that is specified in the [global settings](#).

– Yes - in this case PyCharm uses the value specified in the Right Margin field.

– No - in this case this option is switched off and a line can exceed the value specified in the right margin.

Keep when reformatting

Use the check boxes to configure exceptions that PyCharm will make when reformatting the source code. For example, by default, the *Line breaks* check box is selected. If your code contains lines that are shorter than a standard convention, you can convert them by disabling the Line breaks check box before you [reformat the source code](#).

Wrapping options

The wrapping style applies to the various code constructs, specified in the left-hand pane (for example, method call arguments, or assignment statements).

ItemDescription

Wrapping style	From this drop-down list, select the desired wrapping style: <ul style="list-style-type: none">– Do not wrap - when this option is selected, no special wrapping style is applied. With this option selected, the nested alignment and braces settings are ignored.– Wrap if long - select this option to have lines going beyond the right margin wrapped with proper indentation.– Wrap always - select this option to have all elements in lists wrapped so that there is one element per line with proper indentation.– Chop down if long - select this option to have elements in lists that go beyond the right margin wrapped so that there is one element per line with proper indentation.
----------------	--

Alignment options

ItemDescription

Align when multiline	If this check box is selected, a code construct starts at the same column on each next line. Otherwise, the position of a code construct is determined by the current indentation level.
<code><character(s)></code> on next line	Select this check box to have the specified character or characters moved to the next line when the lines are wrapped.
'else' on new line	Use this check box to have the corresponding statements or characters moved to the next line.
New line after <code><character></code>	Select this check box to have the code after the specified character moved to a new line.
Special else if treatment	If this check box is selected, <code>else if</code> statements are located in the same line. Otherwise, <code>else if</code> statements are moved to the next line to the corresponding indent level.
Indent case branches	If this check box is selected, the <code>case</code> statement is located at the corresponding indent level. Otherwise, <code>case</code> statement is placed at the same indent level with <code>switch</code> .

Braces placement options

ItemDescription

Braces placement style	Use this drop-down list to specify the position of the opening brace in class declarations, method declarations, and other types of declarations. The available options are: <ul style="list-style-type: none">– End of line - select this option to have the opening brace placed at the declaration line end.– Next line if wrapped - select this option to have the opening brace placed at the beginning of the line after the multiline declaration line.– Next line - select this option to have the opening brace placed at the beginning of the line after the declaration line.– Next line shifted - select this option to have the opening brace placed at the line after the declaration line being shifted to the corresponding indent level.– Next line each shifted - select this option to have the opening brace placed at the line after the declaration line being shifted to the corresponding indent level, and have the next line shifted to the next indent level as well.
Force braces	From this drop-down list, choose the braces introduction method for <code>if</code> , <code>for</code> , <code>while</code> , and <code>do () while</code> statements. The available options are: <ul style="list-style-type: none">– Do not force - select this option to suppress introducing braces automatically.– When multiline - select this option to have braces introduced automatically, if a statement occupies more than one line. Note that PyCharm analyzes the number of lines in the entire statement but not only its condition.– Always - select this check box to have braces always introduced automatically.

Blank lines

Use this tab to define where and how many blank lines you want PyCharm to retain and insert in your code after reformatting. For each type of location, specify the number of blank lines to be inserted. The results are displayed in the Preview pane.

ItemDescription

Keep Maximum Blank Lines	In this area, specify the number of blank lines to be kept after reformatting in the specified locations.
In code	Use this field to set the number of the blank lines.

Punctuation

Use the drop-down lists in this tab to form directives in automatic insertion of terminating semicolons, single and double quotes, and trailing commas.

ItemDescription

Semicolon to terminate statements	<ul style="list-style-type: none">– Use semicolon to terminate statements in new code– Use semicolon to terminate statements always
-----------------------------------	--

- Don't use semicolon to terminate statements in new code
- Don't use semicolon to terminate statements always

- Quotes
- Use double quotes in new code
 - Use double quotes always
 - Use single quotes in new code
 - Use single quotes always

- Trailing comma
- Use this drop-down list to configure whether you want to use [trailing commas](#) in objects, arrays, and for the parameters in method definitions and calls. The available options are:
- Keep
 - Remove
 - Add when multiline

Other

Item Description

Formatting options

- Align object properties
- From the drop-down list, select the type of objects' alignment:
- Do not align: the attributes in sequential lines will be not aligned.
 - On colon: the attributes in sequential lines will be aligned against the colon.
 - On value: the attributes in sequential lines will be aligned against the value.

- Align 'var' statements and assignments
- Click one of the following radio-buttons to choose the way equality signs are aligned:
- Do not align: the equality signs are not aligned.
 - Align multiline 'var' statements: the multiline `var` statements are aligned against the equality signs by inserting additional spaces.
 - Align multiple 'var' statements and assignments: the multiple `var` statements are aligned against the equality signs by inserting additional spaces.

Coding style

- Use semicolon to terminate statements
- Select this check box to have statements terminated with a semicolon.

Comment code

- Line comment at first column
- Select this check box to have a line comment start at first column.
- Add a space at comment start
- If this check box is selected, a space will be inserted between a line comment character and the first character of a commented line. So doing, the option Line comment at first column must be unselected.
- Align C-style comments
- If this check box is selected, the multiline C-style comments are aligned against the asterisk. Otherwise, the multiline comments are not aligned.

Imports

ItemDescription

- Merge imports for members from the same module
- When this check box is selected, imported symbols from the same module are listed in one `import` statement with a comma as separator. The members are listed in the order in which they are imported. To have them listed alphabetically, select the Sort imported members check box and run Code | Optimize Imports.
 - When this check box is cleared, for each imported symbol a separate `import` statement is generated.

- Use paths relative to the project, resource or sources roots
- This option is applied during automatic generation of import statements in JavaScript code.
- When this check box is selected, PyCharm suggests paths relative to the project root, `resource` root, or `sources` root.
 - By default, this check box is cleared and PyCharm suggests paths relative to the current file.

- Sort imported members
- When this check box is selected, PyCharm lists the imported members in merged `import` statements alphabetically. Note that the members are listed comma-separated in the order they are imported and re-sorted only when you run Code | Optimize Imports.
 - When this check box is cleared, the members in merged `import` statements are always listed comma-separated in the order they are imported.

- Sort imports by modules
- When this check box is selected, `import` statements are re-sorted alphabetically by the module names when you run Code | Optimize Imports.
 - When this check box is cleared, `import` statements are always shown in the order they are generated and this order is not changed after you run Code | Optimize Imports.

Set from...

Click this link to choose the base for the current language default code style from the pop-up list, that appears. The list contains two options:

- Language: choose this option to inherit the coding style settings from another language. Select the source language from the list, that opens. So doing, only the settings that are applicable to the current language are taken. All the other settings are not affected.
- Predefined code style: choose this option to use the coding standards defined for a specific framework. Select one of the following frameworks from the list:
 - [JavaScript Standard Style](#)
 - [Google JavaScript Style Guide](#)

This link appears in the upper-right corner of the language-specific code style page, when applicable.

Click Reset to discard changes and return to the initial set of code style settings.

Use this page to configure formatting options for Pug(Jade) files. View the result in the Preview pane on the right.

On this page:

- [Tabs and Indents](#)
- [Set from...](#)

Before you start working with Pug(Jade), make sure that Pug(ex-Jade) plugin is [installed and enabled](#). The plugin is not bundled with PyCharm.

Tabs and Indents

ItemDescription

- | | |
|-------------------|---|
| Use tab character | <ul style="list-style-type: none">– If this check box is selected, tab characters are used:<ul style="list-style-type: none">– On pressing the <code>Tab</code> key– For indentation– For code reformatting– When the check box is cleared, PyCharm uses spaces instead of tabs. |
|-------------------|---|
-

Indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.
--------	---

Set from...

Click this link to reveal the list of languages to be used as the base for the current language code style. So doing, only the settings that are applicable to the current language are taken. All the other settings are not affected.

This link appears in the upper-right corner of the language-specific code style page, when applicable.

Click Reset to discard changes and return to the initial set of code style settings.

Use this page to configure formatting options for Puppet files. View the result in the Preview pane on the right.

On this page:

- [Tabs and Indents](#)
- [Set from...](#)

Tabs and Indents

ItemDescription

Use tab character	<ul style="list-style-type: none">– If this check box is selected, tab characters are used:<ul style="list-style-type: none">– On pressing the <code>Tab</code> key– For indentation– For code reformatting– When the check box is cleared, PyCharm uses spaces instead of tabs.
Smart tabs	<ul style="list-style-type: none">– If this check box is selected, the part of indentation defined by the nesting of code blocks, is made of the tabs and (if necessary) spaces, while the part of indentation defined by the alignment is made only of spaces.– If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment. <p>The Smart Tabs check box is available if the Use Tab Character check box is selected.</p>
Tab size	In this text box, specify the number of spaces included in a tab.
Indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.
Continuation indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted between the elements of an array, in expressions, method declarations and method calls.
Keep indents on empty lines	<p>If this check box is selected, then PyCharm will keep indents on the empty lines as if they contained some code.</p> <p>If this check box is not selected, PyCharm will delete the tab characters and spaces.</p>

Set from...

Click this link to reveal the list of languages to be used as the base for the current language code style. So doing, only the settings that are applicable to the current language are taken. All the other settings are not affected.

This link appears in the upper-right corner of the language-specific code style page, when applicable.

Click Reset to discard changes and return to the initial set of code style settings.

Use this page to configure formatting options for Sass files. View the result in the Preview pane on the right.

On this page:

- [Tabs and Indents](#)
- [Set from...](#)

Tabs and Indents

ItemDescription

Use tab character

- If this check box is selected, tab characters are used:
 - On pressing the `Tab` key
 - For indentation
 - For code reformatting
- When the check box is cleared, PyCharm uses spaces instead of tabs.

Smart tabs

- If this check box is selected, the part of indentation defined by the nesting of code blocks, is made of the tabs and (if necessary) spaces, while the part of indentation defined by the alignment is made only of spaces.
- If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment.

The Smart Tabs check box is available if the Use Tab Character check box is selected.

Tab size In this text box, specify the number of spaces included in a tab.

Indent In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.

Continuation indent In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted between the elements of an array, in expressions, method declarations and method calls.

Keep indents on empty lines

- If this check box is selected, then PyCharm will keep indents on the empty lines as if they contained some code.
- If this check box is not selected, PyCharm will delete the tab characters and spaces.

Set from...

Click this link to reveal the list of languages to be used as the base for the current language code style. So doing, only the settings that are applicable to the current language are taken. All the other settings are not affected.

This link appears in the upper-right corner of the language-specific code style page, when applicable.

Click Reset to discard changes and return to the initial set of code style settings.

Use this page to configure formatting options for SCSS files. View the result in the Preview pane on the right.

On this page:

- [Tabs and Indents](#)
- [Set from...](#)

Tabs and Indents

ItemDescription

Use tab character

- If this check box is selected, tab characters are used:
 - On pressing the `Tab` key
 - For indentation
 - For code reformatting
- When the check box is cleared, PyCharm uses spaces instead of tabs.

Smart tabs

- If this check box is selected, the part of indentation defined by the nesting of code blocks, is made of the tabs and (if necessary) spaces, while the part of indentation defined by the alignment is made only of spaces.
- If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment.

The Smart Tabs check box is available if the Use Tab Character check box is selected.

Tab size

In this text box, specify the number of spaces included in a tab.

Indent

In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.

Continuation indent

In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted between the elements of an array, in expressions, method declarations and method calls.

Keep indents on empty lines

If this check box is selected, then PyCharm will keep indents on the empty lines as if they contained some code.

If this check box is not selected, PyCharm will delete the tab characters and spaces.

Set from...

Click this link to reveal the list of languages to be used as the base for the current language code style. So doing, only the settings that are applicable to the current language are taken. All the other settings are not affected.

This link appears in the upper-right corner of the language-specific code style page, when applicable.

Click Reset to discard changes and return to the initial set of code style settings.

Use this page to configure formatting options for SQL files. View the result in the Preview pane on the right.

On this page:

- [General](#)
- [Tabs and Indents](#)
- [Spaces](#)
- [Wrapping and Braces](#)
- [Blank Lines](#)
- [Code Generation](#)
- [Set from...](#)

General

Use this tab to specify certain formatting behaviors in your code. The results are displayed in the Preview pane.

ItemDescription

Word Case	In this node, specify whether the keywords, identifiers, or quoted identifiers case should change automatically. Click the right-hand column, and check the desired behavior on the menu to turn it on. The possible options are: <ul style="list-style-type: none"> – To upper: the elements are automatically converted to the upper case. – To lower: The elements are automatically converted to lower case. – Do not change: The case of elements is left as is.
Identifier quotations	Click the right-hand column, and check the desired behavior on the menu to turn it on. The possible options are: <ul style="list-style-type: none"> – Quote: the identifiers are automatically quoted. – Unquote: The identifiers are automatically unquoted. – Do not change: The identifiers are left as is.
New line before/after	Use these nodes to define how many blank lines you want PyCharm to retain and insert in your code after reformatting. For each type of SQL element, select or clear the check box to the right. The results are highlighted in the Preview pane.
New line around semicolon	Use this check box to manage line delimiters around a semicolon.
Alignment	Select the check boxes next to the desired elements of source code to have them automatically aligned.

Tabs and Indents

ItemDescription

Use tab character	<ul style="list-style-type: none"> – If this check box is selected, tab characters are used: <ul style="list-style-type: none"> – On pressing the  key – For indentation – For code reformatting – When the check box is cleared, PyCharm uses spaces instead of tabs.
Smart tabs	<ul style="list-style-type: none"> – If this check box is selected, the part of indentation defined by the nesting of code blocks, is made of the tabs and (if necessary) spaces, while the part of indentation defined by the alignment is made only of spaces. – If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment. <p>The Smart Tabs check box is available if the Use Tab Character check box is selected.</p>
Tab size	In this text box, specify the number of spaces included in a tab.
Indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.
Continuation indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted between the elements of an array, in expressions, method declarations and method calls.
Keep indents on empty lines	<p>If this check box is selected, then PyCharm will keep indents on the empty lines as if they contained some code.</p> <p>If this check box is not selected, PyCharm will delete the tab characters and spaces.</p>

Spaces

Use this tab to specify where you want spaces in your code. To have PyCharm automatically insert a space at a location, select the check box next to this location in the list. The results are displayed in the Preview pane.

Wrapping and Braces

In this tab, customize the code style options, which PyCharm will apply when [reformatting the source code](#). The left-hand pane contains the list of exceptions (Keep when reformatting), and placement and alignment options for the various code constructs (lists, statements, operations, annotations, etc.). The right-hand pane shows preview.

Alignment takes precedence over [indentation options](#).

ItemDescription

Keep When Reformatting	Use the check boxes in this node to configure exceptions that PyCharm will make when reformatting the source code. For example, by default, the <i>Line brakes</i> check box is selected. If your code contains lines that are shorter than a standard convention, you can convert them by disabling the <i>Line brakes</i> check box before you reformat the source code .
Wrap inside	Use this node to define wrapping style for the lengthy elements. You can use different wrapping style options that are available from the drop-down list in the left-hand pane.
Values expression	Use this option to define wrapping style for values expressions. You can use different wrapping style options that are available from the drop-down list in the left-hand pane.
Wrapping Style Options	The wrapping style applies to the various code constructs, specified in the left-hand pane (for example, expressions, or assignment statements). From the drop-down list, select the desired wrapping style: <ul style="list-style-type: none">– Do not wrap - when this option is selected, no special wrapping style is applied. <div style="background-color: #ffff00; padding: 2px;">Note If this option is selected, the nested alignment and braces settings are ignored.</div> <ul style="list-style-type: none">– Wrap if long - select this option to have lines going beyond the right margin wrapped with proper indentation.– Chop down if long - select this option to have elements in lists that go beyond the right margin wrapped so that there is one element per line with proper indentation.– Wrap always - select this option to have all elements in lists wrapped so that there is one element per line with proper indentation.

Blank Lines

Use this tab to insert blank lines into your code.

ItemDescription

Keep Maximum Blank Lines	Use this area to specify blank lines in your code. Use In code field to enter the number of lines you want to insert. The default number is 2.
--------------------------	---

Code Generation

The tab contains templates for the names of primary and foreign key constraints, and indexes. These templates are used to generate default names for the constraints and indexes when you create them in the Create Table or the Modify Table dialog.

The templates can contain variables (e.g. `{table}`) and text. When generating a name, the specified text is reproduced literally.

To get the info about the variables and how you should use them, place the cursor into the field of interest and press `Ctrl+Q`.

`{columns}` and `{ref_columns}`, depending on the situation, are the name of the column, or a list where the column names are separated with the underscore (`_`).

`{unique?u:}` checks if the index is unique (`unique?`), and, if it is, inserts the sequence of characters specified between `?` and `:` (in this example, it's `u`). If the index is not unique, the sequence between `:` and `}` is inserted (in this example, it's nothing).

Example. Using the template `{table}_{columns}_{unique?u:}index`, you are creating an index on the columns `FirstName` and `LastName` in the table `persons`. If the index is unique, its name, by default, will be `persons_FirstName_LastName_uindex`. If the index is not unique, its name will be `persons_FirstName_LastName_index`.

Set from...

Click this link to reveal the list of languages to be used as the base for the current language code style. So doing, only the settings that are applicable to the current language are taken. All the other settings are not affected.

This link appears in the upper-right corner of the language-specific code style page, when applicable.

Click Reset to discard changes and return to the initial set of code style settings.

Use this page to configure formatting options for Stylus files. View the result in the Preview pane on the right.

On this page:

- [Tabs and Indents](#)
- [Set from...](#)

Tabs and Indents

ItemDescription

Use tab character

- If this check box is selected, tab characters are used:
 - On pressing the `Tab` key
 - For indentation
 - For code reformatting
- When the check box is cleared, PyCharm uses spaces instead of tabs.

Smart tabs

- If this check box is selected, the part of indentation defined by the nesting of code blocks, is made of the tabs and (if necessary) spaces, while the part of indentation defined by the alignment is made only of spaces.
- If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment.

The Smart Tabs check box is available if the Use Tab Character check box is selected.

Tab size In this text box, specify the number of spaces included in a tab.

Indent In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.

Continuation indent In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted between the elements of an array, in expressions, method declarations and method calls.

Keep indents on empty lines

- If this check box is selected, then PyCharm will keep indents on the empty lines as if they contained some code.
- If this check box is not selected, PyCharm will delete the tab characters and spaces.

Set from...

Click this link to reveal the list of languages to be used as the base for the current language code style. So doing, only the settings that are applicable to the current language are taken. All the other settings are not affected.

This link appears in the upper-right corner of the language-specific code style page, when applicable.

Click Reset to discard changes and return to the initial set of code style settings.

Use this page to configure formatting options for TypeScript files. View the result in the Preview pane on the right.

On this page:

- [Tabs and Indents](#)
- [Spaces](#)
- [Wrapping and braces](#)
 - [Right Margin \(columns\)](#)
 - [Wrap on typing](#)
 - [Keep when reformatting](#)
 - [Wrapping options](#)
 - [Alignment options](#)
 - [Braces placement options](#)
- [Blank lines](#)
- [Punctuation](#)
- [Other](#)
- [Imports](#)
- [Set from...](#)

Tabs and Indents

ItemDescription

Use tab character	<ul style="list-style-type: none"> – If this check box is selected, tab characters are used: <ul style="list-style-type: none"> – On pressing the <code>Tab</code> key – For indentation – For code reformatting – When the check box is cleared, PyCharm uses spaces instead of tabs.
Smart tabs	<ul style="list-style-type: none"> – If this check box is selected, the part of indentation defined by the nesting of code blocks, is made of the tabs and (if necessary) spaces, while the part of indentation defined by the alignment is made only of spaces. – If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment. <p>The Smart Tabs check box is available if the Use Tab Character check box is selected.</p>
Tab size	In this text box, specify the number of spaces included in a tab.
Indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.
Continuation indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted between the elements of an array, in expressions, method declarations and method calls.
Keep indents on empty lines	<p>If this check box is selected, then PyCharm will keep indents on the empty lines as if they contained some code.</p> <p>If this check box is not selected, PyCharm will delete the tab characters and spaces.</p>
Indent chained methods	<p>In declarations of functions, the second and further methods in a chain are displayed on a separate line.</p> <ul style="list-style-type: none"> – When the check box is selected, the second and further methods in a chain are aligned with the first call. – When the check box is cleared, the second and further methods in a chain are aligned with the object on which they are invoked.
Indent all chained calls in a group	The check box is available only when the Indent chained methods check box is selected.

Spaces

Use this tab to specify where you want spaces in your code. To have PyCharm automatically insert a space at a location, select the check box next to this location in the list. The results are displayed in the Preview pane.

Wrapping and braces

In this tab, customize the code style options, which PyCharm will apply on [reformatting the source code](#). The left-hand pane contains the list of exceptions (Keep when reformatting), and placement and alignment options for the various code constructs (lists, statements, operations, annotations, etc.) The right-hand pane shows preview.

Alignment takes precedence over indentation options.

Right Margin (columns)

Use Right Margin field to specify a margin space required on the right side of an element. If you select Default option then a value of the right margin from the [global settings](#) is used.

Wrap on typing

Use Wrap on typing settings to specify how the edited text is fitted in the specified Right margin. You can select one the following options:

- Default - in this case PyCharm uses the Wrap on typing option that is specified in the [global settings](#).
- Yes - in this case PyCharm uses the value specified in the Right Margin field.
- No - in this case this option is switched off and a line can exceed the value specified in the right margin.

Keep when reformatting

Use the check boxes to configure exceptions that PyCharm will make when reformatting the source code. For example, by default, the *Line breaks* check box is selected. If your code contains lines that are shorter than a standard convention, you can convert them by disabling the Line breaks check box before you [reformat the source code](#).

Wrapping options

The wrapping style applies to the various code constructs, specified in the left-hand pane (for example, method call arguments, or assignment statements).

ItemDescription

Wrapping style	From this drop-down list, select the desired wrapping style: <ul style="list-style-type: none">- Do not wrap - when this option is selected, no special wrapping style is applied. With this option selected, the nested alignment and braces settings are ignored.- Wrap if long - select this option to have lines going beyond the right margin wrapped with proper indentation.- Wrap always - select this option to have all elements in lists wrapped so that there is one element per line with proper indentation.- Chop down if long - select this option to have elements in lists that go beyond the right margin wrapped so that there is one element per line with proper indentation.
----------------	--

Alignment options

ItemDescription

Align when multiline	If this check box is selected, a code construct starts at the same column on each next line. Otherwise, the position of a code construct is determined by the current indentation level.
<character(s)> on next line	Select this check box to have the specified character or characters moved to the next line when the lines are wrapped.
'else' on new line	Use this check box to have the corresponding statements or characters moved to the next line.
New line after <character>	Select this check box to have the code after the specified character moved to a new line.
Special else if treatment	If this check box is selected, <code>else if</code> statements are located in the same line. Otherwise, <code>else if</code> statements are moved to the next line to the corresponding indent level.
Indent case branches	If this check box is selected, the <code>case</code> statement is located at the corresponding indent level. Otherwise, <code>case</code> statement is placed at the same indent level with <code>switch</code> .

Braces placement options

ItemDescription

Braces placement style	Use this drop-down list to specify the position of the opening brace in class declarations, method declarations, and other types of declarations. The available options are: <ul style="list-style-type: none">- End of line - select this option to have the opening brace placed at the declaration line end.- Next line if wrapped - select this option to have the opening brace placed at the beginning of the line after the multiline declaration line.- Next line - select this option to have the opening brace placed at the beginning of the line after the declaration line.- Next line shifted - select this option to have the opening brace placed at the line after the declaration line being shifted to the corresponding indent level.- Next line each shifted - select this option to have the opening brace placed at the line after the declaration line being shifted to the corresponding indent level, and have the next line shifted to the next indent level as well.
Force braces	From this drop-down list, choose the braces introduction method for <code>if</code> , <code>for</code> , <code>while</code> , and <code>do () while</code> statements. The available options are: <ul style="list-style-type: none">- Do not force - select this option to suppress introducing braces automatically.- When multiline - select this option to have braces introduced automatically, if a statement occupies more than one line. Note that PyCharm analyzes the number of lines in the entire statement but not only its condition.- Always - select this check box to have braces always introduced automatically.

Blank lines

Use this tab to define where and how many blank lines you want PyCharm to retain and insert in your code after reformatting. For each type of location, specify the number of blank lines to be inserted. The results are displayed in the Preview pane.

ItemDescription

Keep Maximum Blank Lines	In this area, specify the number of blank lines to be kept after reformatting in the specified locations.
In code	Use this field to set the number of the blank lines.

Punctuation

Use the drop-down lists in this tab to form directives in automatic insertion of terminating semicolons, single and double quotes, and trailing commas.

ItemDescription

Semicolon to terminate statements	<ul style="list-style-type: none"> - Use semicolon to terminate statements in new code - Use semicolon to terminate statements always - Don't use semicolon to terminate statements in new code - Don't use semicolon to terminate statements always
Quotes	<ul style="list-style-type: none"> - Use double quotes in new code - Use double quotes always - Use single quotes in new code - Use single quotes always
Trailing comma	<p>Use this drop-down list to configure whether you want to use trailing commas in objects, arrays, and for the parameters in method definitions and calls. The available options are:</p> <ul style="list-style-type: none"> - Keep - Remove - Add when multiline

Other

Item Description

Align object properties	<p>From the drop-down list, select the type of objects' alignment:</p> <ul style="list-style-type: none"> - Do not align: the attributes in sequential lines will be not aligned. - On colon: the attributes in sequential lines will be aligned against the colon. - On value: the attributes in sequential lines will be aligned against the value.
Use semicolon to terminate statements	Select this check box to have statements terminated with a semicolon.
Generated Code	<p>In this area, configure the code style in <code>import</code> statements, see Creating Imports, section Importing TypeScript Symbols.</p> <ul style="list-style-type: none"> - String literal type: from this drop-down list, choose the type of quotes to be used in <code>import</code> statements. The available options are Single quotes and Double quotes.

Use 'public' modifier	<p>Use this check box to have the <code>public</code> access modifier inserted or omitted in the generated code. For example, during generation of a <code>public</code> method from the following:</p>
-----------------------	---

```
class Test {
  public test():void {
    var x = 1;
  }
}
```

- If the check box is selected, the `public` access modifier is automatically inserted in the generated code:

```
class Test {
  public test():void {
    this.extracted();
  }
  extracted(){
    var x = 1;
  }
}
```

- If the check box is cleared, the `public` access modifier is omitted during code generation:

```
class Test {
  public test():void {
    this.extracted();
  }
  public extracted(){
    var x = 1;
  }
}
```

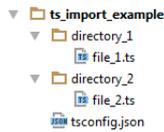
See [TypeScript Language Handbook](#), chapter Private/Public Modifiers .

Imports

ItemDescription

Merge imports for members from the same module	<ul style="list-style-type: none"> - When this check box is selected, imported symbols from the same module are listed in one <code>import</code> statement with a comma as separator. The members are listed in the order in which they are imported. To have them listed alphabetically, select the Sort imported members check box and run Code Optimize Imports. - When this check box is cleared, for each imported symbol a separate <code>import</code> statement is generated.
--	--

Use paths relative to <code>tsconfig.json</code>	<p>When this check box is selected, PyCharm calculates import paths using the <code>tsconfig.json</code> file as the root. When this check box is cleared, PyCharm calculates import paths relative to the project root. For example, if your project is structured as follows:</p>
--	---



With the check box selected, PyCharm generates the following import statement:

```
import {ClassName} from 'directory_2/file_2'
```

If the check box is cleared, the following import statement is generated:

```
import {ClassName} from '../directory_2/file_2'
```

Use directory import (Node-style module resolution)

- When this check box is selected, `import` statements are generated in compliance with the [Node.js module resolution strategy](#).
- When this check box is cleared, `import` statements are generated in compliance with the [TypeScript classic module resolution strategy](#).

Do not import exactly from

In this field, specify the exact paths that PyCharm should skip during automatic import of a symbol. Instead, PyCharm will look for alternative paths to import the symbol.

This is particularly useful for modules that allow importing their submodules instead of the entire module. For example, to prefer imports like `import {Observable} from 'rxjs/Observable'` to a more general `import {Observable} from 'rxjs'`, add `rxjs` to the list.

To manage the list of modules to skip:

1. Click  to the right of the field.
2. In the Change modules dialog box that opens, click  and specify the module name in the Add module dialog box. To remove a module from the list, select it and click .

Sort imported members

- When this check box is selected, PyCharm lists the imported members in merged `import` statements alphabetically. Note that the members are listed comma-separated in the order they are imported and re-sorted only when you run Code | Optimize Imports.
- When this check box is cleared, the members in merged `import` statements are always listed comma-separated in the order they are imported.

Sort imports by modules

- When this check box is selected, `import` statements are re-sorted alphabetically by the module names when you run Code | Optimize Imports.
- When this check box is cleared, `import` statements are always shown in the order they are generated and this order is not changed after you run Code | Optimize Imports.

Set from...

Click this link to reveal the list of languages to be used as the base for the current language code style. So doing, only the settings that are applicable to the current language are taken. All the other settings are not affected.

This link appears in the upper-right corner of the language-specific code style page, when applicable.

Click Reset to discard changes and return to the initial set of code style settings.

Use this page to configure formatting options for XML files. View the result in the Preview pane on the right.

On this page:

- [Tabs and Indents](#)
- [Other](#)
- [Arrangement](#)
- [Set from...](#)

Tabs and Indents

ItemDescription

Use tab character	<ul style="list-style-type: none"> – If this check box is selected, tab characters are used: <ul style="list-style-type: none"> – On pressing the <code>Tab</code> key – For indentation – For code reformatting – When the check box is cleared, PyCharm uses spaces instead of tabs.
Smart tabs	<ul style="list-style-type: none"> – If this check box is selected, the part of indentation defined by the nesting of code blocks, is made of the tabs and (if necessary) spaces, while the part of indentation defined by the alignment is made only of spaces. – If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment. <p>The Smart Tabs check box is available if the Use Tab Character check box is selected.</p>
Tab size	In this text box, specify the number of spaces included in a tab.
Indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.
Continuation indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted between the elements of an array, in expressions, method declarations and method calls.
Keep indents on empty lines	<p>If this check box is selected, then PyCharm will keep indents on the empty lines as if they contained some code.</p> <p>If this check box is not selected, PyCharm will delete the tab characters and spaces.</p>

Other

ItemDescription

Right Margin	Use these settings to specify a margin space required on the right side of an element. If you select Default option then a value of the right margin from the global settings will be used.
Wrap on typing	<p>Use these settings settings to specify how the edited text is fitted in the specified Right margin. You can select one the following options:</p> <ul style="list-style-type: none"> – Default - in this case PyCharm uses the Wrap on typing option that is specified in the global settings. – Yes - in this case the value in the specified right margin is used. – No - in this case this option is switched off and a line can exceed the number that is specified in the right margin.
Keep line breaks	Select this check box to have PyCharm honor line breaks when reviewing XML files in the editor.
Keep line breaks in text	Select this check box to have PyCharm honor line breaks in attributes (for example, lengthy descriptions) when reviewing XML files in the editor.
Keep blank lines	In this text box, specify the minimum number of sequential blank lines to be retained after reformatting.
Wrap attributes	<p>Use this drop-down list to determine how attribute lines should be wrapped. The available options are:</p> <ul style="list-style-type: none"> – Do not wrap - if this option is selected, no special wrapping style is applied to the code. – Wrap if long - select this option to have lines going beyond the right margin wrapped with proper indentation. – Chop down if long - select this option to have elements in lists that go beyond the right margin wrapped to give one element per line with proper indentation. – Wrap always - select this option to have all elements in lists wrapped to give one element per line with proper indentation.
Wrap text	Select this check box to have long lines wrapped according to the code style settings.
Align attributes	Select this check box to have attributes in sequential lines aligned.
Keep white spaces	When this check box is selected, the editor preserves all whitespaces within tags. The same refers also to the indents, and line breaks.
Spaces	<p>In this area, define the usage of spaces for attributes and tag names.</p> <ul style="list-style-type: none"> – Around "=" in attribute: select this check box to have spaces added around the "=" symbol in attributes. – After tag name: select this check box to have spaces added after tag names. – In empty tag: select this check box to have spaces added in empty tags.
CDATA	<p>In this area, define the usage of whitespaces around and inside CDATA sections in MXML files:</p> <ul style="list-style-type: none"> – Whitespaces around: from the drop-down list, choose how whitespaces around <code>CDATA</code> will be treated.

- Preserve: all whitespaces will be left intact after reformatting.
- Remove (keep with tags): all whitespaces around `<CDATA` will be removed, and tags will be kept on the same lines.
- New lines: new lines will be added before and after `<CDATA`.
- Keep whitespaces inside: If this check box is selected, whitespaces will be preserved after `<CDATA[` and before `]`.

Arrangement

This tab lets you define a set of rules that rearranges your code according to your preferences.

ItemDescription

Matching rules	<p>Use this area to define elements order as a list of rules, where every rule has a set of matches such as modifier or type.</p> <ul style="list-style-type: none"> -  - use this button to add a rule. The empty rule dialog window opens. -  - use this button to remove the rule from the list. -  - use this button to edit an existing rule. To see this button, navigate to the rule that you want to edit and click on the button. In pop-up window that opens, modify the rule fields. -  - use these buttons to move the selected rule up or down.
Empty rule	<p>Use this window to create a new matching rule or edit an existing one. You can select from the following filters:</p> <ul style="list-style-type: none"> - Type - use this filter to choose classes or methods for your rule. <p>Note that clicking twice on the type keyword will negate the condition.</p> <ul style="list-style-type: none"> - Name - use this field to specify entry names in the rule. This filter matches only entry names such as field names, method names, class names and etc. The filter supports regular expressions and uses a standard syntax. The match is performed against the entire name. - Namespace - use this field to specify the namespace in the rule. It lets you specify a rule that controls a namespace attribute position. - Order - use this drop-down list to select the sorting order for the rule. This option is useful when more than one element uses the same matching rule. In this case, selecting Keep order will keep the same order as was set before the rearrangement and selecting Order by Name will sort the elements with the same matching rule by their names. - Aliases - this option displays aliases that were defined in the Rules Alias Definition dialog. You can remove the ones you do not need.
	<p>This icon appears when you select Order by Name in Order option. The icon indicates that the items in this rule are alphabetized.</p>
Additional Settings	<p>Use this area to set additional arrangement options. The Force rearrange drop-down list lets you select options that affect the Rearrange entries check box in the Reformat Code dialog.</p> <p>You can select from the following options:</p> <ul style="list-style-type: none"> - Use current mode (toggled in the Reformat Code dialog) - In this case the Rearrange entries check box stays active and you can modify it in the Reformat Code dialog. - Always - In this case the Rearrange entries check box is selected and becomes read-only. - Never - In this case the Rearrange entries check box is cleared and becomes read-only.

Set from...

Click this link to reveal the list of languages to be used as the base for the current language code style. So doing, only the settings that are applicable to the current language are taken. All the other settings are not affected.

This link appears in the upper-right corner of the language-specific code style page, when applicable.

Click Reset to discard changes and return to the initial set of code style settings.

Use this page to configure formatting options for YAML files. View the result in the Preview pane on the right.

On this page:

- [Tabs and Indents](#)
- [Set from...](#)

Tabs and Indents

ItemDescription

- | | |
|-------------------|---|
| Use tab character | <ul style="list-style-type: none">– If this check box is selected, tab characters are used:<ul style="list-style-type: none">– On pressing the <code>Tab</code> key– For indentation– For code reformatting– When the check box is cleared, PyCharm uses spaces instead of tabs. |
|-------------------|---|
-

Indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.
--------	---

Set from...

Click this link to reveal the list of languages to be used as the base for the current language code style. So doing, only the settings that are applicable to the current language are taken. All the other settings are not affected.

This link appears in the upper-right corner of the language-specific code style page, when applicable.

Click [Reset](#) to discard changes and return to the initial set of code style settings.

Ctrl+Alt+S



On this page:

- [Scheme](#)
- [Text files and unsupported file types](#)

Scheme

In this area, choose the code style scheme and change it as required. Code style scheme settings are automatically applied every time PyCharm generates, refactors, or reformats your code.

Code styles are defined at the project level and at the IDE level (global).

- At the **Project** level, settings are grouped under the Project scheme, which is predefined and is marked in bold. The **Project** style scheme is applied to the current project only.

You can copy the Project scheme to the IDE level, using the Copy to IDE... command.

- At the **IDE** level, settings are grouped under the predefined Default scheme (marked in bold), and any other scheme created by the user by the Duplicate command (marked as plain text). Global settings are used when the user doesn't want to keep code style settings with the project and share them.

You can copy the IDE scheme to the current project, using the Copy to Project... command.

ItemDescription

Scheme From this drop-down list, select the scheme to be used. The **predefined** schemes are shown bold. The **custom** schemes, ones created as copies of the predefined schemes, are in plain text. The location where the scheme is stored is written next to each scheme, for example, the Default scheme is stored in the IDE, the Project scheme is stored in the project.

Click this button to invoke the drop-down list of commands to manage the schemes:

ItemDescriptionAvailable for

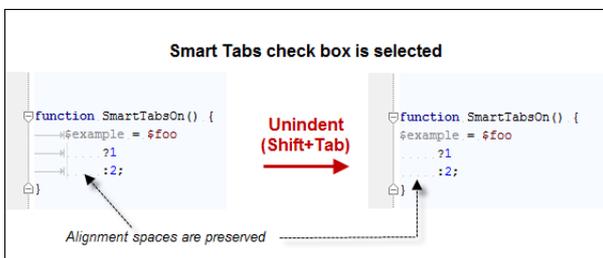
Copy to IDE...	Choose this command to copy the scheme settings to the IDE.	Project
Export...	Choose this command to export the selected scheme to an <code>xml</code> file in the selected location:	Project and IDE
Import Scheme...	Choose this command to import the scheme of the selected type from the specified location:	Project and IDE
Copy to Project...	Choose this command to copy the scheme settings to be stored with a project.	IDE
Duplicate...	Choose this command to create a copy of the selected scheme.	IDE
Reset	Choose this command to reset the default or bundled color scheme to the initial defaults shipped with PyCharm. This command becomes available only if some changes have been done.	IDE
Rename	Choose this command to change the name of the selected custom scheme. Press <code>Enter</code> to save changes, or <code>Escape</code> to cancel.	Custom schemes

Text files and unsupported file types

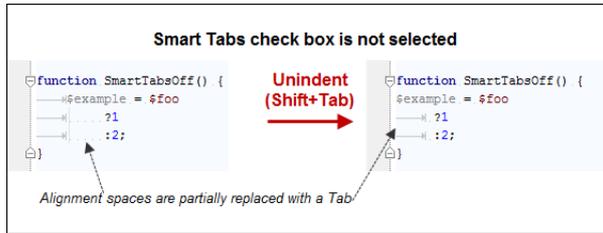
ItemDescription

- Use tab character**
- If this check box is selected, tab characters are used:
 - On pressing the `Tab` key
 - For indentation
 - For code reformatting
 - When the check box is cleared, PyCharm uses spaces instead of tabs.

- Smart tabs**
- If this check box is selected, the part of indentation defined by the nesting of code blocks, is made of the tabs and (if necessary) spaces, while the part of indentation defined by the alignment is made only of spaces.



- If this check box is cleared, only tabs are used. This means that a group of spaces that fits the specified tab size is automatically replaced with a tab, which may result in breaking fine alignment.



The Smart Tabs check box is available if the Use Tab Character check box is selected.

Tab size	In this text box, specify the number of spaces included in a tab.
Indent	In this text box, specify the number of spaces (or tabs if the Use Tab Character check box is selected) to be inserted for each indent level.

Inspections

File | Settings | Editor | Inspections for Windows and Linux

PyCharm | Preferences | Editor | Inspections for macOS

Ctrl+Alt+S



Use this page to [customize inspection profiles](#), [configure inspection severities](#), [disable and enable inspections](#), and [configure inspections for different scopes](#).

The page is divided into the following areas:

- [Profile management](#)
- [Toolbar](#)
- [Inspection severity and scopes](#)
- [Options](#)

Profile management

ItemDescription

Profile	From this drop-down list, select the name of the profile to configure. All modified inspections are highlighted. Note that the selected profile is automatically used for project highlighting after clicking Apply.
*-	Click this button to reveal the following submenu:
Copy to	Choose the command Copy as IDE to move the selected profile to the global level.
IDE/Copy to Project	Choose the command Copy to Project to create the project level duplicate of the selected profile.
Duplicate	Choose this command to create a copy, based on the current profile .
Rename	Choose this command to change the name of the current profile in the Profile field.
Delete	Choose this command to delete the current profile. The pre-defined profiles cannot be deleted, so this command is only available for the user-defined profiles.
Add description	If this command is selected, a text field appears on the right of the Profile field, enabling you to type in the description of the current profile. Press Enter to save the entered description, or Escape to cancel typing.
Export	Choose this command to export the selected profile as an <code>xml</code> file.
Import Profile	Choose this command to import a profile from an <code>xml</code> file.

Toolbar

Item Tooltip and Shortcut



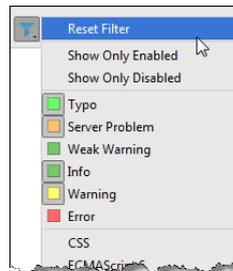
Use this text box to search through the list of inspections. In the search field, start typing the desired inspection name, or any characters contained in the inspection name or description. PyCharm shows the list of matching occurrences. As you type a search string, the matching inspections are highlighted. To finalize the search, press **Enter**. The used search strings are memorized in the history list.

- : Click this button to reveal the history list.
- : Click this button to clear the search history.



Filter Inspections

Click this button to show the list of available filters:



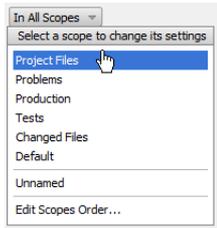
Click the desired filters to reduce the list. The command Reset Filters becomes available, if some of the filters are checked.

	Expand All/Collapse All	Click these buttons to have all inspection nodes expanded/collapsed. <div style="text-align: center;"> Ctrl+NumPad Plus Ctrl+NumPad - </div>
	Reset to Empty	Click this button to have all the check boxes of the profile cleared and thus disable all the profile inspections.
	Advanced Settings	Click this button to show the menu with the following check commands: <ul style="list-style-type: none"> - Disable new inspections by default: select this check box, if you don't want the new inspections that appear after PyCharm update, to become available. - Reset to Default settings: select this check box to discard all changes.

Inspection severity and scopes

ItemDescription

Description	This read-only field shows the description of the selected inspection.
Inspection severity	From this drop-down list, select the desired severity to assign to the current inspection. Refer to the section Configuring Inspection Severities for details.
Scopes	Click this drop-down list to reveal the list of available scopes:



Clicking a scope in the list results in showing the scopes toolbar:

Alt+Insert	Click this button to specify the scope for the selected inspection.
Alt+Delete	Click this button to delete the selected scope for the current inspection.

Choosing the option Edit Scopes Order results in showing the Scopes Order dialog box, where one can change the order of scopes using the up and down arrows (↑↓) or keyboard shortcuts (Alt+Up / Alt+Down).

It is possible to select several inspections and add/remove scopes for the entire selection.

Options

This area is only available for some types of inspections, provided that an inspection of this type is enabled (the check box next to it is selected). Use the controls in this area to re-configure the default inspection settings.

File and Code Templates

File | Settings | Editor | File and Code Templates for Windows and Linux

PyCharm | Preferences | Editor | File and Code Templates for macOS

Ctrl+Alt+S



Files can be created according to pre-defined templates (file templates). Use this page to view, edit, and create such templates.

Different groups of templates are located on different [tabs](#).

When you select a template, its contents and description are displayed in the right-hand part of the page.

- [Per-project vs default scheme](#)
- [Tabs](#)
- [Toolbar](#)
- [Template settings and contents](#)

Per-project vs default scheme

ItemDescription

Scheme From this drop-down list, choose whether file and code template settings pertain to the entire workspace, or the current project:

- **Default scheme** is selected, when file and code templates are global.
- **Project scheme** is selected, if you want to use the sharable project-specific file and code templates.

Refer to the section [Project and IDE Settings](#) for details.

Tabs

TabDescription

Files This tab displays the available file templates. You can edit the existing templates, or create new ones.

Note The templates shown in bold font cannot be deleted; their names and extensions cannot be edited.

Includes This tab shows the templates for reusable fragments that can be included in file templates. You can edit the existing templates, or create new ones.

Code This tab displays built-in snippets, i.e. templates for code fragments that PyCharm can generate in various typical situations, for example, for generating implemented or overridden method bodies. You can edit the existing snippets, but you cannot create new ones.

Toolbar

ItemTooltipDescription

	Create Template	Click this button to create a new template in the currently opened tab. This option is only available in the Files and the Includes tabs. The location of the new template is defined by the Schema drop-down list. If
	Remove Template	Click this button to delete the selected template. This option is only available for certain templates in the Files and the Includes tabs.
	Copy Template	Click this button to create a copy of the selected template. This option is only available in the Files and the Includes tabs.
	Reset to Default	Click this button to revert the selected template to its original state. This option is only available for templates that have been modified (they are highlighted in blue).
N/A	Reset	This link appears in the top-right corner of the page when you start editing a template. Clicking this link resets all unsaved changes to any template in any tab.

Template settings and contents

ItemDescription

Name This text box appears when a new template is created. Specify the name of the new template.

Extension In this text box, specify the extension. PyCharm will apply this template when new files of this type are created.

Template text Edit the template contents. You can use:

- Plain text.
- `#parse` directives to work with [template includes](#).
- Custom variables. Their names can be defined right in the template through the `#set` directive or will be defined during the file creation.
- Variables to be expanded into corresponding values in the `${<variable_name>}` format.

The available predefined file template variables are:

`$(PROJECT_NAME)` - the name of the current project.

- `{PROJECT_NAME}` - the name of the current project.
- `{NAME}` - the name of the new file which you specify in the New File dialog box during the file creation.
- `{USER}` - the login name of the current user.
- `{DATE}` - the current system date.
- `{TIME}` - the current system time.
- `{YEAR}` - the current year.
- `{MONTH}` - the current month.
- `{DAY}` - the current day of the month.
- `{HOUR}` - the current hour.
- `{MINUTE}` - the current minute.
- `{PRODUCT_NAME}` - the name of the IDE in which the file will be created.
- `{MONTH_NAME_SHORT}` - the first 3 letters of the month name. Example: Jan, Feb, etc.
- `{MONTH_NAME_FULL}` - full name of a month. Example: January, February, etc.

Treating dollar sign

- You can prevent treating dollar characters (\$) in template variables as prefixes. If you need a dollar character (\$) inserted as is, use the `{DS}` file template variable instead. When the template is applied, this variable evaluates to a plain dollar character (\$).
- Examples:
- To use some version control keywords (such as `$Revision$` , `$Date$` , etc.) in your default class template, write `{DS}` instead of the dollar prefix (\$).
 - The template code `{DS}this` will be rendered as `$this` .

Note PyCharm doesn't prompt for the values of Velocity variables defined with `#set` .

Reformat according to style	Select this check box, to have PyCharm reformat generated stub files according to the style defined on the Code Style page . This option is only available in the Files tab.
Enable Live Templates	Select this check box to use a live template inside a file template. So doing, one has to put the live template fragments into Velocity escape syntax. For example: <pre>#[[\$MY_VARIABLE\$ \$END\$]]</pre> Thus, one can specify the cursor position. Note that it is required to use the live template variables here!
Description	This read-only field provides information about the template, its predefined variables, and the way they work. This field is not available in custom templates.

File Encodings

File | Settings | Editor | File Encodings for Windows and Linux

PyCharm | Preferences | Editor | File Encodings for macOS

Ctrl+Alt+S



Use this dialog to configure encoding options for a project and for the entire IDE.

The [file or directory encodings](#) take precedence over the [project encoding](#), which, in turn, takes precedence over the [IDE encoding](#).

If the file or directory encodings are not defined, then the project encoding is taken. If the project encoding cannot be taken (for example, if a project is not yet created), then the IDE encoding is taken.

ItemDescription

IDE Encoding	From this drop-down list, choose the encoding to be used when no project is currently opened. The encoding will be applied, for example, when you specify settings of a default project or check out sources from a version control storage. Choose System Default to have the default encoding of your operating system used or choose a specific encoding.
Project Encoding	From this drop-down box, choose the default encoding to use in the folders for which no encoding is appointed in the Default encoding field below. Choose System Default to have the default encoding of your operating system used or choose a specific encoding.
File/Directory	This column displays the project tree view.
Default encoding	This column displays encoding for a file or directory, if applicable. If encoding is defined within a file, it cannot be configured, and is shown in grey font. If encoding is configurable, click the Default Encoding column for a selected file or directory, and choose encoding from the drop-down list. Encoding information embedded in a file overrides the selected one; encoding information for the nested files or directories overrides that for the outer directories or the whole project.
Default encoding for properties files	Use this drop-down list to define encoding for the properties files in the project.
Transparent native-to-ascii conversion	Select this option to show in properties files the national characters (non-ISO 8859-1), stored as escape sequences. If this check box is not selected, the national characters will not be shown.

Live Templates

File | Settings | Editor | Live Templates for Windows and Linux

PyCharm | Preferences | Editor | Live Templates for macOS

Ctrl+Alt+S



Use this page to [create, manage, and edit live templates](#).

On this page:

- [List of available live templates](#)
- [Context menu of a live template](#)
- [Template editing area](#)
- [Side note about predefined template variables](#)
- [Predefined functions to be used in live template variables](#)

List of available live templates

Item	Tooltip and shortcut	Description
------	----------------------	-------------

By default expand with		Use this drop-down list to specify the default invocation key for all templates. Individual expansion keys for the particular templates are defined in the editing area . If the standard expansion keys (Tab, Enter, or Space) are not desirable, select the Custom option from this drop-down list. When Custom is selected, the Change link appears next to the drop-down, leading you to the Keymap page.
------------------------	--	---

Live Templates		This list shows all currently available template abbreviations supplied with their descriptions. The abbreviations are grouped below nodes and sorted alphabetically within each group. To activate a template or an entire group, select the check box to its left. Note <ul style="list-style-type: none">- Only active templates are displayed upon invoking live templates in the editor.- If a template is active, the editor is sensitive to its abbreviation. Otherwise, the abbreviation is considered merely a set of characters.
----------------	--	---

+	Add	Click this button to have a new template item added to the current group of template. You can define the template abbreviation, description, text, variables, expansion key, and context in the editing area below .
---	-----	--

Alt+Insert

-	Remove	Click this button to have the selected live template removed from the list.
---	--------	---

Delete



Duplicate	Click this button to create a new template based on the selected template. A new template item is added to the current node and the fields in the Template Text area show the definition of the selected template.
-----------	--



Restore	Click this button to restore the deleted live templates. This button is only enabled when the changes are applied.
---------	--

Context menu of a live template

Item	Description
------	-------------

Move	Choose a group to move the selected template to.
------	--

Change context	Choose this command to modify the set of contexts where the current template is enabled. Upon choosing this command, a list of supported language contexts is displayed. To make PyCharm consider a context sensitive to the template, select a check box next to the context name. The available context types depend on the enabled plugins.
----------------	--

Copy	Choose this command to create a serialized template XML in the system clipboard. Refer to section Sharing Live Templates .
------	--

Paste	Choose this command to paste an XML representation of the copied templates to the selected group of templates . Refer to section Sharing Live Templates .
-------	---

Restore defaults	This command only appears on the context menus of the modified templates, marked blue. Choose this command to restore the default template settings.
------------------	--

Template editing area

The focus is moved to this area in the following cases:

- When you click the Add + or Copy button.
- When you select a live template in the list.
- When you select a fragment of code in the editor and choose [Tools | Save as Live Template](#).

Use controls of this area to create new [live templates](#) and edit the settings for the existing ones.

You can navigate through the Template Text Area using the hot keys that are marked in the field labels.

ItemDescription

Abbreviation	In this text box, specify the template abbreviation .
Description	In this text box, provide optional description of a template or an example of its usage.
Template Text	<p>In this text box, type the template body that may contain plain text and variables in the format <code><variable name>\$</code>.</p> <p>When editing the live template variables, mind the following helpful hints:</p> <ul style="list-style-type: none">- If you need a dollar sign (<code>\$</code>) in the template text, escape it by duplicating this character (<code>\$\$</code>).- To change the variables in a template, click the Edit Variables button and configure the variables as described in Creating and Editing Template Variables. <p>The Edit Variables button is enabled only if the template body contains at least one user-defined variable, that is, a variable different from <code>\$END\$</code> or <code>\$SELECTION\$</code>.</p>

Side note about predefined template variables

PyCharm supports two predefined live template variables: `END` and `$SELECTION$`.

You cannot edit the predefined live template variables `END` and `$SELECTION$`.

- `END` indicates the position of the cursor after the template is expanded. For example, the template `return END;` will be expanded into

```
return ;
```

with the cursor positioned **right before** the semicolon.

- `$SELECTION$` is used in surround templates and stands for the code fragment to be wrapped. After the template is expanded, the selected text is wrapped as specified in the template.

For example, if you select `EXAMPLE` in your code and invoke the `"$SELECTION$"` template via the assigned abbreviation or by pressing `Ctrl+Alt+T` and selecting the desired template from the list, PyCharm will wrap the selection in double quotes as follows:

```
"EXAMPLE"
```

Applicable in:	This read-only field shows the languages and/or pieces of code where the editor should be sensitive to the template. Upon pressing <code>Ctrl+J</code> in such context, PyCharm displays a list of templates that are valid for this context.
Change	Click this link to modify the set of contexts where the current template is enabled. Upon clicking the link, a list of supported language contexts is displayed. To make PyCharm consider a context sensitive to the template, select a check box next to the context name. The available context types depend on the enabled plugins.
Edit Variables	Click this button to open the Edit Template Variables dialog box, where you can define how PyCharm should process template variables upon template expansion. The Edit Variables button is enabled only if the template body contains at least one user-defined variable, that is, a variable different from <code>\$END\$</code> or <code>\$SELECTION\$</code> . The Edit Template Variables dialog box contains a complete list of available functions. See the list of predefined functions below on this page.
Options	In this area, define the behavior of the editor when a template is expanded. <ul style="list-style-type: none">- Expand with - from this drop-down list, choose the key to invoke the template.- Reformat according to style - select this check box to have PyCharm automatically reformat the expanded text according to the current style settings, defined on the Code Style page.

Predefined functions to be used in live template variables

Warning! Note that the list of predefined functions in PyCharm depends upon the installed and enabled plugins.

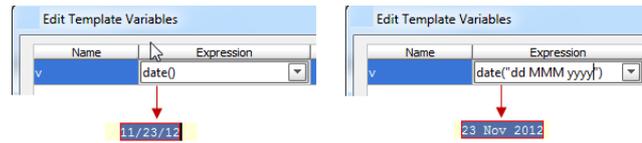
ItemDescription

<code>camelCase(String)</code>	Returns the string passed as a parameter, converted to camel case. For example, <code>my-text-file / my text file / my_text_file</code> will be converted to <code>myTextFile</code> .
<code>capitalize(String)</code>	Capitalizes the first letter of the name passed as a parameter.
<code>capitalizeAndUnderscore(sCamelCaseName)</code>	Capitalizes the all letters of a CamelCase name passed as a parameter, and inserts an underscore between the parts. For example, if the string passed as a parameter is <code>FooBar</code> , then the function returns <code>FOO_BAR</code> .
<code>classNameComplete()</code>	This expression substitutes for the class name completion at the variable position.
<code>clipboard()</code>	Returns the contents of the system clipboard.
<code>snakeCase(String)</code>	Returns CamelCase string out of snake_case string. For example, if the string passed as a parameter is <code>foo_bar</code> , then the function returns <code>fooBar</code> .
<code>collectionElementName</code>	Removes <code>_list</code> and plural ending (s).
<code>complete()</code>	This expression substitutes for the code completion invocation at the variable position.
<code>completeSmart()</code>	This expression substitutes for the smart type completion invocation at the variable position.

`date(sDate)`

Returns the current system date in the specified format.

By default, the current date is returned in the default system format. However, if you specify date format in double quotes, the date will be presented in this format:



`decapitalize(sName)`

Replaces the first letter of the name passed as a parameter with the corresponding lowercase letter.

`djangoBlock`

Shows completion popup for the available Django blocks.

`djangoFilter`

Shows completion popup for the available Django filters.

`djangoTemplateTags`

Shows completion popup for the available Django template tags

`djangoVariable`

Shows completion popup for the available Django variable.

`enum(sCompletionString1,sCompletionString2,...)`

List of comma-delimited strings suggested for completion at the template invocation.

`escapeString(sEscapeString)`

Escapes the specified string.

`expectedType()`

Returns the type which is expected as a result of the whole template. Makes sense if the template is expanded in the right part of an assignment, after return, etc.

`fileName(sFileName)`

Returns file name with extension.

`fileNameWithoutExtension()`

Returns file name without extension.

`firstWord(sFirstWord)`

Returns the first word of the string passed as a parameter.

`lineNumber()`

Returns the current line number.

`lowercaseAndDash(String)`

Returns lower case separated by dashes, of the string passed as a parameter. For example, the string `MyExampleName` is converted to `my-example-name`.

`snakeCase(sCamelCaseText)`

Returns snake_case string out of CamelCase string passed as a parameter.

`spaceSeparated(String)`

Returns string separated with spaces out of CamelCase string passed as a parameter. For example, if the string passed as a parameter is `fooBar`, then the function returns `foo bar`.

`pyClassName()`

Returns the name of the current Python class (the class where the template is expanded).

`pyFunctionName()`

Returns the name of the current Python function.

`time(sSystemTime)`

Returns the current system time.

`underscoresToCamelCase(sCamelCaseText)`

Returns the string passed as a parameter with CamelHump letters substituting for underscores. For example, if the string passed as a parameter is `foo_bar`, then the function returns `fooBar`.

`underscoresToSpaces(sParameterWithSpaces)`

Returns the string passed as a parameter with spaces substituting for underscores.

`user()`

Returns the name of the current user.

`JsArrayVariable`

Returns JavaScript array name.

`jsClassName()`

Returns the name of the current JavaScript class.

`jsComponentType`

Returns the JavaScript component type.

`jsMethodName()`

Returns the name of the current JavaScript method.

`jsQualifiedClassName`

Returns the complete name of the current JavaScript class.

`jsSuggestIndexName`

Returns a suggested name for an index.

`jsSuggestVariableName`

Returns a suggested name for a variable.

Ctrl+Alt+S



The dialog opens when you click the Edit Variables button in the [Template Text](#) area on the [Live Templates](#) page.

The Edit Variables button is enabled only if the template body contains at least one user-defined variable, that is, a variable different from `END` or `$SELECTION$`.

Use this dialog box to create and edit expressions for variables in the selected live template.

On this page:

- [Controls](#)
- [Predefined Functions to Use in Live Template Variables](#)

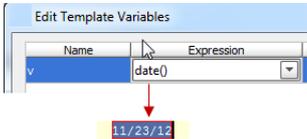
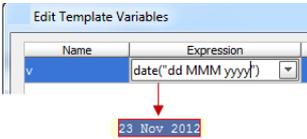
Controls

ItemDescription

Name	In this text box, view or edit the variable name in the format <code><variable_name>\$</code> .
Expression	In this text box, specify the expression to have the value of the corresponding template input field calculated automatically. This expression may contain constructs of the following basic types: <ul style="list-style-type: none"> - String constants in double quotes. - The name of another variable defined in a live template. - Predefined functions with possible arguments. <p>Type an expression manually or select a predefined function from the drop-down list. The list shows also the number and type of parameters, if any, for the selected function. The available functions are listed alphabetically in the Functions table.</p>
Default value	In this text box, specify the default string to be entered in the corresponding input field of the expanded template, if the expression does not give any result after calculation. <p>Note that a default value of a variable is an expression that can refer to other live template variables. To define the default value as a literal, enclose it in quotation marks.</p>
Skip if defined	Select this check box to have PyCharm proceed with the next input field, if the value of the current input field is defined.
Move Up / Move Down	Use these buttons to change the order of variables in the list. The order of variables in the table determines the order in which PyCharm will switch between the corresponding input fields when the template is expanded.

Predefined Functions to Use in Live Template Variables

ItemDescription

<code>camelCase(String)</code>	Returns the string passed as a parameter, converted to camel case. For example, <code>my-text-file / my text file / my_text_file</code> will be converted to <code>myTextFile</code> .
<code>capitalize(String)</code>	Capitalizes the first letter of the name passed as a parameter.
<code>capitalizeAndUnderscore(sCamelCaseName)</code>	Capitalizes the all letters of a CamelCase name passed as a parameter, and inserts an underscore between the parts. For example, if the string passed as a parameter is <code>FooBar</code> , then the function returns <code>FOO_BAR</code> .
<code>classNameComplete()</code>	This expression substitutes for the class name completion at the variable position.
<code>clipboard()</code>	Returns the contents of the system clipboard.
<code>snakeCase(String)</code>	Returns CamelCase string out of snake_case string. For example, if the string passed as a parameter is <code>foo_bar</code> , then the function returns <code>fooBar</code> .
<code>collectionElementName</code>	Removes <code>_list</code> and plural ending (s).
<code>complete()</code>	This expression substitutes for the code completion invocation at the variable position.
<code>completeSmart()</code>	This expression substitutes for the smart type completion invocation at the variable position.
<code>date(sDate)</code>	Returns the current system date in the specified format. <p>By default, the current date is returned in the default system format. However, if you specify date format in double quotes, the date will be presented in this format:</p> <div style="display: flex; justify-content: space-around; align-items: center;"> <div style="text-align: center;">  </div> <div style="text-align: center;">  </div> </div>
<code>decapitalize(sName)</code>	Replaces the first letter of the name passed as a parameter with the corresponding lowercase letter.
<code>djangoBlock</code>	Shows completion popup for the available Django blocks.
<code>djangoFilter</code>	Shows completion popup for the available Django filters.

<code>djangoTemplateTags</code>	Shows completion popup for the available Django template tags
<code>djangoVariable</code>	Shows completion popup for the available Django variable.
<code>enum(sCompletionString1,sCompletionString2,...)</code>	List of comma-delimited strings suggested for completion at the template invocation.
<code>escapeString(sEscapeString)</code>	Escapes the specified string.
<code>expectedType()</code>	Returns the type which is expected as a result of the whole template. Makes sense if the template is expanded in the right part of an assignment, after return, etc.
<code>fileName(sFileName)</code>	Returns file name with extension.
<code>fileNameWithoutExtension()</code>	Returns file name without extension.
<code>firstWord(sFirstWord)</code>	Returns the first word of the string passed as a parameter.
<code>lineNumber()</code>	Returns the current line number.
<code>lowercaseAndDash(String)</code>	Returns lower case separated by dashes, of the string passed as a parameter. For example, the string <code>MyExampleName</code> is converted to <code>my-example-name</code> .
<code>snakeCase(sCamelCaseText)</code>	Returns snake_case string out of CamelCase string passed as a parameter.
<code>spaceSeparated(String)</code>	Returns string separated with spaces out of CamelCase string passed as a parameter. For example, if the string passed as a parameter is <code>fooBar</code> , then the function returns <code>foo bar</code> .
<code>pyClassName()</code>	Returns the name of the current Python class (the class where the template is expanded).
<code>pyFunctionName()</code>	Returns the name of the current Python function.
<code>time(sSystemTime)</code>	Returns the current system time.
<code>underscoresToCamelCase(sCamelCaseText)</code>	Returns the string passed as a parameter with CamelHump letters substituting for underscores. For example, if the string passed as a parameter is <code>foo_bar</code> , then the function returns <code>fooBar</code> .
<code>underscoresToSpaces(sParameterWithSpaces)</code>	Returns the string passed as a parameter with spaces substituting for underscores.
<code>user()</code>	Returns the name of the current user.
<code>JsArrayVariable</code>	Returns JavaScript array name.
<code>jsClassName()</code>	Returns the name of the current JavaScript class.
<code>jsComponentType</code>	Returns the JavaScript component type.
<code>jsMethodName()</code>	Returns the name of the current JavaScript method.
<code>jsQualifiedClassName</code>	Returns the complete name of the current JavaScript class.
<code>jsSuggestIndexName</code>	Returns a suggested name for an index.
<code>jsSuggestVariableName</code>	Returns a suggested name for a variable.

File Types

File | Settings | Editor | File Types for Windows and Linux

PyCharm | Preferences | Editor | File Types for macOS

Ctrl+Alt+S



Use this page to manage the list of file types and extension patterns to be recognized by PyCharm.

ItemKeyboardDescription Shortcut

Recognized file types This list box displays all the default and custom file types currently supported by PyCharm. Refer to the section [File Types Recognized by PyCharm](#).

Use the Add, Edit, and Remove buttons to manage the contents of the list box.

Tip Default types cannot be edited or removed.

+  Click this button to open the [New File Type dialog](#) and define a new custom file type there.

  Click this button to open the [Edit File Type dialog](#) box and edit the selected file type there. This button is disabled when a default file type is selected.

-  Click this button to delete the selected file type from the list. This button is disabled when a default file type is selected.

Registered Patterns Shown in this area are all the registered extensions associated with the file type selected in the Recognized file types list. Use **+**,  and **-** to manage the corresponding patterns.

+  Use this icon or shortcut to open the Add wildcard dialog box and specify a new pattern using wildcards there.

  Use this icon or shortcut to edit the selected pattern.

-  Use this icon or shortcut to remove the selected pattern from the list.

Ignore files and folders In this text box, specify the files and folders, which you want to be ignored by PyCharm. Such files and folders will be completely excluded from any kind of processing. By default the list includes temporary files, service files related to version control systems, etc. You can specify multiple names or wildcard masks, with semicolons (;) as separators.

Below is the default setting, in case you need to restore it:

```
CVS;SCCS;RCS;rcs;.DS_Store;.svn;.pyc;.pyo;*.pyc;*.pyo;.git;*.hprof;.svn;.hg;*.lib;*~;__pycache__; .bundle;*.rbc;*.py.class;
```

Ctrl+Alt+S



The dialog box opens when you click the Add button or select a custom file type and click the Edit button on the [File Types page](#).

Use the dialog box to configure and re-configure presentation and highlighting of keywords, comments, numbers etc. in files of a specific custom file type. These settings make the basis for parsing files of this type in the editor.

ItemDescription

Name	In this text box, specify the name of the file type.
Description	In this text box, provide an optional description of the file type.
Syntax Highlighting	In this area, specify the character strings to indicate borders of comments, the numeric system used, and configure highlighting for brackets, braces, etc. syntax elements.
Line comment	In this text box, specify the character string to indicate the start of a single-line comment.
Only at line start	If this check box is selected, the character string that denotes the beginning of a line comment, is recognized as a comment if located in the first position of a line. This check box becomes available if at least one character is inserted in the Line comment field.
Block comment start	In this text box, specify the character string to indicate the start of a block comment.
Block comment end	In this text box, specify the character string to indicate the end of a block comment.
Hex prefix	In this text box, specify the character string to indicate that the subsequent value is a hexadecimal number. For example, <code>0x</code> .
Number postfixes	In this text box, specify the character string to indicate which numeric system or unit is used. A postfix is a trailing string of characters, for example, <code>e-3</code> , <code>kg</code> etc.
Support paired braces	Select this check box, to have paired braces highlighted.
Support paired brackets	Select this check box, to have paired brackets highlighted.
Support paired parens	Select this check box, to have paired parentheses highlighted.
Support string escapes	Select this check box, to have string escapes highlighted.
Ignore case	Select this check box to have PyCharm ignore case when processing file type extensions.
Keywords	Use this area to flexibly configure highlighting of keywords by grouping them into sets and associating each set with its own highlighting scheme . The area consists of 4 tabs. In each tab, create a set of keywords using the Add and Remove buttons. To associate a keyword set 1-4 with a highlighting scheme, edit the corresponding Keyword1 - Keyword4 property on the Custom page of the Colors and Fonts dialog box.
+	Click this button to open the Add a new keyword dialog box and define the new keyword there.
-	Click this button to delete the selected keyword from the list.

Emmet

File | Settings | Editor | Emmet for Windows and Linux

PyCharm | Preferences | Editor | Emmet for macOS

Ctrl+Alt+S



On the nested pages, enable PyCharm to use Emmet in [HTML](#), [XML](#), [JavaScript \(JSX Harmony\)](#) files and [style sheets](#).

ItemDescription

Expand abbreviation with

Use this drop-down box to select the default key to expand Emmet selectors with.
This key will also by default expand [Emmet live templates](#).

Note PyCharm expands abbreviations only if their output does not exceed 15 KB.

for macOS

Ctrl+Alt+S



ItemDescription

Enable CSS Emmet	Select this check box to enable Emmet support for style sheets. If this check box is not selected, the complicated abbreviations, like <code>bdci:n</code> expanding into <code>border-corner-image: none;</code> , will not work in the editor.
Enable fuzzy search among CSS abbreviations	When this check box is selected, every unknown abbreviation will be scored against available template names. The match with the best score will be used to resolve the template. For example, with this option enabled, the following abbreviations can be equal to: <ul style="list-style-type: none">- <code>ov:h</code>- <code>ov-h</code>- <code>o-h</code>- <code>oh</code>
Enable expansion of unknown properties ('unknown' to 'unknown;')	<ul style="list-style-type: none">- When this check box is selected, any entered word will be expanded into the same word followed with a colon and a semicolon;- When this check box is cleared, only known properties (for example, <code>color</code>) will be expanded this way (<code>color;</code>)
Auto insert CSS vendor prefixes	If this check box is selected, the CSS properties listed in the table below are expanded into constructs that contain pre-pending vendor prefixes. Learn more at Vendor prefixes . If this check box is cleared, the entire table of properties is disabled.
Properties and vendor prefixes	The table contains a list of CSS properties and vendor prefixes that correspond to various browsers. <ul style="list-style-type: none">- To enable or disable a property in a browser, select or clear the check box under the browser column.- To add a new property to the list, click the Add button <code>+</code> or press <code>Alt+Insert</code>. Then type the name of the property in the dialog box that opens and enable it in the relevant browsers.- To delete one or more properties from the list, select them and press Remove <code>-</code> or press <code>Alt+Delete</code>.

Ctrl+Alt+S

**ItemDescription**

Enable XML/HTML Emmet Select this check box to enable Emmet support for XML and HTML. If this check box is not selected, complicated abbreviations, such as `div.class>ul#list>.item$)` and similar, will not work in the editor.

Enable abbreviation preview Select this check box to have PyCharm show a pop-up window with a preview of the entered abbreviation before actually expanding it .

```
table#users>tr.user>td
<table id="users"><tr class="user"><td></td></tr></table>
Ctrl+Down and Ctrl+Up will move caret down and up in the editor >>
```

Enable automatic URL recognition while wrapping text with <a> tag - If this check box is cleared and you attempt to wrap an URL address with the <a> tag, PyCharm simply encloses the URL address in `` and positions the cursor inside the double quotes in the `href` attribute. For example, wrapping `http://www.jetbrains.com` will result in `http://www.jetbrains.com` :

```
<a href="">http://www.jetbrains.com</a>
```

- If this check box is selected and you attempt to wrap an URL address with the <a> tag, PyCharm inserts the URL address inside the double quotes as the value of the `href` attribute and encloses the URL in `<a href="<wrapped URL">` . For example, wrapping `http://www.jetbrains.com` will result in `http://www.jetbrains.com` . Moreover, PyCharm highlights the wrapped URL green as a recognized URL:

```
<a href="http://www.jetbrains.com">http://www.jetbrains.com</a>
```

Add edit point at the end of template If this check box is selected, an editing position adds to the end of an HTML template (\$END\$);

if this check box is not selected, then the new edit point is not added.

Compare the following:

```
<tr>
  <td></td>
  <td></td>
</tr>
<tr>
  <td></td>
  <td></td>
</tr>
```

BEM In this area, specify the BEM separators for the class names, modifiers and short elements. Refer to the [Emmet documentation](#) for details.

Filters enabled by default In this area, specify which Emmet filters you want to be applied to an expanded abbreviation before it is shown in the editor. Learn more about filters at <http://docs.emmet.io/filters/>. To have a filter always applied by default, select the check box next to it. The available options are:

- [XSL tuning](#)
- [Comment tags](#)
- [Escape](#)
- [Single line](#)
- [BEM](#)
- [Trim line markers](#)

Emmet. JSX

File | Settings | Editor | Emmet - JSX for Windows and Linux

PyCharm | Preferences | Editor | Emmet - JSX for macOS

Ctrl+Alt+S



ItemDescription

Enable JSX Emmet Select this check box to use Emmet within XML fragments in the [JSX Harmony](#) context if you have chosen the JSX language level on the [JavaScript](#) page of the Settings dialog box.

Images

File | Settings | Editor | Images for Windows and Linux

PyCharm | Preferences | Editor | Images for macOS

Ctrl+Alt+S



Specify how you want images to be shown in PyCharm. Optionally, specify an external editor for working with images.

ItemDescription

Editor	Use this area to specify the settings according to which images should be displayed in PyCharm.
Show Grid lines by default	Select this check box to have a grid displayed when reviewing image files. Tip When the check box is selected, the area below is enabled where you can define how to display a grid and its elements.
Show Grid lines only when zoom factor equal or more than	Use this spin box to specify the minimum zoom factor to have a grid displayed.
Show Grid line after every (pixels)	Use this spin box to specify the number of pixels between a pair of grid lines.
Grid line color	From this palette, click the colour to display grid lines.
Show transparency chessboard by default	Select this check box to have transparent pieces of images shown as a chessboard.
Chessboard cell size	Use this spin box to specify the chessboard cell size.
Color of 'white' cell	From this palette, click the colour to display chessboard cells located at initially 'white' positions.
Color of 'black' cell	From this palette, click the colour to display chessboard cells located at initially 'black' positions.
Zoom image with mouse wheel (Ctrl+Mouse Wheel)	Select this check box to enable zooming the image through the Ctrl+Mouse wheel combination.
Enable smart zooming for small images	Select this check box to have small images opened with the zooming factor to the size specified below.
Preferred minimum width/height for smart zooming (pixels)	Use this spin box to set the minimum number of pixels to zoom the small images to.
External Editor	In this area, specify an external editor for working with images.
Executable path	In this text box, specify the path to the executable image editor file.

Intentions

File | Settings | Editor | Intentions for Windows and Linux

PyCharm | Preferences | Editor | Intentions for macOS

Ctrl+Alt+S



Use this page to enable and disable [intention actions](#).

In this topic:

- [Intention List](#)
- [Toolbar and Controls](#)

Intention List

The list shows all intention actions currently available in PyCharm.

The intention actions are grouped by languages. To enable an intention action, select the check box to its left.

Toolbar and Controls

Item **Tooltip** **Description**
and
shortcut

	Ctrl+NumPad Plus	Expand all nodes in the intention list.
	Ctrl+NumPad -	Collapse all nodes in the intention list.
		Use this text box to search through the list of intention actions. As you type a search string, the intention actions that match the search pattern are displayed. To finalize the search, press <code>Enter</code> . The previously used search patterns are stored in the search history list.
		Click this button to clear the search history list.
Description		This read-only field shows the description of the selected intention action.
Usage examples		This area illustrates the effect of applying the selected intention action through the following fields: <ul style="list-style-type: none">- Before - this read-only field shows an example of source code before applying the selected intention action.- After - this read-only field shows the result of applying the selected intention action to the above example of source code.

Language Injections

File | Settings | Editor | Language Injections for Windows and Linux

PyCharm | Preferences | Editor | Language Injections for macOS

For the Language Injections page to be available, the IntelliLang [plugin](#) must be enabled. (This plugin is bundled with the IDE and enabled by default.)

Use this page to manage the list of available language injections and to configure the language injection feature for text, attributes, and parameters.

In this section:

- Language Injections
 - [Injection entries](#)
 - [Toolbar](#)
- [Language Injection Settings dialog: Sql Type Injection](#)

See also, [Using Language Injections](#).

Injection entries

You can sort the information by any of the columns by clicking the cells in the header row. The current sorting status is shown by the corresponding sorting marker: ▲ for the ascending order or ▼ for the descending order.

ItemDescription

Check boxes Use the check boxes to enable or disable the corresponding injections. You can also enable or disable a number of injections at once. To do that, select the required injections in the list and click [Enable Selected Injections](#)  or [Disable Selected Injections](#)  on the toolbar.

Name The language in which the injection is available, the injection name and, in parentheses, the package that contains the corresponding implementation.

Language The injected language.

Scope One of the following:

- Built-in. This is a category for pre-defined injections. In terms of the scope, those are the IDE-level injections.
- IDE. User-defined injections that are available in all of your projects.
- Project. User-defined injections that are available only in the current project.

You can move the user-defined injections between the IDE and the project levels by using  on the toolbar.

Toolbar

ItemDescription

 Create a new injection entry. Select the injection category and then specify the injection settings in the dialog that opens.

 Remove the selected entries from the list.

 Edit the settings for the selected injection.
IMPORTANT: Don't edit the settings for built-in injections.

 Create a copy of the selected injection entry. Then edit the settings for that copy as necessary.

 Enable all the injections currently selected in the list.

 Disable all the injections currently selected in the list.

 Move the selected injections between the IDE and the project levels. See also, [Scope](#).

 Import injection entries from another PyCharm installation:

1. In the [Select Path dialog](#), select the `IntelliLang.xml` file to import the info from. As a result, a dialog opens that shows the entries contained in the selected configuration file.
2. Remove the entries that you don't want to import using the Delete button. (This doesn't affect the contents of the source configuration file.)

This selective import feature makes it easy to share certain configurations in a team without losing any local entries as it happens when the settings are imported via the core [Importing Settings](#) feature.

 Export the selected injection entries to a file. In the [Export Selected Injections to File dialog](#) that opens:

- To add the entries to an existing file, select the destination file.
- To save the entries in a new file, specify the file name and choose the file type from the list.

Language Injection Settings dialog: Sql Type Injection

File | Settings | Editor | Language Injections | + | Sql Type Injection for Windows and Linux

PyCharm | Preferences | Editor | Language Injections | + | Sql Type Injection for macOS

Specify a data type pattern and associated injection language. See also, [Using pattern-based injections for user-defined data types](#).

ItemDescription

Type pattern A regular expression pattern for a data type in your SQL code. E.g. `(?i).*DATA` would be a case-insensitive pattern for data types ending in `data`. You can test your pattern: click  or press `Alt+Enter`, and select Check RegExp. Then type the text to be matched against the pattern in the Sample field.

Language The language to be injected into a string value of the corresponding type.

- ID. The language ID or name.
- Prefix. A sequence of characters to be added before the corresponding string value.
- Suffix. A sequence of characters to be added after the corresponding string value.

The prefix and suffix are optional. For more info, see [Using language injection prefixes and suffixes](#).

Spelling

File | Settings | Editor | Spelling for Windows and Linux

PyCharm | Preferences | Editor | Spelling for macOS

Ctrl+Alt+S



Use this settings page to create your own spelling dictionaries and thus expand the basic spelling support provided by PyCharm by default.

On this page:

- [Accepted Words Tab](#)
- [Dictionaries Tab](#)

Accepted Words Tab

Use this tab to configure the list of words that should be skipped by the **Typo** inspection.

Item	Tooltip	Description
------	---------	-------------

		Click this icon to open the Add New Word dialog box and specify a new entry there. CamelCase or snake_case are not supported. If you try to add a word that is already included in one of the spelling dictionaries, PyCharm displays an error message The word <just typed word> is already in the dictionary.
--	--	--

		Click this button to delete the selected item from the list.
--	--	--

Dictionaries Tab

Use this tab to configure the dictionaries to be used for spellchecking.

Item	Description
------	-------------

Dictionaries	This area in the bottom of the page shows a list of the dictionaries that can be used in spellchecking. The list contains the dictionaries that come bundled with PyCharm by default and user-defined dictionaries detected in the folders from the Custom Dictionaries Folder area above. <ul style="list-style-type: none">– To have a default dictionary applied in the current project, select the check box next to it.– To exclude a default dictionary from spellchecking within the scope of the current project, clear the check box next to it.
--------------	--

Custom Dictionaries Folder	This area displays a list of directories that contain user-defined dictionary files (text files with the <code>.dic</code> extension, containing words separated with a newline). <ul style="list-style-type: none">– To add a new folder to the list, click and choose the required folder in the Select Path Dialog dialog that opens . The full path to the folder is added to the Custom Dictionaries Folder list, and all the <code>*.dic</code> files found in this folder are added to the Dictionaries list.– To remove a folder from the list, select it and click .
----------------------------	---

TextMate Bundles

File | Settings | Editor | TextMate Bundles for Windows and Linux

PyCharm | Preferences | Editor | TextMate Bundles for macOS

Ctrl+Alt+S



Use this page to import the [TextMate/SublimeText 2 bundles](#), and to map color scheme of PyCharm to that of TextMate.

Prerequisites

This page appears in the Settings/Preferences dialog, when the TextMate bundle support plugin is enabled.

The plugin is bundled with PyCharm and is activated by default. If it is disabled, you can manually [enable the plugin](#).

ItemDescription

TestMate Bundles

This table of added bundles consists of the following columns:

- Check box: If a check box to the left of the added bundle name is selected, the bundle provides highlighting in the files of the corresponding type.
- Name of the TextMate Bundle
- Bundle location: for each added TextMate bundle, its location is shown.



Click this button to locate the desired bundle using the [Select Path](#) dialog box. When added, the bundle appears in the table of TextMate Bundles.

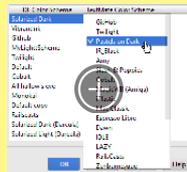


Click this button to remove the selected bundle. The bundle in question is removed from the list, but is not physically deleted from the computer.

IDE Color Scheme - TextMate
Color Scheme

Use this table to establish mapping between the various color schemes of PyCharm and TextMate.

Tip If you want to use a custom TextMate color scheme, you can import a TextMate bundle with schemes, and it will become visible in the list of TextMate color schemes after clicking Apply.



TODO

File | Settings | Editor | TODO for Windows and Linux

PyCharm | Preferences | Editor | TODO for macOS

Ctrl+Alt+S



In this page, configure filters to maintain your lists of TODO items in the [TODO tool window](#) and define TODO patterns to be used in filters.

Patterns

In this area, create and manage the list of available TODO patterns.

ItemDescription

Icon This read-only field displays the icon that is assigned to the current pattern. (This icon appears in the TODO tool window as a marker for the corresponding TODO items.)

Case sensitive This read-only check box indicates whether the current pattern is case sensitive or not. The status is changed in the [Edit Pattern](#) dialog box.

Pattern This read-only field displays the regular expression that describes the TODO pattern. PyCharm recognizes regular expressions in the source code against the specified patterns and displays them in the TODO tool window.

+ or Use this icon or shortcut to open the [Add Pattern](#) dialog box, where you can create a new ToDo pattern by specifying a regular expression.

Alt+Insert

or Use this icon or shortcut to open the [Edit Pattern](#) dialog box, where you can modify the selected pattern.

Enter

- or Use this icon or shortcut to remove the selected pattern from the list.

Alt+Delete

Filters

In this area, manage the list of available filters.

ItemDescription

Name This read-only field shows a list of filter names.

Patterns This read-only field shows the names of patterns included in the current filter. A filter can contain several patterns.

+ or Use this icon or shortcut to open the [Add Filter](#) dialog box, where you can define a new filter.

Alt+Insert

or Use this icon or shortcut to open the [Edit Filter](#) dialog box, where you can edit the settings for the selected filter.

Enter

- or Use this icon or shortcut to delete the selected filter.

Alt+Delete

Ctrl+Alt+S



Use this dialog box to define patterns that help track TODO items in your source code. Make sure the TODO items in the source code are inserted inside comments that are valid for the supported file types.

ItemDescription

Pattern	In this text box, type the regular expression that describes the desired TODO pattern.
Icon	From this drop-down list, choose an icon for the pattern.
Case sensitive	Select this check-box to make the pattern case-sensitive.
Font type	Select the corresponding check box to have the icon text displayed in bold or italic.
Foreground	Select this check box to enable the palette and choose the foreground color.
Background	Select this check box to enable the palette and choose the background color.
Error Stripe Mark	Select this check box to enable the palette and choose the error stripe color.
Effects	Select this check box to enable effects (underscore, strikethrough, etc.) and choose the color for them from the palette.

Add / Edit Filter Dialog

File | Settings | TODO - Add/Edit Filter for Windows and Linux

PyCharm | Preferences | TODO - Add/Edit Filter for macOS

Ctrl+Alt+S



Use this dialog box to define filters that help track TODO items in your source code.

ItemDescription

Name	In this text box, specify the name of the filter.
Patterns	From the list of available patterns, choose the ones to be included in the filter by selecting the check boxes next to them.

Plugins

File | Settings | Plugins for Windows and Linux

PyCharm | Preferences | Plugins for macOS

Ctrl+Alt+S



The Plugins page shows the list of installed [plugins](#). Use the check boxes next to plugin names to enable or disable them.

Other controls on this page let you sort and filter the plugin list, update and uninstall [repository plugins](#), access [plugin repositories](#), and also install the plugins available locally.

- [Main controls](#)
- [Context menu commands](#)
- [Colors for plugin statuses](#)

Main controls

ItemDescription



Type the text to be found.

As you type, the list of plugins changes: only the plugins whose names and descriptions contain the specified text are shown.

To bring the plugin list back to its initial state, delete the text in the search box or click

To access the list of memorized search strings, click

Show

Use this list to specify which plugins should be shown:

- All plugins
- Enabled
- Disabled
- Bundled
- Custom (these are the plugins that are not bundled with the IDE).



If you want to add sorting by plugin status, click and select Status. As a result, disabled plugins will be displayed at the end of the list. This option is also available from the context menu.

Install JetBrains plugin

Click this button to open the [Browse JetBrains Plugins dialog](#) to download and install plugins from the JetBrains repository.

Browse repositories

Click this button to open the [Browse Repositories dialog](#) to work with plugin repositories (to download and install repository plugins, manage enterprise plugin repositories, etc.).

Install plugin from disk

Click this button to install a plugin available locally. Select the file that implements the required plugin in the [dialog that opens](#).

Context menu commands

CommandDescription

Reload List of Plugins

Use this command to check if newer versions of installed plugins are available (see [Colors for plugin statuses](#)).

Sort by | Status

Use this command to sort plugins by their status.

When this option is turned on, disabled plugins are shown at the end of the list, after the enabled plugins.

Update Plugin

For repository plugins: use this command to download and install a newer version of the selected plugin (if available). See [Colors for plugin statuses](#).

Uninstall

For plugins that are not bundled with the IDE: use this command to uninstall the selected plugin. Alternatively, the Uninstall button in the plugin description area can be used.

Colors for plugin statuses

The names of plugins are shown in different colors depending on their status.

ColorPlugin status

Black "Normal" plugin status. For a repository plugin: the plugin version is up-to-date.

Red One of the following:

- The plugin is incompatible with the installed version of PyCharm.
- The plugin depends on another plugin which is disabled.

Blue For a repository plugin: a newer version of the plugin is available.

Green For a repository plugin: the plugin has been downloaded and installed, but has not been activated yet (PyCharm needs to be restarted).

Gray The plugin has been uninstalled, but the changes have not taken effect yet (PyCharm needs to be restarted).

Browse Repositories Dialog

This dialog opens when you click the Browse repositories button on the [Plugins page](#).

This dialog shows a list of [repository plugins](#).

You can sort and filter the plugins list, download and install plugins, manage [enterprise plugin repositories](#), and configure HTTP proxy settings.

- [Main controls](#)
- [Context menu commands](#)

Main controls

ItemDescription

	Type the search string. As you type, the list of plugins changes: only the plugins whose names and descriptions contain the specified string are displayed. If you want PyCharm to remember the search string, press  . To bring the plugin list back to its initial state, delete the text in the search box or click  . To access the list of memorized search strings, click  .
	Use this icon to update the list of plugins and their statuses.
	If you have enterprise plugin repositories configured, there is a list that provides repository-based filtering. You can select to see the contents of all repositories or only the selected one.
	This list provides category-based filtering. You can select to see all plugins, or the plugins that belong to the selected category.
Sort by	Click  and select a category.
Install plugin	Click the Install plugin button in the description area to download and install the selected plugin.
HTTP Proxy Settings	If you access the Internet via an HTTP proxy, click this button and specify the HTTP proxy settings .
Manage repositories	Click this button to create or edit the list of enterprise plugin repositories in the Custom Plugin Repositories dialog .

Context menu commands

CommandDescription

Reload List of Plugins	Use this command to update the list of plugins and their statuses.
Sort by <Category>	Use this command to enable or cancel sorting. The following categories are available: <ul style="list-style-type: none">- Status- Downloads- Rating- Last Updated
Download and Install	Use this command to download and install the selected plugin.

Custom Plugin Repositories Dialog

This dialog opens when you click the Manage repositories button in the [Browse Repositories dialog](#).

Use this dialog to manage the list of [enterprise plugin repositories](#). The repositories are identified by their [URLs](#).

IconShortcutDescription

		Use this icon or shortcut to add a new repository to the list. Specify the URL of the repository in the dialog that opens. Use the Check Now button to make sure that the specified URL is correct: PyCharm will try to connect to the repository.
		Use this icon or shortcut to edit the selected URL.
		Use this icon or shortcut to remove the selected URL from the list.

Version Control

File | Settings | Version Control for Windows and Linux

PyCharm | Preferences | Version Control for macOS

Ctrl+Alt+S



The settings under this node allow configuring integration with different version control systems.

[Common settings](#) that are applied to the project files regardless of which version control system is used:

- [Confirmation](#)
- [Background](#)
- [Ignored Files](#)
- [Issue Navigation](#)
- [Changelist Conflicts](#)

The settings for configuring integration with a specific version control system are located under the following nodes:

- [GitHub](#)
- [CVS](#)
- [Git](#)
- [Mercurial](#)
- [Perforce](#)
- [Subversion](#)

Specify which version control systems will be used for specific directories, or the entire project.

ItemDescription

Directory	<p>This field shows the path to project directories or the project root(s).</p> <p>For projects with Git integration enabled, PyCharm scans project directories to check if there are Git repositories that are not controlled by the IDE. If such repositories are found, they are listed here under Unregistered roots and are marked grey. To add an unregistered root, select it in the list and click the Add button +.</p> <p>PyCharm also checks if registered roots are valid, i.e. that a Git repository exists at the specified path. If invalid repositories are detected, they are marked with red.</p>
VCS	<p>Select a version control system for the specified directory.</p> <p>The list only displays the version control systems for which the corresponding plugins are enabled (See Managing Plugins).</p>
	<p>Click this button to open the Version Control Configurations dialog and update the configuration settings for the selected VCS.</p>
	<p>Click this button to add a directory mapping to the list. The Add VCS Directory Mapping dialog box opens where you can specify the required directory, select a VCS for it, and open the Version Control Configurations dialog box to configure the specified VCS, if necessary.</p>
	<p>Click this button to edit the selected directory mapping. The Edit VCS Directory Mapping dialog box opens where you can update the selected mapping and configure the specified VCS, if necessary.</p>
	<p>Click this button to remove the selected directory mapping from the list.</p>
Limit history to	<p>Select this check box to specify the number of history rows. If this check box is selected, the text box of history depth, and the spin box become enabled.</p>
Show directories with changed descendants	<p>If this check box is selected, the directories that contain changes, are color-marked.</p> <p>The colors are configurable in the Colors and Fonts pages of the Editor settings (File Status - Have immediate changed children, Have changed descendants).</p>
Shelve base revisions of files under distributed version control systems.	<p>This option is relevant only for Git and Mercurial. Select this check box to automatically shelve base revisions of files that are under Git or Mercurial version control.</p> <p>By default, PyCharm always "remembers" the last commit hash. However, this information is not sufficient if the history has been changed since the last commit as a result of running the rebase operation. In this case, having a copy of the base revision may help.</p>
Show changed in last <number> days	<p>Select this check box to have color indication of file status applied during stacktrace analysis and debugging. The names of the files that have been changed within a certain period will be highlighted accordingly.</p> <p>Specify the number of days.</p>
Filter Update Project information by scope	<p>If this option is enabled and a scope is selected, the files that belong to this scope will be marked in bold in the Update Project Info tab of the Version Control Tool Window. If you click the Filter button  in the toolbar in the Update Project Info tab, the files will be filtered by scope, i.e. only the files that belong to the selected scope will be displayed.</p> <p>Click the Manage Scopes link to open the Scopes settings dialog and configure a scope.</p>
Commit message right margin (columns)	<p>In this field, specify the number of symbols that can fit into the right margin of the Commit Changes dialog.</p> <p>Select the Wrap when typing reaches right margin option if you want the text to be transferred to the next line when the maximum number of characters has been reached.</p>
Show unversioned files in Commit dialog	<p>Select this option to see newly added files that have not been added to version control yet under the Unversioned Files node in the Commit Changes dialog.</p>

Check commit message
spelling

Select this check box if you want to automatically check spelling of your commit messages.

Confirmation

File | Settings | Version Control | Confirmation for Windows and Linux

PyCharm | Preferences | Version Control | Confirmation for macOS

Ctrl+Alt+S



In this page, specify whether you want PyCharm to ask you for confirmation before performing specific version control related actions.

ItemDescription

When files are created

In this section, specify whether and how to put a file created in PyCharm under version control. The following options are available:

- Show options before adding to version control: When this option is selected, newly created files are put under version control after you specify the version control options in the dialog box that opens.
- Add silently: When this option is selected, newly created files are automatically put under version control without displaying any messages.
- Do not add: When this option is selected, newly created files remain unversioned and you can put them under version control later.

When files are deleted

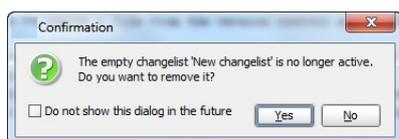
In this section, specify whether and how to remove a file from the version control system when the file is removed in PyCharm. The following options are available:

- Show options before removing from version control: When this option is selected, locally removed files are also removed from the specified VCS but first a dialog box for selecting version control options is displayed.
- Remove silently: When this option is selected, all locally removed files are removed from the specified VCS without asking for confirmation.
- Do not remove: When this option is selected, locally removed files remain under version control.

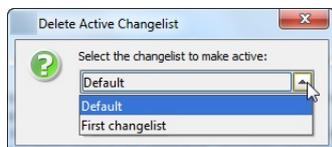
When empty changelist becomes inactive

In this section, specify PyCharm behavior on deleting an empty changelist.

- Show options before removing: When this option is selected, PyCharm asks for confirmation before removing an empty changelist that has lost its active status:



If you choose to delete such a changelist, PyCharm suggests to choose another changelist to be marked as active.



- Remove silently: When this option is selected, PyCharm automatically deletes empty changelists that become inactive, except for the Default changelist.
- Do not remove: When this option is selected, empty changelist is not deleted on losing its active status.

Display Option dialogs when these commands are invoked

In this area, specify whether you want PyCharm to ask you for confirmation when invoking commands, specified below. To enable showing a prompt before performing an action, select the relevant check boxes.

Tip – Availability of commands depends on the particular version control system.
– Use controls in this area to restore the original settings, if you have suppressed showing operation-specific option dialog boxes by selecting the Do not show this dialog in the future check box.

Show Clear Read-Only Status dialog box

Select this check box to have PyCharm explicitly require cancellation of the read-only status when you open a file in the editor and try to modify it.

The read-only status can be cleared in two ways:

- Using the current VCS: the file is added to a [changelist](#).
- Using a file system: the file is not added to the changelist.

This option is relevant for those version control systems that separate check-out from opening for editing.

Suggest to move uncommitted changes to another changelist

When this option is selected, on committing a changelist to the repository with some files excluded from the commit, PyCharm prompts to move these files to another changelist.

Force non-empty check-in comments

Select this check box to suppress committing changes without supplying them with corresponding comments.

Clear initial commit message

If this check box is selected, the previous check-in comment is cleared from the Comment area.

Show patch in explorer after creation

Use this drop-down list to define the PyCharm behavior when a [patch is created](#). The available options are:

- Yes: if this option is selected, native file manager always opens to show the patch file.
- No: if this option is selected, the native file manager will not open.
- Ask: if this option is selected, PyCharm will display a dialog box informing about successful patch creation, and a suggestion to locate the patch file in the native file manager.

Create changelist on failed commit

Use this drop-down list to define the PyCharm behavior in case of failed commits. The available options are:

- Yes: if this option is selected, the Failed commit changelist will be automatically created, and the respective files will be moved to this changelist.
- No: if this option is selected, the files that failed to commit will remain in their original changelist.
- Ask: if this option is selected, on failed commit PyCharm will display a dialog box asking whether the files should be moved to another changelist, or remain in the original one.

See the section [Resolving Conflicts](#).

Background

File | Settings | Version Control | Background for Windows and Linux

PyCharm | Preferences | Version Control | Background for macOS

Ctrl+Alt+S



In this page, enable performing specific version control related operations in the background without any activities from your side.

ItemDescription

Background Operations	<p>In this area, specify the version control related operations to be performed in the background. To enable running an operation in the background, select the relevant check box:</p> <ul style="list-style-type: none">– Perform update from VCS in background– Perform commit to VCS in background– Perform checkout from VCS in background– Perform Edit/Checkout in background– Perform Add/Remove in background– Perform revert in background
-----------------------	---

Changelists to cache initially	Use this spin box to specify the number of changelists to be stored in the cache.
--------------------------------	---

Refresh changes every ... minutes	<p>Use this spin box to specify how often the VCS should check for new changes and refresh the cache.</p> <p>The spin box is only enabled when the Enable background processes check box is selected.</p>
-----------------------------------	---

"Changed on server" conflicts	<p>In this area, specify whether you want PyCharm to check whether a checked out files have been updated by someone else. To enable background synchronization with the server, select the Check every ... minutes. In the spin box, specify how often you want synchronization to take place.</p> <p>This area is only enabled when the Enable background processes check box is selected.</p>
-------------------------------	---

Ignored Files

File | Settings | Version Control | Ignored Files for Windows and Linux

PyCharm | Preferences | Version Control | Ignored Files for macOS

Ctrl+Alt+S



Use this dialog to configure a list of files and directories that you do not want to put under version control. These can be file names associated with VCS administration, backup files, and any other artifacts that you want to remain unversioned. You can also specify [patterns](#) of files you want to ignore.

Tip You can only ignore unversioned files, i.e. files that have not yet been put under version control.

ItemKeyboardDescription shortcut

		Use this icon or shortcut to add an item to the list. The Ignore Unversioned Files dialog box opens where you can type an exact path to a file or directory to be ignored or specify a pattern that defines the names of files and directories to be ignored.
		Use this icon or shortcut to edit the selected path or pattern in the Ignore Unversioned Files dialog box.
		Use this icon or shortcut to remove the selected path or pattern from the list.

Patterns

Two characters can be used as wildcards:

-  : to replace any string.
-  : to replace a single character.

For example, `*.iml` will ignore all files with the `iml` extension; `*.?m1` will ignore all files whose extension ends with `m1`.

Ctrl+Alt+S



The dialog box opens when you click the Add  or Edit  button on the [Ignored Files](#) page.

Use this dialog to configure rules that define which files and folders should be ignored by version control systems. The files you want to ignore can be appointed explicitly by their names or through name patterns with wildcards. To ignore a directory, you need to specify the full path to it relative to the project root.

Tip You can only ignore unversioned files, i.e. files that have not yet been put under version control.

Select the relevant option and fill in the text box next to it.

ItemDescription

Ignore specified file	<p>In this text box, specify the name of the file to be ignored. Do one of the following:</p> <ul style="list-style-type: none">– Type the file name relative to the project root, for example, <code>my_folder/my_subfolder1/my_subfolder2/my_file</code>.– Click the Browse button  and select the desired file in the Select File to Ignore dialog box.
Ignore all files under	<p>In this text box, specify the name of the directory to be ignored. Do one of the following:</p> <ul style="list-style-type: none">– Type the directory name relative to the project root, for example, <code>my_folder/my_subfolder1/</code>.– Click the Browse button  and select the desired folder in the Select Directory to Ignore dialog box. <p>The rule is applied recursively to all subdirectories of the specified directory. If a directory has several subdirectories and you want only one of them ignored, specify the required directory explicitly, for example, <code>my_folder/my_subfolder1/my_subfolder2/</code>.</p>
Ignore all files matching	<p>In this text box, specify a pattern that defines the names of files to ignore. The rule is applied to all the directories under the project root. Using wildcards in combination with slashes (<code>/</code>) to restrict the scope to a certain directory is not supported.</p>

Issue Navigation

File | Settings | Version Control | Issue Navigation for Windows and Linux

PyCharm | Preferences | Version Control | Issue Navigation for macOS

Ctrl+Alt+S



Use this dialog to create a list of the so-called **issue navigation patterns**. An **issue navigation pattern** maps an **issue ID pattern** in commit messages with the URL addresses of the referenced issues. This enables you to navigate from committed changes to issues related to these changes. As soon as PyCharm encounters a match to the issue ID pattern in a commit message, the match is displayed as a link in the **Version Control tool window**. If you mention several issues, all of them will be displayed as links. Clicking such link opens the matching issue in the default browser.

ItemDescription

Issue	This read-only field shows the issue pattern.
Link	This read-only field shows the link to navigate from the issue pattern in the current row to the issue in the bug tracking system.
	Click this button to create a new issue navigation pattern and link. The Add Issue Navigation Link dialog box opens where you can specify: <ul style="list-style-type: none">- A regular expression to define the issue ID.- A regular expression to define the navigation link to the issue.
Tip Refer to Regular Expressions Syntax Reference for details on using special characters in regular expressions.	
	Click this button to create a new JIRA pattern. The Create JIRA Issue Navigation Pattern dialog box is opened where you can specify the URL to your JIRA installation. The regular expression that defines the pattern is added automatically.
	Click this button to create a new pattern for YouTrack . In the dialog box that opens, specify the URL to your YouTrack installation. The regular expression that defines the pattern is added automatically.
	Click this button to update the selected issue navigation link.
	Click this button to remove the selected issue navigation link from the list.

Example

The example below shows how PyCharm applies the abovementioned rules to detect a reference to an issue in a commit message and compose a link to it in the issue tracking system.

Issue ID pattern	The regular expression that defines the format in which issues are referenced in commit messages.
<pre>[A-Z]+\-\d+</pre>	
This regular expressions matches all character strings that consist of two substrings separated by an n-dash character: <ol style="list-style-type: none">1. Substring 1: An unlimited number of upper case alphabetic characters.2. Substring 2: An unlimited number of digital characters.	
Issue link pattern	A combination of the URL address of your issue tracking system and a regular expression that identifies issues in it.
<pre>http://mytracker/issue/\$0</pre>	
Here <code>\$0</code> indicates a back reference to the entire match. This means that as soon as PyCharm detects a match in a commit message, it is added to the URL address of the tracker as is.	
Matching issue ID	PyCharm detects the following reference to an issue in the commit message of interest:
<pre>MYPROJECT-110</pre>	
Composed issue link	In accordance with the above issue navigation pattern, the detected matching reference is added to the URL of the tracker as is, so the link to the referenced issue is composed as follows:
<pre>http://mytracker/issue/MYPROJECT-110</pre>	

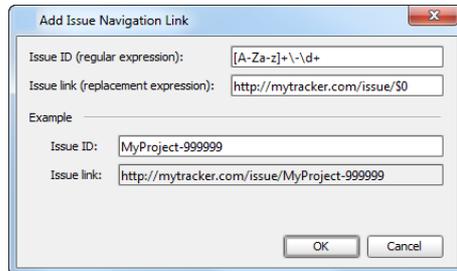
Use this dialog box to create an issue pattern and navigation link to a bug tracking system.

ItemDescription

Issue ID (regular expression) In this field, type a regular expression that will be converted to a specific issue id.

Issue link (replacement expression) In this field, type a regular expression that will be converted to a navigation link to the issue.

Example In this section, type some specific issue id to make sure it matches the specified pattern:



Changelist Conflicts

File | Settings | Version Control | Changelist Conflicts for Windows and Linux

PyCharm | Preferences | Version Control | Changelist Conflicts for macOS

Ctrl+Alt+S

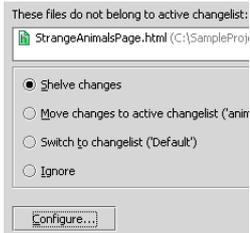


Use this page to configure protection of files belonging to inactive changelists, from occasional conflicts.

ItemDescription

Enable changelist conflict tracking If this check box is selected, PyCharm makes it possible to protect files in inactive changelists. Such protection can be performed on the various levels enabled by the options listed below.

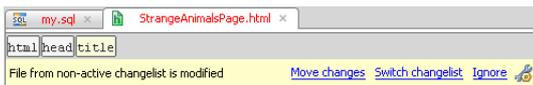
Show conflict resolving dialog If this check box is selected, PyCharm shows Resolve Changelist Conflict dialog box on an attempt to modify a file.



You need to define the way to resolve a conflict, before you proceed with changes. The possible ways to resolve a conflict are as follows:

- **Shelve** changes.
- **Move** file to the active changelist.
- **Switch** changelists to make the current changelist active.
- **Ignore** conflict. In this case, the file in question will be added to the list of files with ignored conflicts.

Highlight files with conflicts If this check box is selected, PyCharm shows a yellow stripe on top of a file from an inactive changelist, when such file has been modified.



In this stripe, you can choose one of the possible ways to resolve conflict:

- **Move** file to the active changelist.
- **Switch** changelists.
- **Ignore** conflict.

Names of the files belonging to inactive changelists are shown in red font in the editor tabs and in the Project view.

Highlight files from non-active changelists If this check box is selected, names of the files belonging to inactive changelists are shown in light-blue font in the editor tabs and in the Project view.

Files with ignored conflicts This area shows the list of files, for which Ignore option has been selected in the Resolve Changelist Conflict dialog box, or in the editor stripe.

Clear Click this button to remove all files from the list of files with ignored conflicts.

GitHub

File | Settings | Version Control | GitHub for Windows and Linux

PyCharm | Preferences | Version Control | GitHub for macOS

Ctrl+Alt+S



Use this page to specify your [GitHub](#) remote storage account credentials or create a GitHub account if you do not have one yet.

ItemDescription

Host	Specify the URL of your GitHub repository.
Auth Type	Use this drop-down list to select how you want to be authenticated on GitHub. The available options are: <ul style="list-style-type: none">– Password. If this option is selected and you have two-factor authentication enabled in your GitHub account settings, you will be asked to enter an authentication code each time PyCharm requires you to log in to your GitHub account.– Token (recommended by GitHub for authentication from third-party applications, as it does not require PyCharm to remember your password).
Login	In this text box, type your GitHub logon name. This field is only available if Password is selected as authentication method above.
Password	In this text box, type your GitHub account password. This field is only available if Password is selected as authentication method above.
Test	Click this button to verify the credentials you have specified.
Token	Specify your personal access token. This field is only available if Token is selected as authentication method above.
Create API Token	Click this button if you do not have a personal API token yet. Specify your GitHub credentials in the Login to GitHub dialog that opens and click the Login button. The token will be generated automatically.
Sign up	Click this link to open the Sign up for GitHub page where you can create a GitHub account.
Connection timeout	Specify the time period to wait for connection to be established.

CVS

File | Settings | Version Control | CVS for Windows and Linux

PyCharm | Preferences | Version Control | CVS for macOS

Ctrl+Alt+S



Use this page to specify the version control settings to be applied to the directories of your project that are under CVS control.

ItemDescription

Updating	<p>In this area, define the PyCharm behavior when merge conflicts occur during update from the server.</p> <ul style="list-style-type: none">– Show dialog - select this option to have PyCharm display a dialog box where you can examine, analyze, and resolve possible conflicts before updating.– Skip merging for all project or module files merged with conflicts - select this option to suppress updating files where conflicts occurred during merge.– Get latest repository versions silently - select this option to have your local files in question automatically updated with their latest repository versions.
Use read-only flag for not edited file	Select this check box to have the read-only status assigned to a file automatically after the check out, update, or commit operations.
Show CVS server output	Select this check box to have the server output of CVS commands displayed in the CVS Output tool window.
Default keyword substitution for text files	<p>Use this drop-down list to specify the keyword expansion mode. CVS uses the keyword substitution mode of a file to differentiate binary files from ASCII files and to indicate what type of keyword substitution is applied when files are committed and checked out.</p> <p>The available options are:</p> <ul style="list-style-type: none">– keyword&value (<code>-kkv</code>)– keyword, value&locker (<code>-kkv1</code>)– keyword only (<code>-kk</code>)– original string (<code>-ko</code>)– binary (<code>-kb</code>)– value only (<code>-kv</code>)
Global Settings	Click this button to open the Global CVS Settings dialog box.

Ctrl+Alt+S



Use this dialog box to set up CVS options at the global level. The dialog is available for files and directories that are under CVS control.

ItemDescription

Charset	From this drop-down list, select the character set to be used.
Use gzip compression	Select this check box to apply <code>gzip</code> compression.
Password file	In this text box, specify the fully qualified path to the <code>.cvspass</code> file. Click  to select the file in the corresponding dialog .
Connection timeout	In this text box, type the timeout value in seconds.
Send environment variable to server	Select this check box to have CVS-related environment variables sent to the server.
Log CVS client/server output to <code>cv.s.log</code> file	Select this check box to enable logging and have the <code>cv.s.log</code> log file stored in the <code>log</code> directory of your PyCharm installation.

Git

File | Settings | Version Control | Git for Windows and Linux

PyCharm | Preferences | Version Control | Git for macOS

Ctrl+Alt+S



Use this page to specify the version control settings that will be applied to the directories of your project that are under [Git](#) control.

ItemDescription

Path to Git executable	In this text box, specify the path to the Git executable file. Type the path manually or click the Browse button  and specify the path in the dialog that opens.
Test	Click this button to verify the path to the Git executable file.
SSH executable	Use this drop-down list to specify the SSH version to be used with Git. The available options are: <ul style="list-style-type: none">– Built-in: select this option to use the implementation provided by PyCharm.– Native: select this option to use native implementation. Note that on some platforms, the use of native ssh implementation may cause hang-up issues. You may need to configure a platform-specific <code>ssh-askpass</code> to receive prompts for passwords.
Control repositories synchronously	This option only becomes available if you have a multirooted project, i.e. there are several Git repositories within a single project. Select this option if you want branch operations (such as <code>checkout</code> , <code>merge</code> , etc.) to be applied synchronously to all repositories within your project.
Commit automatically on cherry-pick	When you cherry pick a specific commit, the Commit Changes dialog is displayed. If the Commit automatically on cherry-pick option is selected, the selected commit is submitted silently on clicking the cherry-pick button  , without displaying the Commit Changes dialog .
Warn if CRLF line separators are about to be committed	Select this option to enable smart handling of <code>LF</code> and <code>CRLF</code> line separators. PyCharm will analyze your configuration, warn you if you are about to commit CRLF into the repository, and suggest changing the <code>core.autocrlf</code> setting to <code>true</code> or <code>input</code> depending on your operating system. Note Note that this setting is not applied to files where you have set any related Git attributes . In this case, PyCharm assumes that you clearly understand what you are doing and excludes such files from analysis. If this option is deselected, you will have to fix issues with line endings manually using the Difference Viewer dialog .
Warn when committing in detached HEAD or during rebase	Select this option if you want PyCharm to display a warning when a commit is performed from a detached head or on rebase, as this may cause issues and code loss.
Update method	Use this drop-down list to choose the strategy to synchronize your local repository with the remote storage. The selected method will be used when the <code>push</code> operation is rejected (if the Auto-updated if push of the current branch was rejected option is enabled), or when you invoke the Update Project operation. The following options are available: <ul style="list-style-type: none">– Merge: choose this option to have the merge strategy applied. The result is identical with that of running <code>git fetch ; git merge</code> or <code>git pull -no-rebase</code>.– Rebase: choose this option to have the rebase strategy applied. The result is identical with that of running <code>git fetch ; git rebase</code> or <code>git pull --rebase</code>.– Branch Default: choose this option to have the default command for the branch applied. The default command is specified in the <code>branch.<name></code> section of the <code>..git/config</code> configuration file.
Auto-update if push of the current branch was rejected	Select this checkbox if you want the current branch to be updated automatically if the <code>push</code> operation from the current branch to its tracked branch is rejected. If this option is deselected, PyCharm will display the Push Rejected dialog when pushing a branch is rejected because your local repository and the remote storage are not synchronized. Note Note the following: <ul style="list-style-type: none">– If you have never seen the Push Rejected dialog box before and you are enabling the check box initially, PyCharm will update the conflicting local branch silently by means of the <code>merge</code> operation.– If you have already encountered the Push Rejected dialog box and selected the Remember the update method choice... option, PyCharm saves your last choice (<code>rebase</code> or <code>merge</code>) and will apply it to update the conflicting local branch silently. Accordingly, to change the “remembered” setting, clear the check box, access the Push Rejected dialog box, select the Auto-update if push ... rejected option, and invoke another update strategy.
Allow force push	If this check box is selected, the Force push option is added to the Push Commits dialog (as a drop-down option on the Push button).
Protected branches	If you have selected the Allow force push option , but want to disable it for certain branches, list them here (this is a team-shared parameter that is stored in <code>..idea/vcs.xml</code>). You can list several branches separated by a colon, or supply branch patterns as the input is treated as a list of regular expressions.

Mercurial

File | Settings | Version Control | Mercurial for Windows and Linux

PyCharm | Preferences | Version Control | Mercurial for macOS

Ctrl+Alt+S



Use this page to specify the version control settings to be applied to the directories of your project that are under [Mercurial](#) control.

ItemDescription

Path to hg executable	<p>Specify the location of the Mercurial executable file. Enter the path manually, or click the Browse button  and select the path in the dialog that opens.</p> <p>If you followed the standard installation procedure, the default location is <code>/opt/local/bin</code> or <code>/usr/local/bin</code> for Linux and macOS and <code>/Program Files/TortoiseHG</code> for Windows.</p> <p>It is recommended that you add the path to the Mercurial executable file to the <code>PATH</code> variable. In this case, you can specify only the executable name, the full path to the executable location is not required. For more information about environment variables, see Path Variables.</p>
Test	<p>Click this button to verify the path to the Mercurial executable.</p>
Control repositories synchronously	<p>This option only becomes available if you have a multirooted project, i.e. there are several Mercurial repositories within a single project. Select this option if you want branch operations (such as <code>checkout</code>, <code>merge</code>, etc.) to be applied synchronously to all repositories within your project.</p>
Check for incoming and outgoing changesets	<p>Select this option if you want PyCharm to detect incoming and outgoing changes in the background mode. PyCharm will automatically request the server for incoming and outgoing changesets every 5 minutes.</p>
Ignore whitespace differences in annotations	<p>Select this check box if you want white spaces to be ignored when annotating, and, thus, get more meaningful annotations and cast out senseless ones.</p>

Perforce

File | Settings | Version Control | Perforce for Windows and Linux

PyCharm | Preferences | Version Control | Perforce for macOS

Ctrl+Alt+S



Use this page to specify the version control settings to be applied to those directories of your project that are under Perforce control

ItemDescription

Perforce is online	Select this check box to work with Perforce in the online mode.
Switch to offline mode automatically if Perforce is unavailable	Select this check box to have PyCharm automatically go offline as soon as Perforce becomes unavailable and display the corresponding message.
Use P4CONFIG or default connection	If this option is selected, <code>P4CONFIG</code> or the default Perforce connection are used to connect to the Perforce server. Using <code>P4CONFIG</code> makes it trivial to switch between Perforce settings for different projects, when necessary.
Use connection parameters	If this option is selected, the connection credentials (port, client, user name, and charset) are specified manually.
Port	In this text box, type the server and the port which your Perforce client will listen to. For the default Perforce server configuration, it looks like <code>perforce:1666</code> .
Client	In this text box, type the name of your Perforce workspace name.
User	In this text box, type your user name to authenticate to the server.
Charset	From this drop-down list, select the character set to be used.
Dump Perforce commands to <path>	Select this check box to have PyCharm create a file <code>P4.output</code> and store the output of Perforce commands in it.
Use login authentication	When this check box is selected, Perforce requires a login to authenticate a user.
Test Connection	Click this button to check whether the specified settings ensure establishing connection to the Perforce server.
Path to P4 executable	In this text box, specify the path to the Perforce Command Line Client's executable file <code>P4</code> . Click the Browse button  to open the Select Path - P4 Configuration dialog box and select the executable file in the directories tree.
Path to P4V executable	In this text box, specify the path to the Perforce Visual Client's executable file <code>P4V</code> . Click the Browse button  to open the Select Path - P4 Configuration dialog box and select the executable file in the directories tree.
Show branching history ...	Select this check box to enable displaying the branch history of a specified file, including all file branch points, edits, and merges.
Show integrated changelists in committed changes	Select this check box to have PyCharm point at committed changes that are also integrated to other changelists and provide information on the target changelists that received the content in question.
Server timeout	In this text box, specify the time period in seconds after when the Perforce client cancels its attempts to establish connection to the server.
Enable Perforce Jobs Support	When this check box is selected, user interface for attaching and detaching Perforce jobs to change lists is provided in the Version Control tool window and in the Commit Changes dialog box.

Subversion

File | Settings | Version Control | Subversion for Windows and Linux

PyCharm | Preferences | Version Control | Subversion for macOS

Ctrl+Alt+S



Use this page to specify the version control settings to be applied to the directories of your project that are under Subversion control

In this section:

- [General tab](#)
- [Presentation tab](#)
- [Network tab](#)
- [SSH Settings](#)

General tab

Use this tab to configure the general Subversion integration settings.

Item Description

Use command line client	Select this option if you want to use the command line svn client. Enter the name of the executable file, or click the Browse button  and select the path in the dialog that opens.
Enable interactive mode	Select this option if you want PyCharm to emulate the behavior when Subversion commands are executed directly from the terminal in the interactive mode (dialogs will pop up where you can input credentials). This is required to handle password/passphrase prompts for svn+ssh repositories, and trust invalid server certificates for https repositories.
Use system default Subversion configuration directory	Select this check box to store Subversion configuration files in the system default location.
Subversion configuration directory	In this text box, specify the Subversion configuration directory if you do not want to use the default one. Enter the path manually, or click the Browse button  and select the path in the dialog that opens. This option is only available when the Use system default Subversion configuration directory option is deselected.
Update administrative information only in changed subtrees	This option only applies to working copies older than SVN 1.7 managed by SVNKit. During synchronization with the server (update), SVN locks your working copy one subtree after another by creating empty <code>lock</code> files in the corresponding administrative <code>.svn</code> directories. After that, SVN starts comparing file hashes to detect which local files need to be synchronized. When this option is selected, SVN first checks if any files from a subtree have been modified on the server, and locks this subtree (i.e. creates a <code>.svn/lock</code> file) only if such files are detected. This approach improves performance but may cause concurrency issues, for example, with antiviral software.
Clear Auth Cache	Click this button to delete all stored credentials for the <code>http</code> , <code>svn</code> and <code>svn+ssh</code> protocols from the authentication cache.

Presentation tab

Use this tab to configure the data presentation settings.

Item Description

Check svn:mergeinfo in target subtree when preparing for merge	Select this check box if you want PyCharm to check the merge tracking information for the target branch before merging to prevent duplicates.
Maximum number of revisions to look back in annotations	Select this check box to limit the number of revisions to look back at when calculating annotations, and specify the number of revisions.
Show merge source in history and annotations	Select this check box if you want merge sources to be visible in annotations and file history.
Ignore whitespace differences in annotations	Select this check box if you want white spaces to be ignored when annotating, and, thus, get more meaningful annotations and cast out senseless ones.

Network tab

Use this tab to configure the connection settings.

Item Description

Use PyCharm general proxy settings as default for Subversion	Select this check box if you want Subversion to use the default PyCharm proxy settings.
HTTP timeout	Specify the number of seconds to wait for HTTP connection to be established.
SSH connection timeout	Specify the number of seconds to wait for SSH connection to be established.
SSH read timeout	Specify the number of seconds to wait for response.
SSL protocols	In this area, select which SSL protocol you want to use. The available options are: <ul style="list-style-type: none">- All- SSLv3- TLSv1

SSH Settings

Use this tab to configure the settings used to connect to an SVN server via a tunneling SSH protocol.

ItemDescription

SSH executable	Specify the path to the SSH client. Enter the name of the executable file, or click the Browse button  and select the path in the dialog that opens. If not specified, 'ssh' is used by default. This field is only available if the Password or the Private key option is selected.
User name	Specify the user name for SSH connection. If the user name is explicitly specified in the repository URL, this value will be used and this setting will be ignored. This field is only available if the Password or the Private key option is selected.
Port	If your server is listening on a non-standard port (22 for svn+ssh://), modify the default value. This field is only available if the Password or the Private key option is selected.
Password	Select this option if you want to use a password for SSH authentication.
Private key	Select this option if you want to use a private key for SSH authentication.
Path	Specify the path to the private key. Enter the path manually, or click the Browse button  and select the path in the dialog that opens.
Subversion config	Select this option if you want to use the default settings stored in Subversion configuration for SSH connection.
SSH tunnel	This field displays the SSH tunnel settings stored in Subversion configuration. You can modify the value and click the Update button to write this value to the Subversion configuration.
Update	Click this button to check the Subversion configuration and update the value if necessary, or to write the value you have entered to the Subversion configuration.
SVN_SSH	This field displays the environment variable that can be used in the tunnel configuration (by default, <code>SVN_SSH</code>) and is stored in Subversion configuration.

The dialog box opens when you click the Edit Network Options button on the [Subversion](#) page of the [Settings/Preferences](#) dialog box. In this dialog box, specify the Subversion network settings stored in the servers Subversion runtime configuration file.

The dialog box contains two tabs:

- System file - this tab displays the default network configuration settings specified by the system administrator.
- User file - in this tab, customize the default network configuration settings.

The dialog box consists of two panes:

- On the left-hand pane, add, edit, and remove configuration profiles. Network configuration settings are arranged into profiles of two types:
 - Group - settings from such profile apply to a specific group, defined by a glob pattern.
 - Global - settings from this profile are applied to all servers that do not match any glob pattern.
- On the right-hand pane, specify the settings for the selected configuration profile.

Toolbar Options

Item	Tooltip and shortcut	Description
------	----------------------	-------------

	Add	Click this button to have a new configuration profile added to the list.
	Delete	Click this button to remove the selected profile from the list.
	Copy	Click this button to have a copy of the selected profile created.

HTTP Proxy Settings

Item	Description
------	-------------

URL Patterns	In this text box, type the patterns that define the URL addresses of repositories to be accessed via proxy. Use commas to separate patterns.
Exceptions	In this text box, type the patterns that define the URL addresses of repositories to be accessed directly, without using proxy. Use commas to separate patterns.
Server	In this text box, specify the name or IP address of the proxy server to use.
Port	In this text box, specify the port number the proxy server listens to.
Connection timeout	In this text box, specify the time period in seconds after when the Subversion client cancels its attempts to establish connection to the server.
User	In this text box, type the user name or login to authenticate at the specified proxy server.
Password	In this text box, type the password that matches the specified login or user name.

SSL Settings

Item	Description
------	-------------

Comma separated paths to CAs certificate files	In this text box, specify the paths to files that contain certificates of the Certificate Authority (CAs) files that are accepted by the Subversion client when accessing the repository.
SSL client certificate file	In this text box, specify the location of the SSL client certificate file. Type the path to the file manually or click the Browse button and choose the location in the dialog that opens .
SSL client certificate passphrase	In this text box, type the SSL client certificate passphrase to use.
Trust default CAs	Select this check box to have the Subversion integration trust the set of default Certificate Authority files shipped with OpenSSL .

Repositories

Item	Description
------	-------------

Repositories	This list displays the URL addresses of the previously accessed repositories.
Test connection	Click this button to make sure that connection to the selected repository can be established successfully according to the settings specified in the dialog box.

When you click this button, PyCharm displays the [Authentication Required](#) dialog box.

Current Project

This page contains settings of the current project.

- [Project Dependencies](#)
- [Project Interpreter](#)
- [Project Structure Dialog](#)

Project Dependencies

File | Settings | Project Dependencies for Windows and Linux

PyCharm | Preferences | Project Dependencies for macOS

Ctrl+Alt+S



Use this page to configure dependencies between projects [opened in the same window](#).

Note This page is not shown, if there is only one open project.

In this section:

- [Projects pane](#)
- [Project dependencies pane](#)

Projects pane

ItemDescription

Projects This pane displays the list of projects, opened in the same window. The first project in the list is the **primary project**.

Project dependencies pane

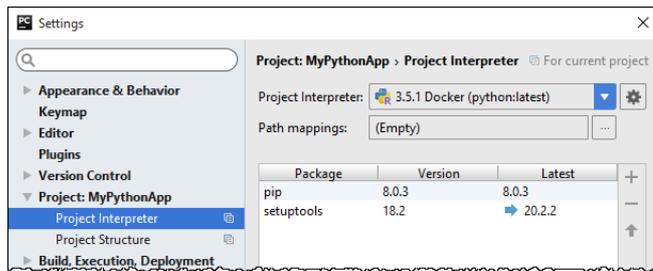
ItemDescription

Project depends on these projects For each project in the Projects pane, this pane shows the list of the other [projects, opened in the same window](#).

Select the check boxes to the left of the names of the desired projects to add them as dependencies.
Use the arrow buttons to change the order of the dependent projects, in which they appear in the PYTHONPATH.

package versions.

The buttons on this toolbar are disabled for the [Docker interpreters](#):



All the packages should be already installed in the Docker image. If some packages are missing, then you will have to create a new Docker image, as described on the page [Quickstart Guide: Compose and Django](#).

Tip One can have multiple [available interpreters already configured](#), but only the one selected becomes the [project interpreter](#). If one has, say, [two projects opened in the same window](#), then it is possible to have two different interpreters, selected from the list of available ones.

Ctrl+Alt+S



Project interpreters

This dialog box shows up on choosing More...:



Use this dialog to configure the list of available Python interpreters.

ItemTooltip and shortcut

ItemTooltip	Description	and shortcut
	Add	Click this button to select a Python interpreter type (local, remote, or virtual environment) from the pop-up list. Refer to the section Configuring Available Python Interpreters for details.
		
	Remove	Click this button to delete the selected Python interpreter from the list of available interpreters.
		
	Edit	Click this button to change name and location of the selected Python interpreter. Refer to the section Changing Name of a Python interpreter or Virtual Environment .
		
	Show virtual environments associated with other projects	If this button is pressed, the virtual environments, already associated with the other projects, are shown in the list of available interpreters. Otherwise, the list only includes the virtual environments not used in the other projects. This behavior is stipulated by the check box Associate this virtual environment with the current project , (refer to the section Creating Virtual Environment for details).
	Show paths for selected interpreter	Click this button to show the list of interpreter paths .

Interpreter paths

This dialog box shows up on clicking  in the [Project Interpreters](#) dialog box. It shows the paths where the external libraries reside. If a library has been added, it is recommended to click the Reload button to rescan the Python installation.

ItemTooltip and shortcut

	Add	Click these buttons to add external libraries for the selected Python interpreter.
		
	Remove	Click these buttons to remove external libraries for the selected Python interpreter.
		
	Reload List of Paths	Click this button to rescan libraries for the selected Python installation. For example, clicking this button helps update external libraries, if some packages have been upgraded.

Ctrl+Alt+S



This dialog is a package manager that enables the following functions:

- Viewing packages, available in the remote repositories
- Installing selected packages locally.
- Adding and removing repositories

ItemDescription

Search field Type here the search string. So doing, the list of packages shrinks to show the package names containing the entered string only.

- : Click this button to use the search history.
- : Click this button to clear the search field.

Package This area shows the list of packages residing in the accessible remote repositories. For each package you can view its name and repository. Additional information (description and versions) is displayed to the right.

Description For any selected package this field displays brief description, the latest version, author names, web site and email address.

Specify version If this check box is selected, the specified version of a package will be downloaded and installed. If this check box is not selected, then the latest version will be installed.

Options Select this check box to enter the option of the `pip install` command. One can explore the possible options by typing `pip install - help` in the command line.

Install package Click this button to install the selected package.

Manage repositories Click this button to open the Manage Repositories dialog box. By default, only the [pypi](#) repository is available. This repository cannot be deleted. It is possible to add more repositories, and delete the selected ones.

Project Structure Dialog

File | Settings | Project Structure for Windows/Linux

PyCharm | Preferences | Project Structure for macOS

Ctrl+Alt+S



In this section:

- [Projects pane](#)
- [Project Structure](#)
- [Configure content roots](#)
- [Add/remove content roots](#)

Projects pane

ItemDescription

Projects This pane displays the list of projects, opened in the same window. The first project in the list is the primary project.

Note This pane is not shown, if there is only one open project.

Project Structure

- [Configure content roots](#)
- [Add/remove content roots](#)

Use this pane to configure content roots for each of the [projects opened in the same window](#).

Configure content roots

ItemTooltipDescription

 Sources Mark a folder as a source root. Such folder displays in the list of source roots. If a folder is marked as a source root, it will be added to `PYTHONPATH`, and resolve will be performed against it. This command is duplicated on the context menu of a content root.

 Excluded Mark a folder as an excluded root. Excluded roots are not visible to PyCharm. Usually, one would like to exclude temporary build folders, generated output, logs, and other project output. Excluding the unnecessary paths is a good way to significantly improve performance. This command is duplicated on the context menu of a content root.

 Templates Mark a folder as a template root, where all the directories, containing templates in the [supported template languages](#), are stored. This command is duplicated on the context menu of a content root.

Note By default, the directory that has been defined as the templates folder on [project creation](#), is marked as . Note that if a directory is specified in the `TEMPLATES_DIR` of the `settings.py` file, then, on the first project opening, it is automatically marked as the template root.

 Resources Click this button to mark a folder as a resource root, for the static contents such as CSS or JavaScript files.

Add/remove content roots

ItemTooltip Description

	Add Content Root	Add a new root to the content roots. Click the button and navigate to the desired folder in the dialog that opens .
	Remove Content Entry	Delete selected content root from the project.
	Unmark	Remove marking of a content root and denote it with a regular directory icon  .

Build, Execution, Deployment

File | Settings | Build, Execution, Deployment

When you select the Build, Execution, Deployment category in the left-hand pane, its main subcategories are listed in the right-hand part of the dialog.

- [Debugger](#)
- [Python Debugger](#)
- [Deployment](#)
- [Buildout Support](#)
- [Console](#)
- [Coverage](#)
- [Docker](#)
- [Docker Registry](#)

Debugger

File | Settings | Build, Execution, Deployment | Debugger for Windows and Linux

PyCharm | Preferences | Build, Execution, Deployment | Debugger for macOS

Use this page to configure behavior of the Debugger and customize its view.

Common options

ItemDescription

Focus application on breakpoint	If this check box is selected, on hitting a breakpoint, PyCharm will show the location of this breakpoint in the editor and will attempt to bring its frame to the front.
Show debug window on breakpoint	If this check box is selected, PyCharm activates the Debug Tool Window on hitting a breakpoint.
Hide debug window on process termination	Automatically hide the Debug window when the debugged program terminates.
Scroll execution point to center	If this check box is selected, the line with the current execution point will be kept in the middle of the screen.

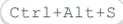
Built-in server

ItemDescription

Port	Use this spin box to specify the port on which the built-in web server runs. By default this port is set to the default PyCharm port <code>63342</code> through which PyCharm accepts connections from services. You can set the port number to any other value starting with 1024 and higher.
Can accept external connections	If this check box is selected, then the files on the built-in server running on the specified port are accessible from another computer. If this check box is cleared (by default), then the debugger listens only to local connections.
Allow unsigned requests	For security reasons, any request to a page on the built-in server from outside PyCharm is by default rejected and the following authorization pop-up window is displayed:



To access the requested page, click Copy authorization URL to clipboard and paste the generated token in the address bar of the browser. However this behaviour may be annoying, for example, it may block your debugging session if manual intervention is impossible. To suppress displaying the authorization pop-up window, select the Allow unsigned requests check box.

 Ctrl+Alt+S

Use this page to manage the way data is displayed in the debugger.

On this page:

- [Common debugger settings](#)
- [Editor](#)

Common debugger settings

Item Description

Sort values alphabetically	Select this option to sort the values in the Variables pane of the Debug tool window .
Enable auto expressions in Variables view	<p>Select this option if you want the PyCharm debugger to automatically evaluate expressions and show the corresponding values in the Variables pane of the Debug tool window.</p> <p>The debugger analyzes the context near the breakpoint (the current statement, one statement before, and one after). It does so to find various expressions in the source code (if available) such as, for example, <code>myvar.myfield</code>.</p> <p>If such expressions don't contain explicit method invocations, the debugger evaluates them and shows the corresponding values in the Variables view.</p>

Editor

ItemDescription

Show values inline	Select this option to enable the Inline Debugging feature that allows viewing the values of variables right next to their usage in the editor.
Show value tooltip	<p>Select this option to enable automatic display of tooltips for values.</p> <p>A tooltip in this context is a pop-up that provides an alternative, sometimes a more convenient presentation of values in the Variables pane of the Debug tool window.</p> <p>If this option is disabled, press  to display a value.</p>
Value tooltips delay (ms)	Specify the delay (in milliseconds) between the moment when the mouse pointer hovers over an object in the Variables pane of the Debug tool window , and the moment when a tooltip with the object's value is displayed.
Show value tooltip on code selection	Select this option to enable tooltips that show the expression value when you select a code fragment in the editor.

Ctrl+Alt+S



Use this page to improve the debug stepping speed and specify the elements to be skipped while stepping.

Item Description

JavaScript

Do not step into library scripts Select this check box to suppress stepping into library scripts while debugging.

Do not step into scripts Select this check box to suppress stepping into certain scripts while debugging. Use the toolbar buttons to manage the list of scripts to be skipped.

ItemShortcutDescription

		Click this button to add a new script filter.
		Click this button to delete the selected filter from the list.
		Click this button to edit the selected filter.
		Use these buttons to arrange filters as required.
		Click this button to create a copy of the selected filter.

Ctrl+Alt+S



The page is available only when the LiveEdit plugin is installed and enabled. The plugin is not bundled with PyCharm, but it can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).

Use this page to enable and disable instant preview of HTML, CSS, and JavaScript files in the actual browser and configure the appearance of the preview.

ItemDescription

Highlight current element in browser on caret change

When this check box is selected, the browser highlights the area that represents the code you are currently editing in PyCharm.

Update

In this area, configure the way changes made to the code during a debugging session are applied. Note that an update will now be performed only if none of the modified files have any syntax errors.

- Auto in (ms): Choose this option to have the changes applied automatically at certain time interval and specify the delay in ms in the text box. This policy is not available for the client-side code of Meteor applications.

- Manual: Choose this option to apply the changes manually by clicking the Update <run configuration name> JavaScript button  or the Update <run configuration name> button  on the toolbar of the Debug tool window.

It is recommended that you choose this option for debugging Meteor applications because applying changes to the client-side code is not supported.

- Restart if hotswap fails:

With changes in HTML, CSS, and JavaScript on the client side, the contents of a Web page in the browser are updated without reloading. For NodeJS or Meteor applications, PyCharm first tries to update the application incorporating the changes without restarting the NodeJS server.

- Select this check box to have PyCharm try restarting the server if the changes cannot be applied automatically. After the restart, PyCharm brings you to the execution point where the problem took place.

If even with this option chosen automatic upload still fails, you will have to restart the server manually by clicking the Rerun <run configuration name> button .

- When the check box is cleared, PyCharm just displays a pop-up window informing you about the failure and suggesting you to restart the server manually.

Python Debugger

Warning! This page only appears when Python Plugin is installed and enabled!

File | Settings | Build, Execution, Deployment | Python Debugger for Windows and Linux

PyCharm | Preferences | Build, Execution, Deployment | Python Debugger for macOS

Ctrl+Alt+S



Use this page to configure Python debug options.

ItemDescription

Attach to subprocess automatically while debugging	If this check box is selected, PyCharm will automatically attach all subprocesses of the process being debugged. Thus, if the parent process has subprocesses, their breakpoints will always work.
Collect run-time types information for code insight	If this check box is selected, the types of function calls are preserved during debugging, and passed to the type checker. Refer to the section Using Docstrings to Specify Types for details.
Clear caches	Click this button to remove information about the types of arguments, collected at run time.
Gevent compatible	If this check box is selected, the debugger will be compatible with the Gevent-monkeypatched code. Note This parameter works for Python >= 2.6, Python >= 3.3
PyQt compatible	If PyQt is installed on the interpreter, but is not imported in the application code, some import errors may occur. Unchecking this option fixes these errors.

Deployment

File | Settings | Build, Execution, Deployment | Deployment for Windows and Linux

PyCharm | Preferences | Build, Execution, Deployment | Deployment for macOS

Ctrl+Alt+S



In this section:

- Deployment
 - [Basics](#)
 - [Toolbar and common options](#)
- [Connection Tab](#)
- [Mappings Tab](#)
- [Excluded Paths Tab](#)
- [Add Server Dialog](#)
- [Options](#)
- [Advanced Options Dialog](#)
- [Files/Folders Default Permissions Dialog](#)

Basics

On this page, create, edit, and delete **server access configurations** that give you control over interaction between PyCharm and servers. Anytime you are going to use a server, you need to define a **server access configurations**, no matter whether your server is on a remote host or on your computer.

Among numerous ways to configure your development and production environments the most frequent ones are as follows:

- The Web server is installed on your computer. The sources are under the server document root (for example, `/htdocs`), and you do your development right on the server.
- The Web server is installed on your computer but the sources are stored in another folder. You do your development, then copy the sources to the server.
- The Web server is on another computer (remote host). Files on the server are available through the FTP/SFTP/FTPS protocol, through a network share, or a mounted drive.

Note that PyCharm assumes that all development, debugging, and testing is done on your computer and then the code is deployed to a production environment. For detailed reasoning of this approach, see [Deployment. Working with Web Servers](#)

Let's define the terms and their meaning in the context of synchronization between PyCharm and servers.

- An **in-place server** is a server whose **document root** is the parent of the project root, either immediate or not. In other words, the Web server is running on your computer, your project is under its document root, and you do your development directly on the server.
 - A **local server** is a server that is running in a local or a mounted folder and whose **document root** is **NOT** the parent of the project root.
 - A **remote server** is a server on another computer (remote host).
 - The **server configuration root** is the highest folder in the file tree on the **local** or **remote** server accessible through the server configuration. For **in-place** servers, it is the project root.
 - A **local file/folder** is any file or folder under the project root.
 - A **remote file/folder** is any file or folder on the server, either local or remote.
- Suppose you have a project `C:/Projects/My_Project/` with a folder `C:/Projects/My_Project/My_Folder` and a local server with the document root in `C:/xampp/htdocs`. You upload the entire project tree to `C:/xampp/htdocs/My_Project`. In the terms of PyCharm, the folder `C:/Projects/My_Project/My_Folder` is referred to as **local** and the folder `C:/xampp/htdocs/My_Project/My_Folder` is referred to as **remote**.
- **Upload** is copying data from the project **TO** the server, either local or remote.
 - **Download** is copying data **FROM** the server to the project.

Synchronization with servers, uploading, downloading, and managing files on them are provided via the Remote Hosts Access bundled plugin, which is by default enabled. If the plugin is disabled, activate it in the Plugins page of the Settings dialog box. For details, see [Enabling and Disabling Plugins](#).

Toolbar and common options

Use the toolbar buttons to manage the list of configurations.

The left-hand pane shows a list of all the server access configurations available in PyCharm. When you select a configuration, the right-hand pane shows the configuration details.

Item	Tooltip	Description
------	---------	-------------

	Add	Click this button to open the Add Server dialog box and define a new configuration there.
	Insert	
	Delete	Click this button to remove the selected configuration from the list.
	Delete	



Copy

Click this button to copy the settings of the selected configuration.

Ctrl+D



Use as
Default

Click this button to have PyCharm apply the settings of the selected configuration by default during automatic upload of changed files.

Connection Tab

Use this tab to choose the [way to access the Web server](#) and specify the [connection settings](#).

ItemDescription

Name	The text box shows the configuration name specified in the Add Server dialog box. Edit the configuration name, if necessary.
Visible only for this project	Use this check box to configure the visibility of the server access configuration (deployment configuration). <ul style="list-style-type: none">– Select the check box to restrict the use of the configuration to the current project. Such configurations cannot be reused outside the current project, they do not appear in the list of available configurations in other projects. For example, if this check box is selected in an SFTP configuration, you cannot use your SSH credentials from it when you configure a remote interpreter.– When the check box is cleared, the configuration is visible in all PyCharm projects and the settings from, including SSH credentials, can be reused. <p>See Configuring Node.js Interpreters and Configuring Remote Python Interpreters for details.</p>
Access type	From this drop-down list, choose the way to access the server. Use the Up and Down keyboard keys to scroll through the list of server configuration types. The available options are: <ul style="list-style-type: none">– FTP: choose this option to have PyCharm access the server via the FTP file transfer protocol.– SFTP: choose this option to have PyCharm access the server via the SFTP file transfer protocol.– FTPS: choose this option to have PyCharm access the server via the FTP file transfer protocol over SSL (the FTPS extension).– Local or mounted folder: choose this option if the Web server is running in a local or a mounted folder and its document root is NOT the parent of the project root.– In-place: choose this option if the Web server is running on your computer, your project is under its document root, and you do your development directly on the server.

Upload/Download Project Files

In this area, specify the settings for accessing the server to upload and download files to and from.

The set of controls in the area depends on the chosen server access type.

ItemDescriptionAvailable for

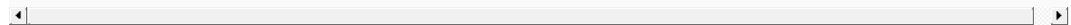
Folder	In this field, specify the server configuration root . The server configuration root is the highest folder in the file tree on the server that can be accessed through the server configuration. The easiest way is to use the document root of your Web server as defined in the Web server configuration file. However you can appoint any other existing folder under the document root .	Local or mounted folder
FTP/FTPS/SFTP host	In this text box, specify the host name of the FTP/SFTP server to upload the files to.	FTP, FTPS, SFTP
Port	In this text box, specify the port to use. The default values are: <ul style="list-style-type: none">– 21 for FTP and FTPS– 22 for SFTP	FTP, FTPS, SFTP
Root Path	In this text box, specify the server configuration root relative to your user home which was defined when you registered your account. This folder will be the highest one in the folder structure accessible through the current server configuration. Do one of the following: <ul style="list-style-type: none">– Accept the default value <code>/</code>, which points at the user home folder on the server.– Type the path manually.– Click the Browse button  and select the desired folder in the Choose Root Path dialog box that opens.– Click the Autodetect button and have PyCharm detect the user home folder settings on the FTP/SFTP server and set up the root path according to them. The button is only enabled when you have specified your user name and password.	FTP, FTPS, SFTP
Autodetect	Click this button to have PyCharm detect the user home folder settings on the FTP/SFTP server and set up the root path according to them.	FTP, FTPS, SFTP
User name	In this text box, type your user name for authentication to the server. <div style="background-color: yellow; padding: 2px;">Note The button is only enabled when you have specified your user credentials.</div>	FTP, FTPS, SFTP
Log in as anonymous	Select this check box to enable anonymous access to the server with your email address as password.	FTP, FTPS, SFTP
Auth type	From this drop-down list, select the client authentication method. The available options are: <ul style="list-style-type: none">– Password - select this option to use standard authentication through a password.– Key pair (OpenSSH) - select this option to use SSH authentication via a key pair. To apply this authentication method, you need to have your private key on the client machine and your public key on the remote server you connect to. PyCharm supports private keys generated using the OpenSSH utility. See http://wiki.qnap.com/wiki/How_To_Set_Up_Authorized_Keys for details.	SFTP
Password	In this text box, type your password for authentication to the server.	FTP, FTPS, SFTP
Private key file	In this text box, specify the location of your private key file.	SFTP
Passphrase	In this text box, specify your authentication passphrase.	SFTP
Save password	Select this check box to have PyCharm remember the specified password.	FTP, FTPS, SFTP

Save passphrase	Select this check box to have PyCharm remember the specified passphrase.	SFTP
Test FTP/FTPS/SFTP connection	Click this button to check that the specified settings ensure successful connection via FTP/SFTP.	FTP, FTPS, SFTP
Explicit	Choose this option to have the explicit (active) security applied. Immediately after establishing connection, the FTP client on your machine sends a command to the server to establish secure control connection through the default FTP port.	FTPS
Implicit	Choose this option to have the implicit (passive) security applied. In this case, security is provided automatically upon establishing connection to the server which appoints a separate port for secure connections.	FTPS
Advanced options	Click this button to specify additional uploading settings in the Advanced Options dialog box that opens.	FTP, FTPS, SFTP
Web server root URL	In this text box, specify the URL address of the Web server root folder . Both the HTTP and the HTTPS protocols are supported. To access a server through HTTPS, you need to acquire a certificate file <code><certificate_name>.cert</code> signed by a recognized authority and import this certificate in the truststore/keystore of the Oracle JRE (Java Runtime Environment) on which PyCharm runs. Note that self-signed certificates are rejected as unsafe.	All

To import a certificate in Oracle JRE:

1. Open the embedded Terminal and type the following command:

```
<jre_home>/bin/keytool.exe -importcert -keystore <path to jre truststore/keystore> -file <full_path_to_<cert_name>.cert>
```



If you are using the Oracle JRE bundled with PyCharm, the default path to the truststore/keystore is

```
<%product_installation_folder>/jre/jre/lib/security/jssecacerts or
```

```
<%product_installation_folder>/jre/jre/lib/security/cacerts .
```

Otherwise it is `<jre_home>/jre/lib/security/jssecacerts` or `<jre_home>/jre/lib/security/cacerts` .

2. When asked to enter a password for the truststore/keystore, specify the default one `changeit` .
3. Open the `PyCharm.exe.vmoptions` file in the `<PyCharm_installation_folder>/bin` and add the following line to it:

```
-Djavax.net.ssl.keyStore=<path to keystore>
```

4. Restart PyCharm.

Learn more at [Java6](#) and [Java7](#).

Open	Click this button to make sure that the specified server root URL address is accessible and points at the correct Web page.	All
------	---	-----

Mappings Tab

In this tab, configure **mappings**, that is, set correspondence between the project folders, the folders on the server to copy project files to, and the URL addresses to access the copied data on the server. The easiest way is to map the entire project root folder to a folder on the server, whereupon the project folder structure will be repeated on the server, provided that you have selected the Create Empty directories check box in the [Options dialog box](#). For more details, see [Customizing Upload/Download](#). Below are definitions of terms used in this topic in the context of synchronization between PyCharm and servers.

- A **local server** is a server that is running in a local or a mounted folder and whose **document root** is **NOT** the parent of the project root.
- A **remote server** is a server on another computer (remote host).
- The **server configuration root** is the highest folder in the file tree on the **local** or **remote** server accessible through the server configuration. For **in-place** servers, it is the project root.
- A **local file/folder** is any file or folder under the project root.
- A **remote file/folder** is any file or folder on the server, either local or remote.

Suppose you have a project `C:/Projects/My_Project/` with a folder `C:/Projects/My_Project/My_Folder` and a local server with the document root in `C:/xampp/htdocs`. You upload the entire project tree to `C:/xampp/htdocs/My_Project`. In the terms of PyCharm, the folder `C:/Projects/My_Project/My_Folder` is referred to as **local** and the folder `C:/xampp/htdocs/My_Project/My_Folder` is referred to as **remote**.

ItemDescription

Use this server as default	Click this button to have PyCharm apply the settings of the selected configuration by default during automatic upload of changed files. This button is only enabled for the non-default servers; for the server used as default , this button is disabled.
Local Path	In this text box, specify the full path to the desired folder in the project tree. In the simplest case it is the project root. Type the path manually or click the Browse button  and select the desired location in the Choose Local Path dialog box that opens.
Deployment Path	In this text box, specify the folder on the server where PyCharm will upload the data from the folder specified in the Local Path text box. Type the path to the folder relative to the server configuration root . If the folder with the specified name does not exist yet, PyCharm will create it, provided that you have selected the Create Empty directories check box in the Options dialog box . For more details, see Customizing Upload/Download . The text box is not available for server configurations of the type In-place.
Web Path	In this text box, type the path to the folder on the server relative to the server configuration root . Actually, type the relative path you typed in the Deployment Path text box.
Add	Click this button to have a new line added to the list of mappings.
Remove	Click this button to remove the selected mapping from the list.

Excluded Paths Tab

Use this tab to configure a list of local and remote folders that you do not want to be involved in upload/download.

ItemDescription

Add local path	Click this button to have an empty line added to the list and specify the location of the folder to be protected against upload/download. Type the path manually or click the Browse button  and choose the required folder in the dialog that opens .
Add deployment path	Click this button to have an empty line added to the list. Click the Browse button  . The Select remote excluded path dialog box that opens shows the data on the host accessed through the selected server configuration. Select the required folder.
Remove path	Click this button to remove the selected item from the list. The button is only available when a line is selected.

Add Server Dialog

The dialog box opens when you click the Add toolbar button  on the [Deployment](#) page.

Use the dialog box to create **server access configurations** that give you control over interaction between PyCharm and servers. Anytime you are going to use a server, you need to define a **server access configurations**, no matter whether your server is on a remote host or on your computer.

ItemDescription

Name	In this text box, type the name of the new Web server configuration.
Type	<p>From this drop-down list, choose the way to access the server. Use the Up and Down keyboard keys to scroll through the list of server configuration types. The available options are:</p> <ul style="list-style-type: none">– FTP: choose this option to have PyCharm access the server via the FTP file transfer protocol.– SFTP: choose this option to have PyCharm access the server via the SFTP file transfer protocol.– FTPS: choose this option to have PyCharm access the server via the FTP file transfer protocol over SSL (the FTPS extension).– Local or mounted folder: choose this option if the Web server is running in a local or a mounted folder and its document root is NOT the parent of the project root.– In-place: choose this option if the Web server is running on your computer, your project is under its document root, and you do your development directly on the server.

When editing the server configuration name in the Name text box, use the Up and Down keys on your keyboard to change the preselected server access to type in the Type drop-down list.

Ctrl+Alt+S

Use this page to specify additional configuration settings for uploading and downloading project files to and from local and remote servers. For more details about various server access configurations, see [Deployment. Working with Web Servers](#).

The options specified in this dialog box apply to all defined server configurations regardless of the server type (local, remote) and the data transfer protocol used. Protocol-specific options for server configurations of the type FTP/SFTP/FTPS are defined in the [Advanced Options Dialog](#).

ItemDescription

Exclude items by name	In this text box, specify patterns for the names of files and folders that you do not need to be deployed. Use semicolons as delimiters. Wildcards are welcome. The exclusion is applied recursively. This means that if a matching folder has subfolders, the contents of these subfolders are not deployed either.
Operations logging	Use this drop-down list to specify how much detailed logging you need to have. The available options are: <ul style="list-style-type: none"> – Errors only: select this option to have the log show only errors occurred during upload. – Brief: select this option to have all events reflected in the log but without details. – Detailed: select this option to have more details on the upload shown in the log, for example, full file paths.
Stop operation on the first error	Select this check box to have data transfer stopped as soon as an error occurs.
Overwrite up-to-date files	Do one of the following: <ul style="list-style-type: none"> – Select this check box to have all the files uploaded no matter whether they have been changed since the previous upload or not. – Clear the check box to upload only files that have been changed since the previous upload.
Preserve files timestamps	Select this check box to prevent resetting timestamps of files on upload.
Delete target items when source ones do not exist	If this check box is selected, any file in the destination directory will be removed if the file with this name is not involved in the current upload. This option is applicable when synchronization is performed from the Project tool window or from the Remote Host tool window.
Create empty directories	Select this check box to have an empty directory on the server created automatically if a new local directory has been created in your project since the last upload in the source folder.
Prompt when overwriting or deleting local items	Select this check box to have PyCharm ask you for confirmation before overwriting or deleting local items for synchronization during download.
Upload changed files automatically to the default server	From this drop-down list, choose when you want PyCharm to automatically upload a file to the default server. The available options are: <ul style="list-style-type: none"> – Always: choose this option to have a file uploaded upon each save, no matter automatic or explicitly invoked. – On explicit save action: choose this option to have a file uploaded after save only if this save was invoked manually by choosing File Save all or pressing Ctrl+S. – Never: choose this option to suppress automatic upload. <p>The default server configuration is appointed on the Deployment page by selecting the desired configuration in the list and clicking the Use as Default toolbar button .</p>
Upload external changes	Select this check box to have PyCharm upload also the local changes that were made using a third-party tool.
Override default permissions on files	Select this check box to change the default permissions assigned to uploaded files on remote hosts. Click the Browse button  to open the Files Default Permissions dialog box, where you can manage access to uploaded files on remote hosts by assigning permissions.
Override default permissions on folders	Select this check box to change the default permissions assigned to uploaded folders on remote hosts. Click the Browse button  to open the Folders Default Permissions dialog box, where you can manage access to uploaded folders on remote hosts by assigning permissions.
Warn when uploading over newer file	Use this drop-down list to define the version-control policy to apply when uploading files to remote hosts. Depending on this choice, PyCharm either checks whether any changes have been made to the corresponding files on the remote host since you downloaded them or just overwrites the remote files. <ul style="list-style-type: none"> – No: choose this option to have the file on the remote host overwritten with its local copy silently. All the changes made to the remote file since your last synchronization will be abandoned. – Compare timestamp and size: if you choose this option, PyCharm performs two checks: <ol style="list-style-type: none"> 1. Compares the sizes of the local and remote files. 2. Compares the remote file timestamp set at the moment of the last synchronization with the current remote file timestamp. <p>If the files differ in their size or the remote file timestamps differ, PyCharm opens a Difference Viewer for Files, where you can explore and integrate the differences.</p> <p>This type of check depends on the timezone setting. If the timezone setting on your local machine is different from that on the remote host, the check may be successful even though the file versions actually differ.</p> <ul style="list-style-type: none"> – Compare content: when this option is chosen, PyCharm compares the content of the local and remote files. If any diversions are detected, PyCharm opens a Difference Viewer for Files, where you can explore and integrate the differences.

Notify about remote changes Select this check box to receive notifications about changes on the remote host. The check box is available only when the Compare timestamp and size: or Compare content: option is selected in the Warn when uploading over newer file drop-down list.

SFTP advanced options (IDE level)

Add new host key to known_hosts Choose whether PyCharm should ask about connecting to a host not mentioned in the file `known_hosts`. The following options are available:

- **Always:** always connect and add its record to the file `known_hosts`.
- **Ask:** ask whether PyCharm should connect to a host and add its record to the file `known_hosts`.
- **Never:** do not connect.

Hash hosts in known_hosts file If this check box is selected, the new host record will be stored in hash format.

Advanced Options Dialog

The dialog box opens when you click the Advanced Options button in the [Connection](#) tab of the [Deployment](#) page. Use this dialog box to customize upload/download by specifying additional protocol-specific options for server configurations of the type FTP/SFTP/FTPS.

Item	Description	Available for
Passive mode	Select this check box to set the client on your machine to the passive mode , when it connects to the server to inform about being in the passive mode, receives the port number to listen to, and established data connection through the port with the received number. This mode is helpful when your machine is behind a firewall.	FTP, FTPS
Show and process hidden files	When this check box is selected: <ol style="list-style-type: none"> 1. Hidden files and directories are shown in the Remote Host Tool Window. 2. Hidden files and directories are involved in diff and synchronization operations. <p>The name of a hidden file or directory starts with a dot (<code>.</code>).</p>	FTP, FTPS
Compatibility mode	Select this check box to ensure compatibility in child file naming with your FTP server. This option is helpful if the remote FTP server reports the following error: <div style="border: 1px solid #ccc; padding: 5px; margin: 5px 0;"> Invalid descendant file name <file name> </div> <p>Selecting this option may slow down synchronization with the server.</p>	FTP
Retrieve accurate files timestamps	Use this drop-down list to specify the MDTM FTP command calling policy to retrieve the last-modified time of a given file on the remote host. The available options are: <ul style="list-style-type: none"> – Always - select this option to have MDTM called for every file shown in the Remote Host tool window. – On copy - select this option to have MDTM called in the following cases: <ul style="list-style-type: none"> – To check whether a file is up to date when the Overwrite up-to-date files check box in the Options dialog box is cleared. – To preserve the actual time stamp of a file during download. – Never - select this option to suppress calling MDTM. 	FTP, FTPS, SFTP
Limit concurrent connections	Select this check box to have PyCharm restrict the number of connections to be supported simultaneously and specify the maximum number of allowed connections in the text box.	FTP, FTPS, SFTP
Control encoding	In this text box, specify the encoding that matches the encoding used by your server. Accept the default value if you are not sure that it supports UTF-8 encoding.	FTP, FTPS, SFTP
Always use LIST command	Select this check box to use the standard <code>LIST</code> command for listing instead of the <code>MLSD</code> command. This lets you avoid problems, for example, failure during upload with the <code>Invalid descendent file name</code> exception if the FTP server supports <code>MLSD</code> and returns <code>cdin</code> .	FTP, FTPS
Send keep alive message each	In this text box, specify how often you want PyCharm to send commands to the server to reset the timeout and thus preserve the connection.	FTP, FTPS, SFTP
Use keep alive command	From this drop-down list, choose the commands to be sent to the server to reset the timeout and thus preserve the connection.	FTP, FTPS
Ignore info messages	On some SFTP servers, the SSH banner may be enabled. Every time a connection is established, a pop-up window with an information message may be shown and to continue you would need to click OK. To suppress showing the information pop-up window, select the ignore info messages check box.	SFTP

Files/Folders Default Permissions Dialog

This dialog opens when you select the [Override default permissions on files](#) or [Override default permissions on folders](#) check box in the [Options](#) dialog and click the [Browse](#) button  next to it.

Use this dialog box to re-assign default server permissions to owners of files or folders, groups of owners, and other users.

The following identifiers are used for permission types:

- R stands for Read.
- W stands for Write.
- X stands for Execute

ItemDescription

Owner	In this row, specify what an owner of a file or folder may do by selecting the check boxes below the corresponding identifiers.
Group	In this row, specify what a group of owners of a file or folder may do by selecting the check boxes below the corresponding identifiers.
Others	In this row, specify what any one else may do by selecting the check boxes below the corresponding identifiers.
Octal	In this text box, specify the octal representation of the specified set of permissions. By default, PyCharm calculates and re-calculates the value of the field as you select or clear the desired check boxes. You can also specify the octal value manually.

Buildout Support

File | Settings | Buildout for Windows and Linux

PyCharm | Preferences | Buildout for macOS

Ctrl+Alt+S



Use this page to configure [Buildout support](#) for each of the [projects opened in the same window](#).

In this section:

- [Projects pane](#)
- [Prerequisites](#)
- [Buildout support pane](#)

Projects pane

ItemDescription

Projects This pane displays the list of projects, opened in the same window. The first project in the list is the **primary project**.

Note This pane is not shown, if there is only one open project.

Prerequisites

PyCharm recognizes `buildout` support in existing projects. However, you have to perform the following general steps as prerequisites (outside of PyCharm):

1. Add `buildout.cfg` and `setup.py` files to the root directory of your project.
2. Check out `bootstrap.py` and place it to the root directory of your project.
3. Execute `bootstrap.py` to create the `buildout` directory structure, download `buildout.exe`, and the other dependencies.
4. Execute `bin\buildout.exe` script. This will download the dependencies and generate a runner script for each of the parts listed in `buildout.cfg`.

Buildout support pane

ItemDescription

Enable `buildout` support This check box is selected automatically, when PyCharm detects `buildout` in your project. In this case, all the references in the source code are resolved according to the settings specified in `buildout.cfg` file.

Use paths from script Select the script that contains the paths to be used for resolving references in the source code. By default, PyCharm suggests `buildout-script.py` (for Windows) or `buildout.sh` (for the other operating systems). However, you can specify any of the runner scripts, generated by `bin\buildout.exe`, depending on the specific tasks of your project.

Console

File | Settings | Build, Execution, Deployment | Console for Windows and Linux

PyCharm | Preferences | Build, Execution, Deployment | Console for macOS

Ctrl+Alt+S



Use this page to define console options for the Python console.

In this section:

- Console
 - [Console common options](#)
- [Django Console](#)
- [Python Console](#)

Console common options

Item	Description
Always show debug console	If this check box is selected, the debug console will be shown by default in the Debug view.
Use IPython if available	When the check box is selected (by default): If IPython is installed, then IPython console will be launched.
	If the check box is not selected, then, even with the installed IPython, a Python console will be launched.

Warning! This page only appears when Django support is enabled!

File | Settings | Build, Execution, Deployment | Console | Django Console for Windows and Linux

PyCharm | Preferences | Build, Execution, Deployment | Console | Django Console for macOS

Ctrl+Alt+S

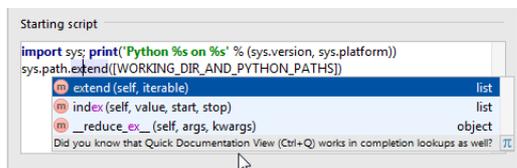


Use this page to define the Python interpreter, its options, starting script etc. for the Django console.

ItemDescription

Environment

Project	Click this drop-down list to select one of the projects, opened in the same PyCharm window , where this run/debug configuration should be used. If there is only one open project, this field is not displayed.
Environment variable	<p>This field shows the list of environment variables. If the list contains several variables, they are delimited with semicolons.</p> <p>To fill in the list, click the browse button, or press <code>Shift+Enter</code> and specify the desired set of environment variables in the Environment Variables dialog box.</p> <p>To create a new variable, click <code>+</code>, and type the desired name and value.</p>
Python Interpreter	Select one of the pre-configured Python interpreters from the drop-down list.
Interpreter options	In this field, specify the string to be passed to the interpreter. If necessary, click  and type the string in the editor.
Working directory	Specify a directory to be used by the running console. When this field is left blank, the project directory will be used.
Configure interpreters	If the desired interpreter is missing in the drop-down list, click this link to open the Project Interpreters page, and configure an interpreter or virtual environment, as described in the section Configuring Python SDK .
Add content roots to PYTHONPATH	Select this check box to have the content roots added to the PYTHONPATH.
Add source roots to PYTHONPATH	Select this check box to have the source roots added to the PYTHONPATH.
Starting script	In this editor area, type the script to be executed in the console after its start-up and initialization. Note that syntax highlighting, code completion, import assistance, documentation, inspections and quick fixes are available in this editor:



By default, this area contains the following script, which causes printing out a header information and extending the system paths:

```
import sys; print('Python %s on %s' % (sys.version, sys.platform))
sys.path.extend([WORKING_DIR_AND_PYTHON_PATHS])
```

If you want to omit such a printout, delete this script.

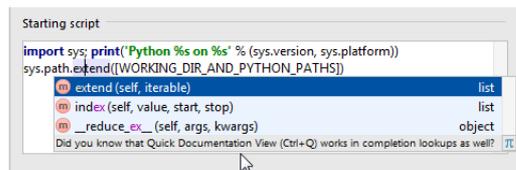
Ctrl+Alt+S



Use this page to define the Python interpreter, its options, starting script etc. for the Python console.

ItemDescription**Environment**

Project	Click this drop-down list to select one of the projects, opened in the same PyCharm window , where this run/debug configuration should be used. If there is only one open project, this field is not displayed.
Environment variable	<p>This field shows the list of environment variables. If the list contains several variables, they are delimited with semicolons.</p> <p>To fill in the list, click the browse button, or press <code>Shift+Enter</code> and specify the desired set of environment variables in the Environment Variables dialog box.</p> <p>To create a new variable, click <code>+</code>, and type the desired name and value.</p>
Python Interpreter	Select one of the pre-configured Python interpreters from the drop-down list.
Interpreter options	In this field, specify the string to be passed to the interpreter. If necessary, click  and type the string in the editor.
Working directory	Specify a directory to be used by the running console. When this field is left blank, the project directory will be used.
Configure interpreters	If the desired interpreter is missing in the drop-down list, click this link to open the Project Interpreters page, and configure an interpreter or virtual environment, as described in the section Configuring Python SDK .
Add content roots to PYTHONPATH	Select this check box to have the content roots added to the PYTHONPATH.
Add source roots to PYTHONPATH	Select this check box to have the source roots added to the PYTHONPATH.
Starting script	In this editor area, type the script to be executed in the console after its start-up and initialization. Note that syntax highlighting, code completion, import assistance, documentation, inspections and quick fixes are available in this editor:



By default, this area contains the following script, which causes printing out a header information and extending the system paths:

```
import sys; print('Python %s on %s' % (sys.version, sys.platform))
sys.path.extend([WORKING_DIR_AND_PYTHON_PATHS])
```

If you want to omit such a printout, delete this script.

Coverage

File | Settings | Build, Execution, Deployment | Coverage for Windows and Linux

PyCharm | Preferences | Build, Execution, Deployment | Coverage for macOS

Ctrl+Alt+S



Use this page to define the way the coverage data will be processed.

ItemDescription

When new coverage is gathered

Show options before applying coverage to the editor Choose this option to show the Code Coverage dialog every time you launch a new run configuration with code coverage. The coverage options dialog is displayed, when different coverage data have been produced.

Do not apply collected coverage Choose this option to cancel applying the new code coverage results.

Replace active suites with the new one Choose this option to have the active suites replaced with the new one every time you launch a new run configuration with code coverage.

Add to active suites Choose this option to have the new code coverage suite added to the active suites every time you launch a new run configuration with code coverage.

Activate coverage view Select this check box to have the [Coverage](#) tool window opened automatically when an application or test is run with coverage.

Python coverage

Use bundled coverage.py If this check box is selected, PyCharm will use the bundled `coverage.py`.

If this check box is not selected, PyCharm will use the coverage tool included in the selected Python interpreter. Refer to the section [Code Coverage](#) for details.

Branch coverage

This check box enables branch coverage in `coverage.py` tool. Thus additional information to the pure line coverage reports is added, marking the coverage of lines with conditional statements as incomplete in case one or more branches haven't been executed:

```
10
11 def demo(b, a, c):
12     d = b ** 2 - 4 * a * c
13     if d >= 0:
14         root1 = (-b + math.sqrt(d)) / (2 * a)
15         root2 = (-b - math.sqrt(d)) / (2 * a)
16         print("Two roots: ", root1, root2)
17     else:
18         print("No roots")
19     raise Exception
20
```

Refer to [this page](#) for details.

Docker

File | Settings | Build, Execution, Deployment | Docker for Windows and Linux

PyCharm | Preferences | Build, Execution, Deployment | Docker for macOS

For this page to be available, the Docker integration plugin must be installed and enabled.

Docker

Specify the settings for accessing the Docker API and Docker Compose.

ItemDescription

Name	The name of the configuration.
API URL	<p>The Docker API URL. Depending on your Docker version and operating system:</p> <ul style="list-style-type: none">– Docker for Windows: <code>tcp://localhost:2375</code>– Docker for macOS or Linux: <code>unix:///var/run/docker.sock</code>– Docker Toolbox for Windows or macOS (deprecated): <code>https://192.168.99.100:2376</code>
Certificates folder	<p>The path to the certificates folder - if you are using the Docker Toolbox. Depending on your Docker version and operating system:</p> <ul style="list-style-type: none">– Docker for Windows, macOS or Linux: This field must be empty.– Docker Toolbox for Windows (deprecated): <code><your_home_directory>\.docker\machine\machines\default</code>– Docker Toolbox for macOS (deprecated): usually, <code><your_home_directory>/\.docker/</code> or its subdirectory.
Docker Compose executable	<p><code>docker-compose</code> or an actual path to <code>docker-compose.exe</code> or <code>docker-compose.sh</code> (normally located in the <code>bin</code> folder of the Docker installation directory).</p> <p>The default setting <code>docker-compose</code> for Docker Compose executable is fine if:</p> <ul style="list-style-type: none">– The actual name of the executable file is <code>docker-compose</code>.– The path to the directory where the file is located is included in the environment variable <code>Path</code>. <p>See Docker Compose.</p>
Import credentials from Docker Machine	<p>If you want to import credentials for accessing the Docker Remote API from Docker Machine, select the check box and specify the associated settings:</p> <ul style="list-style-type: none">– Docker Machine executable. <code>docker-machine</code> or an actual path to <code>docker-machine.exe</code> (normally located in the Docker Toolbox installation folder). <p>The default setting <code>docker-machine</code> is fine if:</p> <ul style="list-style-type: none">– The actual name of the executable file is <code>docker-machine</code>.– The path to the directory where the file is located is included in the environment variable <code>Path</code>. <p>Detect. When you click Detect, PyCharm sets Docker Machine executable to <code>docker-machine</code>, tries to detect the executable file and, if a success, shows its version.</p> <ul style="list-style-type: none">– Machine. After specifying the Docker Machine executable or clicking Detect, the list contains the Docker Machines available locally, and you can select the one to be used.

See also, [Docker](#).

Toolbar

Use **+** to create configurations and **-** to delete them.

Docker Registry

File | Settings | Build, Execution, Deployment | Docker Registry for Windows and Linux

PyCharm | Preferences | Build, Execution, Deployment | Docker Registry for macOS

Warning! The following is only valid when Docker Integration Plugin is installed and enabled!

This page lets you manage your Docker Registry configurations which represent your Docker image repository user accounts.

Toolbar

ItemDescription

	Create a new configuration.
	Delete the selected configurations.

Configuration settings

ItemDescription

Name	The name of the configuration
Address	The image repository service URL, e.g. – registry.hub.docker.com for Docker Hub – quay.io for Quay
Username	The user name for your user account.
Password	Your password.
Email	The email address that you specified when creating your user account.
Server	The associated Docker configuration (used to connect to the service to check that your user account settings are correct).

Languages and Frameworks

File | Settings | Languages and Frameworks for Windows and Linux

PyCharm | Preferences | Languages and Frameworks for macOS

When you select the Languages and Frameworks category in the left-hand pane, its main subcategories are listed in the right-hand part of the dialog.

- JavaScript
- Schemas and DTDs
- BDD
- JSON Schema
- Django
- Google App Engine
- Jupyter Notebook
- Markdown
- Node.js and NPM
- Stylesheets
- Puppet
- Python Template Languages
- SQL Dialects
- SQL Resolution Scopes
- TypeScript

JavaScript

This feature is supported in the Professional edition only.

Warning! The following is only valid when JavaScript Support Plugin is installed and enabled!

File | Settings | Languages and Frameworks | JavaScript for Windows and Linux

PyCharm | Preferences | Languages and Frameworks | JavaScript for macOS

Ctrl+Alt+S



Use the pages in this section to specify the JavaScript language version for your project and configure JavaScript support in it.

In this part:

- [Code Quality Tools](#)
 - [Closure Linter](#)
 - [JSHint](#)
 - [ESLint](#)
 - [JSCS](#)
 - [JSLint](#)
- [Bower](#)
- [JavaScript Libraries](#)
 - [JavaScript Usage Scope](#)
- [Yeoman](#)
- [Meteor](#)
- [PhoneGap/Cordova](#)

ItemDescription

JavaScript Language Version	From this drop-down list, choose the JavaScript language version that represents the set of the language features to use in your project. The available options are: <ul style="list-style-type: none">- ECMAScript 3- ECMAScript 5.1- JavaScript 1.8.5- ECMAScript 6- React JSX- Flow
-----------------------------	---

Prefer Strict mode	Select this check box to have the strict mode standard applied to JavaScript code. This helps improve your code by enforcing best practices and suppressing insecure ones.
--------------------	--

Only type-based completion	<ul style="list-style-type: none">- When this check box is cleared, the completion list contains multiple variants in complicated cases.- When the check box is selected, the completion list strongly depends on the PyCharm type inference. This makes completion more precise but in case of poor inference the list may be empty. <p>By default, the check box is cleared.</p>
----------------------------	---

Flow executable	In this field, specify the path to the Flow executable file. The field is available only when Flow is chosen from the JavaScript Language Version drop-down list.
-----------------	--

Use Flow server for:	In this area, specify the basis for coding assistance by selecting or clearing the following check boxes: <ul style="list-style-type: none">- Type checking: When this check box is selected, syntax and error highlighting is provided based on the data received from the Flow server. When the check box is cleared, only the basic internal PyCharm highlighting is available.- Navigation, code completion, and type hinting: When this check box is selected, suggestion lists for reference resolution and code completion contain both suggestions retrieved from integration with Flow and suggestions calculated by PyCharm. When the check box is cleared, references are resolved through PyCharm calculation only.- Code highlighting and built-in inspections: Select this checkbox to power native Flow code highlighting and built-in inspections. Please note that enabling this option may cause performance problems. By default, the check box is cleared.
----------------------	--

The check boxes are available only when the path to the [Flow](#) executable file is specified.

Code Quality Tools

This feature is supported in the Professional edition only.

Warning! The following is only valid when JavaScript Support Plugin is installed and enabled!

File | Settings | Languages and Frameworks | JavaScript | Code Quality Tools for Windows and Linux

PyCharm | Preferences | Languages and Frameworks | JavaScript | Code Quality Tools for macOS

Ctrl+Alt+S



Use the pages under this node to enable the built-in JavaScript code verifiers and configure their behaviour.

In this part:

- [Closure Linter](#)
- [JSHint](#)
- [ESLint](#)
- [JSCS](#)
- [JSLint](#)

Closure Linter

This feature is supported in the Professional edition only.

Warning! The following is only valid when JavaScript Support Plugin is installed and enabled!

[File](#) | [Settings](#) | [Languages and Frameworks](#) | [JavaScript](#) | [Code Quality Tools](#) | [Closure Linter for Windows and Linux](#)

[PyCharm](#) | [Preferences](#) | [Languages and Frameworks](#) | [JavaScript](#) | [Code Quality Tools](#) | [Closure Linter for macOS](#)

Ctrl+Alt+S



Use this page to enable the JavaScript [Closure Linter](#) code verifier and to configure its behaviour.

ItemDescription

Enable	Select this check box to have Closure Linter applied to verify the code in the current project. After that the other controls in the page are enabled.
--------	--

Closure Linter executable file	In this text box specify the path to the Closure Linter executable file: <ul style="list-style-type: none">- <code><Python_home>\Scripts\jslint.exe</code> for Windows- <code>/usr/local/bin/gjslint</code> for Linux and macOS
--------------------------------	--

Configuration file	In this text box, specify the location of the previously created configuration text file to apply.
--------------------	--

This feature is supported in the Professional edition only.

Warning! The following is only valid when JavaScript Support Plugin is installed and enabled!

[File](#) | [Settings](#) | [Languages and Frameworks](#) | [JavaScript](#) | [Code Quality Tools](#) | [JSHint for Windows and Unix](#)

[PyCharm](#) | [Preferences](#) | [Languages and Frameworks](#) | [JavaScript](#) | [Code Quality Tools](#) | [JSHint for macOS](#)

Use this page to enable the built-in or downloaded JavaScript [JSHint](#) code verifier and configure its behaviour.

Item**Description**

Enable	Select this check box to have JSHint applied to verify the code in the current project. After that the other controls in the page are enabled.
Use config files	<p>Select this check box to have the code verified according to the settings from a configuration file. A configuration file is a JSON file with the extension <code>.jshintrc</code> that specifies which JSHint options should be enabled or disabled. PyCharm will look for a <code>.jshintrc</code> file in the working directory. If the search fails, PyCharm will search in the parent folder, then again in the parent folder. The process is repeated until PyCharm finds a <code>.jshintrc</code> or reaches the project root. To have PyCharm still run verification in this case, specify the default configuration file to use.</p> <p>Starting with PyCharm version 6.0.1, the <code>globals</code> setting in <code>.jshintrc</code> is supported. For earlier versions, a <code>Predefined (, separated)</code> tree node is displayed in the bottom of the tree in the UI.</p> <p>When this check box is selected, the Options area is hidden and the default verification settings are unavailable.</p>
Version	<p>Use this drop-down list to choose the version of the tool to apply.</p> <p>PyCharm comes bundled with <code>version 1.0.0</code>, which is used by default. PyCharm provides the ability to download another version, which is not bundled. Actually, the alternative version is downloaded only once, whereupon it is available without download.</p>
Options	<p>In this area, configure JSHint behaviour by enabling or disabling JSHint options. To enable or disable an option, select or clear the corresponding check box respectively. The controls in the area fall into two groups:</p> <ul style="list-style-type: none"> - Enforcing options: select the check boxes in this group to enable very strict behaviour of the verification tool and thus allow only safe JavaScript. - Relaxing options: select/clear the check boxes in this area to suppress warnings when certain types of discrepancies are detected. - Environments: select/clear these check boxes to specify for which environments you want global properties predefined. <p>The area is available only when the Use config files check box is cleared.</p>

This feature is supported in the Professional edition only.

Warning! The following is only valid when JavaScript Support Plugin is installed and enabled!

File | Settings | Languages and Frameworks | JavaScript | Code Quality Tools | ESLint for Windows and Linux

PyCharm | Preferences | Languages and Frameworks | JavaScript | Code Quality Tools | ESLint for macOS

Ctrl+Alt+S



Use this page to enable the JavaScript [ESLint](#) code verifier and to configure its behaviour. If you are using [JavaScript Standard Style](#) in your project, you can enable linting with it here.

To configure PyCharm code style options for JavaScript to follow the main rules Standard declares so they are applied when you type the code or reformat it, open the [Code Style. JavaScript](#) page (File | Settings | Editor | Code Style | JavaScript for Windows and Linux or PyCharm | Preferences | Editor | Code Style | JavaScript for macOS), click Set from, and then choose Predefined Style | JavaScript Standard Style. The style will replace your current scheme.

ItemDescription

Enable	Select this check box to have ESLint/Standard applied to verify the code in the current project. After that the other controls in the page are enabled.
Node Interpreter	In this field, specify the NodeJS installation home. Type the path to the NodeJS executable file manually, or click the  button and select the location in the dialog box, that opens. If you have appointed one of the installations as default , the field displays the path to its executable file.
ESLint Package	In this field, specify the location of the <code>eslint</code> package installed in the current project, see Installing ESLint or the <code>standard</code> package installed in your project or globally, see Installing JavaScript Standard .
Configuration file	In this area, appoint the configuration to use. By default, PyCharm first looks for a <code>.eslintrc</code> configuration file. PyCharm starts the search from the folder where the file to be checked is stored, then searches in the parent folder, and so on until reaches the project root. If no <code>.eslintrc</code> file is found, ESLint uses its default embedded configuration file. Accordingly, you have to define the configuration to apply either in a <code>.eslintrc</code> configuration file, or in a custom JSON configuration file, or rely on the default embedded configuration. <ul style="list-style-type: none"> - To have PyCharm look for a <code>.eslintrc</code> file, choose the Search for <code>.eslintrc</code> option. If no <code>.eslintrc</code> file is found, the default embedded configuration file will be used. - To use a custom file, choose the Configuration File option and specify the location fo the file in the Path field. Choose the path from the drop-down list, or type it manually, or click the  button and select the relevant file from the dialog box that opens.
Additional Rules Directory	In this field, specify the location of the files with additional code verification rules. These rules will be applied after the rules from <code>.eslintrc</code> or the above specified custom configuration file and accordingly will override them.
Extra ESLint Options	In this field, specify additional command line options to run ESLint with using spaces as separators. See ESLint Command Line Interface Options for details.

JSCS

This feature is supported in the Professional edition only.

Warning! The following is only valid when JavaScript Support Plugin is installed and enabled!

[File](#) | [Settings](#) | [Languages and Frameworks](#) | [JavaScript](#) | [Code Quality Tools](#) | [JSCS for Windows and Linux](#)

[PyCharm](#) | [Preferences](#) | [Languages and Frameworks](#) | [JavaScript](#) | [Code Quality Tools](#) | [JSCS for macOS](#)

Ctrl+Alt+S



Use this page to enable the JavaScript [JSCS](#) code verifier and to configure its behaviour.

ItemDescription

Enable	Select this check box to have JSCS applied to verify the code in the current project. After that the other controls in the page are enabled.
Node Interpreter	In this field, specify the NodeJS installation home. Type the path to the NodeJS executable file manually, or click the  button and select the location in the dialog box, that opens. If you have appointed one of the installations as default , the field displays the path to its executable file.
JSCS Package	In this field, specify the location of the <code>jscs</code> package installed in the current project, see Installing JSCS .
Configuration File	<p>In this area, appoint the configuration to use.</p> <p>By default, PyCharm first looks for a <code>jscsConfig</code> property in the <code>package.json</code> file of the current project. If no such property is found, PyCharm looks for a <code>.jscsrc</code> or a <code>.jscs.json</code> configuration file. PyCharm starts the search from the folder where the file to be checked is stored, then searches in the parent folder, and so on until reaches the project root. Accordingly, you have to define the configuration to apply either as a <code>jscsConfig</code> property in the <code>package.json</code> file or in a <code>.jscsrc</code> or a <code>.jscs.json</code> configuration file, or in a custom JSON configuration file.</p> <p>You can also apply a predefined set of rules, either independently or in combination with a configuration file. In the latter case, the rules from the configuration file override the predefined rules.</p> <ul style="list-style-type: none">- To have PyCharm look for a <code>jscsConfig</code> property in the <code>package.json</code> file or for a <code>.jscsrc</code> or a <code>.jscs.json</code> file, choose the Search for config(s) option.- To use a custom file, choose the Configuration File option and specify the location fo the file in the Path field. Choose the path from the drop-down list, or type it manually, or click the  button and select the relevant file from the dialog box that opens.- To have a predefined set or rules applied, choose the desired set from the Code Style Preset drop-down list.
Code Style Preset	From this drop-down list, choose the set of predefined rules associated with the code style you use.

This feature is supported in the Professional edition only.

Warning! The following is only valid when JavaScript Support Plugin is installed and enabled!

File | Settings | Languages and Frameworks | JavaScript | Code Quality Tools | JSLint for Windows and Linux

PyCharm | Preferences | Languages and Frameworks | JavaScript | Code Quality Tools | JSLint for macOS

Ctrl+Alt+S



Use this page to enable the built-in JavaScript [JSLint](#) code verifier and configure its behaviour.

ItemDescription

Enable	Select this check box to have JSLint applied to verify the code in the current project. After that the other controls in the page are enabled.
Tolerate	In this area, specify the discrepancies you want JSLint to consider worth reporting by enabling or disabling JSLint options . By default, discrepancies of all types are considered worth reporting. To have the tool skip discrepancies of a certain type, select the check box next to this type.
Assume	In this area, specify for which environments you want global properties predefined.
Stop on first error	Select this check box to have the verification process suspend as soon as the first valuable discrepancy is detected.
Safe Subset	Select this check box to have ADSafe safe subset rules enforced.
Verify ADsafe	Select this check box to have the ADSafe rules enforced.
Indentation	In this text box, type the number of spaces to be used for indentation. This setting corresponds to the <code>indent</code> option.
Maximum line length	In this text box, limit the number of characters acceptable in one line. This setting corresponds to the <code>maxLen</code> option.
Maximum number of errors	In this text box, specify the maximum number of warnings that can be reported. The default setting is 50. This setting corresponds to the <code>maxerr</code> option.
Predefined	In this text box, specify the predefined global variables by names or through object keys. Use commas (,) as separators.

Note This setting corresponds to the `predef` option.

Validate also	<p>In this area, enable or disable additional verification of HTML, CSS, or JSON context and configure the checking mode, when enabled.</p> <ul style="list-style-type: none"> – HTML: select this check box to have HTML context verified. Specify how you want HTML event handlers and HTML fragments treated: by default, they are reported as errors. To have the tool skip them, select the Tolerate HTML event handlers and Tolerate HTML fragments check boxes respectively. – CSS: select this check box to have CSS context verified. Specify how you want CSS workarounds treated: by default, they are reported as errors. To have the tool skip them, select the Tolerate CSS workarounds check box. – JSON: select this check box to have HTML context verified.
---------------	--

Bower

This feature is supported in the Professional edition only.

Warning! The following is only valid when JavaScript Support Plugin is installed and enabled!

File | Settings | Languages and Frameworks | JavaScript | Bower for Windows and Unix

PyCharm | Preferences | Languages and Frameworks | JavaScript | Bower for macOS

On this page, specify the location of the **Bower package manager** executable and configuration files and install, uninstall, or upgrade external tools through it.

ItemDescription

Node interpreter	Specify here the location of the Node.js executable. Type the path manually, or choose it from the drop-down list if it was previously used, or click the Browse button  and choose the location in the dialog box that opens. See Node.js for details.
Bower executable	In this field, specify the location of the Bower executable file (<code>bower.cmd</code> or other depending on the operating system used). Type the path manually, or choose it from the drop-down list if it was previously used, or click the Browse button  and choose the location in the dialog box that opens.
bower.json	In this text box, specify the location of the bower.json file. Type the path manually, or click the Browse button  and choose the location in the dialog box that opens.
Dependencies	<p>The area shows a list of all the -dependent packages that are currently installed on your computer.</p> <ul style="list-style-type: none">– Package: this read-only field shows the name of a package, exactly as it should be referenced if you were installing it in the command line mode.– Version: this read-only field shows the version of the package installed on your computer.– Latest: this read-only field shows the latest released version of the package. If a package is not up-to-date, it is marked with a blue arrow .– Click  to have a new package installed. In the Available Packages dialog box that opens, select the relevant package. Click Install Package when ready.– Click  to have the selected package removed.– Click  to have the current version of the selected package replaced with the latest released version. The button is enabled only when the selected project is not up-to-date.

This feature is supported in the Professional edition only.

Warning! The following is only valid when JavaScript Support Plugin is installed and enabled!

File | Settings | Languages and Frameworks | JavaScript | Libraries for Windows and Linux

PyCharm | Preferences | Languages and Frameworks | JavaScript | Libraries for macOS

Ctrl+Alt+S



Use this page to set up additional JavaScript libraries to expand the basic assistance provided through the JavaScript [plugin](#). Note that the JavaScript libraries are global.

In this section:

- [Libraries](#)
- [Buttons](#)
- [New Library / Edit Library Dialog Box](#)

Libraries

ItemDescription

Enabled	Each check box in this column shows whether the corresponding library is available or not: <ul style="list-style-type: none"> – A check box is selected when the corresponding library is defined and enabled in the current project. – A check box is marked with a Dash (-) if the corresponding library is defined and enabled on a more detailed level, for example, in a directory or in a file. – A check box is cleared when the corresponding library is disabled.
---------	---

Name	This column shows the names of the directories defined for a project.
------	---

Type	This column shows library types.
------	----------------------------------

Buttons

ItemDescription

Add	Click this button to create a new JavaScript library in the New Library dialog box. Refer to the section Configuring JavaScript Libraries .
Edit	Click this button to change the name and contents of the selected library in the Edit Library dialog box.
Remove	Click this button to delete the selected library.
Download	Click this button to open the Download Library dialog box, where you can have PyCharm download and install one of the popular official JavaScript-related libraries, such as: <ul style="list-style-type: none"> – Dojo – ExtJS – jQuery – jQuery UI – Prototype – etc. <p>Besides the above listed official libraries, you can download stubs for TypeScript definition files.</p> <p>Choose the group of libraries in the drop-down list. The available options are Official libraries and TypeScript community stubs. Depending on your choice, PyCharm displays a list of available libraries. Select the one to be downloaded and installed, and click Download and Install. You return to the JavaScript Libraries page where the new library is added to the list. Click OK to save the settings.</p>
Manage scopes	Click this button to configure libraries to be used for specific files and/or directories in the JavaScript. Usage Scope dialog box.

New Library / Edit Library Dialog Box

ItemDescription

Name	Specify the library name.
Framework type	From this drop-down list, choose the framework to configure as a library.
Version	In this text box, specify the version of the selected framework to use.
Visibility	In this area, specify where you want the library to be available for associating with files and folders. The available options are: <ul style="list-style-type: none"> – Current project: when this option is chosen, the library can be associated with files and folders within the current project only. If you later try to use the framework with another project, you will have to configure the library anew. – Global: choose this option to enable associating the library with any of your PyCharm projects.
Files	In this section, set up the library contents.
+ (Add)	Click this button to attach a JavaScript file or directory from the file system.

 (Remove)	Click this button to detach the selected file or directory from a library.
Name	This read-only column shows the name of the selected library file or the names of relevant library files from the selected directory.
Type	Click the column to show the drop-down list of the available versions of library files or directories: debug or release.

PyCharm enables you to create a library containing just one `.js` file, if this file is located on the Internet and can be accessed over HTTP. If you refer to a JavaScript library that is not yet available locally, but is available online, use the Download library [intention action](#):



The library will be placed to the user home directory, and will appear in the list of configured libraries in the JavaScript - Libraries page of the Settings dialog.

This feature is supported in the Professional edition only.

Warning! The following is only valid when JavaScript Support Plugin is installed and enabled!

File | Settings | JavaScript | Libraries - Manage Scopes for Windows and Linux

PyCharm | Preferences | JavaScript | Libraries - Manage Scopes for macOS

Ctrl+Alt+S



The dialog box opens when you click the Manage Scopes button in the [JavaScript. Libraries](#) page.

Use this dialog box to define the scopes where the JavaScript libraries should apply for code completion, highlighting and navigation. The scopes may cover the whole project, directories and even individual files. This helps make JavaScript code completion more precise, and avoid too long suggestion lists.

ItemDescription

File/Directory This column displays the project tree view.

Library This column displays libraries for a file or directory, if applicable.

If a library can be specified for a certain node of the project tree view, click the Library column for a selected file or directory, and choose the desired library from the list of available libraries.

If JavaScript libraries are not applicable to a particular node, 'N/A' is shown in grey font.

Yeoman

This feature is supported in the Professional edition only.

Warning! The following is only valid when Yeoman and Node.js Plugins are installed and enabled!

File | Settings | Languages and Frameworks | JavaScript | Yeoman for Windows and Linux

PyCharm | Preferences | Languages and Frameworks | JavaScript | Yeoman for macOS

Ctrl+Alt+S



The page is available only when the **Yeoman** and the **NodeJS** plugins are installed and enabled. The plugins are not bundled with PyCharm, but they can be installed from the **JetBrains plugin repository** as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#). Once enabled, the plugins are available at the IDE level, that is, you can use them in all your PyCharm projects.

On this page, specify the location of the **Node.js** executable file and the [Yeoman project generator](#) package installed through **Node Package Manager**.

ItemDescription

Node interpreter	Specify here the location of the Node.js executable. Type the path manually, or choose it from the drop-down list if it was previously used, or click the Browse button  and choose the location in the dialog box that opens. See Node.js for details.
Yeoman yo package	In this field, specify the location of the global <code>yo</code> package, see Installing and Removing External Software Using Node Package Manager , section Installing an External Tool Globally.

This feature is supported in the Professional edition only.

Warning! The following is only valid when JavaScript Support and Meteor Plugins are installed and enabled!

File | Settings | Languages and Frameworks | JavaScript | Meteor for Windows and Linux

PyCharm | Preferences | Languages and Frameworks | JavaScript | Meteor for macOS

The page is available when the **Meteor** plugin is enabled. The plugin is not bundled with PyCharm, but it can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).

ItemDescription

Meteor executable In this field, specify the location of the **Meteor** executable file (see [Installing Meteor](#)).

Automatically exclude ".meteor/local" directory on open project

- By default, the check box is selected and the `.meteor/local` folder, which is intended for storing the built application, is automatically marked as **excluded** and is not involved in indexing.
- Clear the check box to have PyCharm show the `.meteor/local` folder and its contents in the Project tool window.

By default, **excluded** files are shown in the project tree. To have the `.meteor/local` folder hidden, click the ****** button on the toolbar of the Project tool window and remove a tick next to the Show Excluded Files option.

Enable Meteor 'Hot code push'

- Select this check box to activate the native [Meteor hot code pushes functionality](#) and thus have the changes made to have the client-side code during a debugging session applied. By default, the check box is selected. Changes made to the server-side code are uploaded through the PyCharm **Live Edit** functionality, see [Configuring the Update Policy for the Server Side Code](#).
- When the check box is cleared, PyCharm does not provide any way to apply the changes made to the client-side code during a debugging session.

Automatically import Meteor packages as external library

- When the check box is selected, PyCharm automatically imports the external packages from the `meteor/packages` file. As a result, PyCharm provides full range coding assistance: resolves references to **Meteor** built-in functions, for example, `check(true)`, and to functions from third-party packages, provides proper syntax and error highlighting, supports debugging with source maps, etc.
- When this check box is cleared, PyCharm does not automatically import the external packages from the `meteor/packages` file. As a result no coding assistance is provided. To improve the situation, open the `meteor/packages` file in the editor and click the Import packages as library link or run the `meteor --update` command.

By default, the check box is selected.

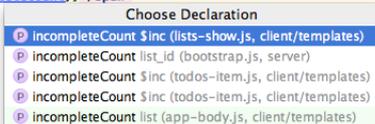
Weak search for Spacebars templates

- Select this check box to enable PyCharm to search inside Spacebars templates. When this check box is selected and you invoke the **Go to Declaration** action on a **helper**, PyCharm displays a list of all occurrences of this helper in templates. Choose the relevant one from the list.

```

{{#each Lists}}
<a href="{{pathFor 'ListsShow'}}" class="list-todo {{activeListClass}}" title="{{name}}">
  {{#if userId}}
  <span class="icon-lock"></span>
  {{/if}}
  {{#if incompleteCount}}
  <span class="count-list">{{incompleteCount}}</span>
  {{/if}}
  {{name}}
</a>
{{/each}}
</div>
</section>

```

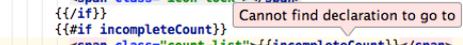


- When you invoke **Go to Declaration** with this check box cleared, PyCharm does not perform any search and displays a tooltip which the following message: Cannot find declaration to go to.

```

{{#each Lists}}
<a href="{{pathFor 'ListsShow'}}" class="list-todo {{activeListClass}}" title="{{name}}">
  {{#if userId}}
  <span class="icon-lock"></span>
  {{/if}}
  {{#if incompleteCount}}
  <span class="count-list">{{incompleteCount}}</span>
  {{/if}}
  {{name}}
</a>
{{/each}}

```



This feature is supported in the Professional edition only.

Warning! The following is only valid when JavaScript Support and PhoneGap/Cordova Plugins are installed and enabled!

File | Settings | Languages and Frameworks | JavaScript | PhoneGap/Cordova for Windows and Linux

PyCharm | Preferences | Languages and Frameworks | JavaScript | PhoneGap/Cordova for macOS

Ctrl+Alt+S 

The page is available when the **PhoneGap/Cordova** plugin is enabled. The plugin is not bundled with PyCharm, but it can be installed from the **JetBrains plugin repository** as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).

ItemDescription

PhoneGap/Cordova Executable Path	In this field, specify the location of the executable file <code>phonegap.cmd</code> , <code>cordova.cmd</code> , or <code>ionic.cmd</code> (see Installing PhoneGap/Cordova/Ionic).
PhoneGap/Cordova Version	This read-only field shows the installed PhoneGap/Cordova/Ionic version that PyCharm detects automatically.
PhoneGap/Cordova Working Directory	In this field, specify the folder under which the PhoneGap/Cordova/Ionic application files to run are stored.
Automatically exclude 'platforms' directory on open project	Select this check box to have the platforms directory marked as excluded automatically when you open the project. As a result, PyCharm ignores it during indexing, parsing, and code completion.
Plugins	<p>In this area, configure a list of plugins to use in your development by installing required packages. The list shows all the PhoneGap/Cordova/Ionic plugins that are currently installed on your computer, both at the global and at the project level.</p> <ul style="list-style-type: none"> - To install a plugin, click the Install button +. In the Available Packages dialog box that opens, select the required package. To have the plugin installed globally so it is accessible from all your PyCharm projects, select the Options check box and type <code>--global</code> in the text box. Click Install Package. - To remove a plugin, select it in the list and click the Uninstall button -. - To upgrade a plugin to the latest available version, select the plugin in the list and click the Upgrade button ↑. <p>See Apache Cordova Plugins and PhoneGap Plugins for information about plugins and their use.</p>

Schemas and DTDs

File | Settings | Languages and Frameworks | Schemas and DTDs for Windows and Linux

PyCharm | Preferences | Languages and Frameworks | Schemas and DTDs for macOS

Ctrl+Alt+S



The settings on this page define how your XML, HTML and XHTML files are validated.

– [External Schemas and DTDs](#)

– [Ignored Schemas and DTDs](#)

External Schemas and DTDs

Local XML schema (XSD) and DTD files that are used to validate your XML files are listed in this section.

Each entry is a mapping of a URI that may be referenced in your XML file onto an appropriate local schema or DTD file. Use ,  and  to create, remove and edit the mappings. See also, [Referencing XML Schemas and DTDs](#).

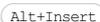
ItemKeyboardDescription Shortcut

URI	A URL of an XML schema (XSD) or DTD file (e.g. <code>http://www.example.org/xsds/example.xsd</code>) or a namespace URI (e.g. <code>http://www.example.org</code> , <code>urn:jboss:domain:1.0</code>). Readonly. If an XML file references the specified URI, it's validated according to a local file whose path is shown in the Location columns.
-----	---

Location	Path to corresponding local XSD or DTD file (readonly).
----------	---

Project	If a check box is not selected, a mapping is available in all of your projects. Select the mappings that you want to be available only in the current project.
---------	---

The selected mappings are stored in `.idea/misc.xml` or the project `.ipr` file. These files are normally shared between development team members through version control.

		Use this icon or shortcut to open the Map External Resource Dialog to define a new mapping between a URI and a local file.
---	---	--

		Use this icon or shortcut to remove the selected mappings from the list.
---	---	--

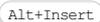
		Use this icon or shortcut to edit the selected mapping. (The Map External Resource Dialog dialog will open.)
---	---	--

Ignored Schemas and DTDs

URIs for ignored schemas and DTDs are listed in this section. (If an XML file references a URI listed in this section, PyCharm ignores that URI and doesn't mark it as an error in the editor.) Use ,  and  to add, remove and edit the URIs. See also, [Referencing XML Schemas and DTDs](#).

ItemKeyboard Shortcut

Description

		Use this icon or shortcut to add a new URI to the list.
---	---	---

		Use this icon or shortcut to edit the selected URI.
---	---	---

		Use this icon or shortcut to remove the selected URIs from the list. The corresponding URIs become available for mapping to local XSD or DTD files.
---	---	--

Ctrl+Alt+S



On this page:

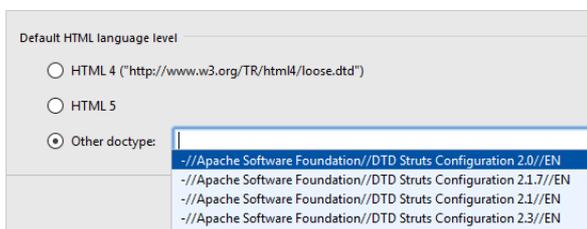
- [Default HTML Language Level](#)
- [Default XML Schema Version](#)

Default HTML Language Level

Normally, an HTML or an XHTML file has the `<!DOCTYPE>` declaration which states the **language level** of the used in the source code from the file. This language level is used as a standard against which the contents of the file are validated. If an HTML or an XHTML file does not have a `<!DOCTYPE>` declaration, the contents of the file will be validated against the default standard (schema). In the Default HTML Language Level area, choose the default schema to validate HTML and XHTML files without a `<!DOCTYPE>` declaration. The available options are:

- HTML 4 or HTML 5: Choose one of these options to have files treated as HTML 4 or HTML 5 and validated against one of these standards.
- Other doctype: Choose this option to have HTML files by default validated against a custom DTD or schema and specify the URL of the DTD or schema to be used.

Note that code completion is available in this field: press `Ctrl+Space` to see the list of suggested URLs.



Default XML Schema Version

In this area, choose the [XSD \(XML Schema Definition\) Schema](#) to validate XML files. The available options are:

- XML Schema 1.1 See [W3C XML Schema Definition Language \(XSD\) 1.1 Part 1: Structures](#) and [W3C XML Schema Definition Language \(XSD\) 1.1 Part 2: Datatypes](#) for details.
- XML Schema 1.0 See [XML Schema Part 1: Structures Second Edition](#) and [XML Schema Part 2: Datatypes Second Edition](#) for details.

Ctrl+Alt+S



Use this dialog to configure your .properties file. This file contains lists of catalog.xml files. The catalog.xml files contain the information on how to resolve various PUBLIC and SYSTEM identifiers during XML processing.

ItemDescription

Catalog property file

Use this field to specify the location of the .propeties file.

Type the path in the field, or click  and select the required file in the [dialog that opens](#) .

For more detailed description of the properties in the catalog property file, see the example of [Annotated CatalogManager properties file](#).

BDD

File | Settings | BDD for Windows and Linux

PyCharm | Preferences | BDD for macOS

Ctrl+Alt+S



Projects pane

ItemDescription

Projects

This pane displays the list of projects, opened in the same window. The first project in the list is the **primary project**.

Note This pane is not shown, if there is only one open project.

ItemDescription

Preferred framework

From this drop-down list, choose the preferred framework for BDD testing.

JSON Schema

File | Settings | Languages and Frameworks | JSON Schema for Windows and Linux

PyCharm | Preferences | Languages and Frameworks | JSON Schema for macOS

Ctrl+Alt+S



On this page, create a list of your custom [JSON schemas](#) in addition to the system schemas provided by PyCharm and configure the scope for each schema, that is, specify the `.json` files to be validated against it. The page consists of two panes: the List of Schemas pane and the Schema Details pane.

On this page:

- [List of Schemas Pane](#)
- [Schema Details Pane](#)
- [Handling Conflicts Among Scopes of Schemas](#)

List of Schemas Pane

The central pane of the page shows a list of custom JSON schemas against which `.json` files in the current project will be validated. Based on a schema, PyCharm checks the structure of `.json` configuration files, reports errors (for example, informs you about missing mandatory properties), provides code completion and documentation look-up.

The pane shows only **custom** schemas that you have previously downloaded or created yourself, in either case, a custom schema must meet the [JSON schema standards](#) and must be located under the project root.

System schemas that PyCharm provides for all supported frameworks and technologies are not shown in the list.

- To add a schema to the list, click **+** on the toolbar of the pane and select the relevant schema file under the project root in the dialog box that opens.
- To remove a schema from the list, select it and click **-** on the toolbar of the pane.
- To configure the scope of a schema, select the schema and specify the files to be validated against it in the right-hand pane of the page.

Schema Details Pane

The pane shows the details of the schema selected in the List of Schemas pane: the name of the selected schema, the `.json` file that implements it, and a list of files and folders that are validated against the schema.

The list contains the names of specific files, the names of entire directories, and filename patterns. You do not need to specify full paths to files and folders, PyCharm searches for files and folders with the specified names and the search is restricted to the the current project.

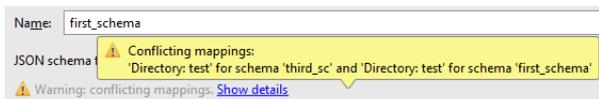
Based on the list, PyCharm internally creates a list of files to be validated. Each file is included in the list only once: if a file with the specified name is stored in a directory from the list or its name matches a pattern, the file is still validated only once.

- To add new files or folders to the list, click **+** on the toolbar of the pane and specify the file, folder, or a file pattern in the [Add JSON Schema Mapping Dialog](#) that opens.
- To remove an item from the list, select it and click **-** on the toolbar of the pane.

Handling Conflicts Among Scopes of Schemas

A conflict arises when a file, or a folder, or a pattern belongs to the scopes of two or more schemas. PyCharm analyzes scopes in two modes:

- **Static Analysis** detects conflicts in scopes of custom schemas. If a conflict is detected, PyCharm displays a warning in the Schema Details pane. To view the overlapping scopes, click the Show details link. PyCharm shows a pop-up message where the conflicting scopes and schemas are listed:



- **Dynamic Analysis** detects conflicts in scopes of both system and custom schemas. This type of analysis is started when you open a file that belongs to a certain scope. If a conflict is detected, PyCharm displays a warning at the top of the editor tab:

There are several JSON Schemas mapped to this file: first_schema; third_sc [Edit JSON Schema Mappings](#)

Click the link to open the [JSON Schema](#) page and edit the scope of the conflicting **custom** schema. Note that you cannot edit the scope of **system** schemas.

The dialog box opens when you select a `.json` schema in the list on the [JSON Schema](#) page and click **+** in the right-hand pane. In this dialog, specify the `.json` files to be validated against the selected schema.

Item **Description**

Files Under	Choose this option to add files stored in a specific folder name to validation. Type the name of the folder manually or click  and choose the folder in the dialog box that opens.
-------------	---

Filename pattern	Choose this option to have all files with the names that match a pattern validated.
------------------	---

File	Choose this option to add files with a specific name to validation. Type the filename manually or click  and choose the file in the dialog box that opens.
------	---

Django

File | Settings | Languages and Frameworks | Django for Windows and Linux

PyCharm | Preferences | Languages and Frameworks | Django for macOS

Ctrl+Alt+S



Use this page to configure [Django support](#) for each of the [projects opened in the same window](#).

On this page:

- [Projects pane](#)
- [Django support pane](#)

Projects pane

ItemDescription

Projects This pane displays the list of projects, opened in the same window. The first project in the list is the **primary project**.

Note This pane is not shown, if there is only one open project.

Django support pane

ItemDescription

Enable Django support The default state of this check box depends on the [project type](#). For the empty projects, Django support is disabled. For the Django projects it is enabled by default; you can clear this check box if required. In this case, the other fields become unavailable.

Django project root By default, this field shows the directory that stores `settings.py` and `manage.py` files of the application. If required, you can specify a different location.

Settings Click the browse button to select the desired settings file.
Use one of the following approaches:

- This can be any file with the name matching `*settings*.py`, located under the Django project root.
- Point to any Python package and store settings in `__init__.py`.

This latter approach is useful when you want to split settings to several modules and import them.

By default, PyCharm shows the `settings.py` file, located under the Django project root.

Manage.py tasks

Manage script Specify here the desired `manage.py` file for the current project.

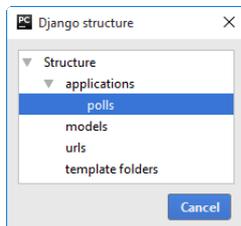
By default, PyCharm shows the `manage.py` file, located under the Django project root. Click  to select the desired manage file from the file system.

Environment variables Specify here the environment variables to be passed to the script.

Commands to skip Specify here the commands of `manage.py` which should be skipped.

If some of the `manage.py` utility commands take too long time to run, a message appears in the `manage.py@<project name>` tool window, suggesting you to include these commands into the skip list.

Show structure Click this button to show the structure of a Django project:



The users are advised to look into this window to see whether the Django project structure is correct.

Google App Engine

File | Settings | Google App Engine for Windows and Linux

PyCharm | Preferences | Google App Engine for macOS

Ctrl+Alt+S



Use this dialog to enable Google [App Engine](#) support, configure settings, and specify your App Engine credentials for each of the [projects opened in the same window](#).

In this section:

- [Projects pane](#)
- [Google App Engine support pane](#)

Projects pane

ItemDescription

Projects This pane displays the list of projects, opened in the same window. The first project in the list is the **primary project**.

Note This pane is not shown, if there is only one open project.

Google App Engine support pane

ItemDescription

Enable Google App Engine support Select this check box to enable Google App Engine support. If the check box is not selected, all the following fields become disabled.

SDK Directory Specify the path to the folder where the **Google App Engine SDK** is installed on your machine.

App Engine Account Settings In this section, select the way you log in to your Google account, and specify your credentials:

- Use passwordless login via OAuth2: select this option to use the `-oauth2` options in the `appcfg.py` file.
- Log in with email and password: select this option to log in using your email address and password, which you have to specify in the provided text fields.

Keep me signed in Select this check box to memorise your credentials.

Jupyter Notebook

File | Settings | Tools | IPython Notebook for Windows and Linux PyCharm | Preferences | Tools | IPython Notebook for macOS

ItemDescription

Python Notebook URL

In the text field, specify URL of the IPython Notebook server. This URL is used to run a cell.

If this field is left blank, then nothing is specified by default in the Start IPython Notebook dialog, and you will have to type the required URL yourself.

Markdown

Warning! The following is only valid when Markdown Support Plugin is installed and enabled!

File | Settings | Tools | Markdown for Windows and Linux

PyCharm | Preferences | Tools | Markdown for macOS

Ctrl+Alt+S



This page contains settings that customise the particular CSS used for previewing the Markdown documents. For example, for Darcula the dark theme is used, for Default - the light theme.

Warning! The following is only valid when Markdown Support Plugin is installed and enabled!

File | Settings | Tools | Markdown | Preview for Windows and Linux

PyCharm | Preferences | Tools | Markdown | Preview for macOS

Ctrl+Alt+S



ItemDescription

Preview browser	Select the engine of your preview browser.
Default editor layout	Select how the editor will be shown by default: both editor and preview panes, editor only, or preview only pane.
Use grayscale rendering for JavaFX preview	If this check box is selected, the antialiasing is turned on for the JavaFX preview.

Node.js and NPM

File | Settings | Languages & Frameworks | Node.js and NPM for Windows and Linux

PyCharm | Preferences | Languages & Frameworks | Node.js and NPM for macOS

Ctrl+Alt+S



This page appears in the Settings dialog box, when the **Node.js** plugin is enabled. The plugin is not bundled with PyCharm, but it can be installed from the **JetBrains plugin repository** as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).

ItemDescription

Node interpreter	<p>In this field, specify the local Node.js interpreter to use. Choose the interpreter from the drop-down list or click  and choose the interpreter in the dialog box that opens.</p> <p>The term local Node.js interpreter denotes a Node.js installation on your computer. The term remote Node.js interpreter denotes a Node.js installation on a remote host or in a virtual environment set up in a Vagrant instance. On the Node.js and NPM page, you can specify only local interpreters. Remote interpreters are configured in the Configure Node.js Remote Interpreter Dialog dialog which can be accessed only from the Run/Debug Configuration: Node JS dialog. See Configuring Node.js Interpreters for details.</p>
------------------	---

Version	This read-only field shows the current version of the runtime environment.
---------	--

Code Assistance	<p>When developing a Node.js application it can be convenient to have code completion, reference resolution, validation, and debugging capabilities for Node core modules (<code>fs</code> , <code>path</code> , <code>http</code> , etc.). However, these modules are compiled into the Node.js binary. PyCharm provides the ability to configure these sources as a JavaScript library and associate it with your project.</p> <ul style="list-style-type: none">– If the Node.js core module sources are not set up, PyCharm displays a notification Node.js Core Library is not enabled with an Enable button. Click this button to have PyCharm configure Node.js Core sources automatically. <p>When the configuration is completed, PyCharm displays information about the currently configured version, the notification Node.js Core Library is enabled, and adds two buttons: the Disable button and the Usage scope button.</p> <ul style="list-style-type: none">– If the library is set up, PyCharm displays information about the currently configured version, the notification Node.js Core Library is enabled, and adds two buttons: the Disable button and the Usage scope button.<ul style="list-style-type: none">– Click the Disable button to discard the configuration of the Node.js Core libraries in the current project.– Click the Usage scope button to associate the desired directories with libraries. <p>If necessary, configure the scope in which the Node.js Core sources are treated as libraries. Click the Usage scope button, and in the Usage Scope dialog box that opens, click the desired directories, and from the drop-down list select the newly configured Node.js core module sources library.</p>
-----------------	--

Packages	<p>A number of tools are started through Node.js, for example, the CoffeeScript, TypeScript, and Less compilers, YUI, UglifyJS, and Closure compressors, Karma test runner, Grunt task runner, etc. The Node Package Manager (npm) is the easiest way to install these tools, the more so that you have to install Node.js anyway. The Packages area shows a list of all the NPM-dependent packages that are currently installed on your computer.</p> <ul style="list-style-type: none">– Package: this read-only field shows the name of a package, exactly as it should be referenced if you were installing it in the command line mode.– Version: this read-only field shows the version of the package installed on your computer.– Latest: this read-only field shows the latest released version of the package. If a package is not up-to-date, it is marked with a blue arrow .– Click  to have a new package installed. In the Available Packages dialog box that opens, select the relevant package. To have the package installed globally, select the Options check box and type <code>-g</code> in the Options text box. Global installation makes the package available at the PyCharm level so it can be used in any PyCharm project. Click Install Package when ready.– Click  to have the selected package removed.– Click  to have the current version of the selected package replaced with the latest released version. The button is enabled only when the selected project is not up-to-date.
----------	--

Stylesheets

File | Settings | Languages and Frameworks | Stylesheets for Windows and Linux

PyCharm | Preferences | Languages and Frameworks | Stylesheets for macOS

The page shows a list of links to the pages where you can activate and configure various tools related to working with CSS, Sass, and SCSS code.

– [Compass](#)

– [Stylelint](#)

`Ctrl+Alt+S`

Use this page to integrate the [Compass framework](#) with PyCharm and thus enable compilation of Sass and SCSS files from Compass-specific projects into CSS. For details, see [Using File Watchers](#).

Prerequisites

Before you start working with Compass, make sure that the Sass support plugin is enabled. The plugin is bundled with PyCharm and is activated by default. If the plugin is not activated, enable it on the [Plugins](#) page of the [Settings / Preferences Dialog](#) as described in [Enabling and Disabling Plugins](#).

Make sure that [Ruby](#) is installed on your computer.

For more details, see [Working with Sass and SCSS in Compass Projects](#).

ItemDescription

Enable Compass support	Select this check box to activate Compass support including the possibility to compile Sass and SCSS files from your Compass project into CSS.
Compass executable file	In this field, specify the location of the <code>compass</code> executable file under the Ruby installation. Type the path manually, for example, <code>C:\Ruby200-x64\bin\compass</code> , or choose it from the drop-down list, or click the Browse button  and choose the location of the <code>compass</code> file in the dialog box that opens.
Config path	In this field, specify the location of the project Compass configuration file <code>config.rb</code> . Type the path manually, for example, <code>C:\my_projects\compass_project\config.rb</code> , or choose it from the drop-down list, or click the Browse button  and choose the location of the <code>compass</code> file in the dialog box that opens. The Compass configuration file <code>config.rb</code> is generated during project set-up through <code>compass create</code> or <code>compass init</code> commands.

Use this page to enable the CSS [Stylelint](#) code verifier in your project and to configure its behaviour.

ItemDescription

Enable	Select this check box to activate the Stylelint support.
Node Interpreter	In this field, specify the NodeJS installation home. Type the path to the NodeJS executable file manually, or click the  button and select the location in the dialog box, that opens. If you have appointed one of the installations as default , the field displays the path to its executable file.
Stylelint package	In this field, specify the location of the <code>stylelint</code> package installed globally or in the current project, see Using Stylelint Code Quality Tool .

Puppet

File | Settings | Languages and Frameworks | Puppet for Windows and Linux

PyCharm | Preferences | Languages and Frameworks | Puppet for macOS

Ctrl+Alt+S



Use this page to configure Puppet integration.

ItemDescription

Puppet language level	Use the drop-down list to select the supported language level (3.X or 4).
-----------------------	---

Path to librarian-puppet	Use this text box to specify the path to the gem <code>librarian-puppet</code> . Use the browse button to locate this gem in your file system.
--------------------------	--

ItemDescription

Synchronize environment with VCS branch name

Select this check box to avoid creating or choosing different Puppet environment on changing branches.
If a [Git branch changes](#), the current environment changes accordingly, thus changing the set of the used libraries automatically.

List of environments and modulepaths

The list is created by **+** and **-** buttons. Besides adding and deleting, it is possible to edit the selected environments.

Note Note the following:

- A **modulepath** is a directory that contains modules.
- The list of modules attached via the modulepath is shown under the node **External Libraries** in the [Project tool window](#).
- To apply a certain environment manually, it's necessary to select it in the list and click Apply.

+

Click this button to add a new environment to the list. Clicking **+** opens the Create New Environment dialog:

ItemDescription

Environment name

This text field shows the suggested environment name. Accept default, or type your preferred name.

Use default modulepath

Select this check box to choose the default modulepath for the created environment.
If this check box is selected, the Custom modulepath field is disabled.

Note that all the existing defaults are already described in the modulepath.

Custom modulepath

This text field is only enabled, when Use default modulepath check box is not selected.

Use this field to specify your preferred modulepath for a generated environment. Type the path manually, or click  and choose the desired directory in the [Select modulepath for this environment](#) dialog.

-

Click this button to delete the selected environment from the list.



Click this button to change the selected environment. Clicking  opens the Edit Environment dialog with the following controls:

ItemDescription

Environment name

This text field shows the existing environment name. Type your preferred name.

Use default modulepath

Select this check box to choose the default modulepath for the selected environment.
If this check box is selected, the Custom modulepath field is disabled.

Custom modulepath

This text field is only enabled, when Use default modulepath check box is not selected.

Use this field to specify your preferred modulepath for the environment. Type the path manually, or click  and choose the desired directory in the [Select modulepath for this environment](#) dialog.

Python Template Languages

File | Settings | Python Template Languages for Windows and Linux

PyCharm | Preferences | Python Template Languages for macOS

Ctrl+Alt+S



Use this page to select the template language to be used in each of the [projects opened in the same window](#), specify template folders .

In this section:

- [Projects pane](#)
- [Template languages pane](#)

Projects pane

ItemDescription

Projects This pane displays the list of projects, opened in the same window. The first project in the list is the **primary project**.

Note This pane is not shown, if there is only one open project.

Template languages pane

ItemDescription

Template language Select the desired template language for your project from the drop-down list. For example, you can mark your project as using Django templates even if it is not a Django project.

The available template languages are:

- None: is this option is selected, the project doesn't use any template language.
- [Django](#)
- [Mako](#)
- [Jinja2](#)
- [Web2Py](#)
- [Chameleon](#)

Template file types In this section, specify the types of files, where templates will be recognized.

Press **+** to show the list of available file types, and choose the desired one.

Press **-** to delete the selected file type. Note that the default file types (HTML, XHTML, and XML) may not be deleted.

SQL Dialects

File | Settings | Languages and Frameworks | SQL Dialects for Windows and Linux

PyCharm | Preferences | Languages and Frameworks | SQL Dialects for macOS

This page lets you specify the SQL dialects (DBMS-specific versions of SQL) used in various scopes.

Note For this page to be available, the Database Tools and SQL plugin must be enabled. See [Enabling and Disabling Plugins](#).

- [Dialect settings](#)
- [Dialect options](#)
- [Example](#)

Dialect settings

Item	Description
Global SQL Dialect	The SQL dialect for all the <code>.sql</code> and <code>.ddl</code> files on your computer; may be redefined in narrower scopes - at the project level, and/or for individual files and directories.
Project SQL Dialect	The SQL dialect for all the <code>.sql</code> and <code>.ddl</code> files in your current project. If <code><None></code> is specified, the global SQL dialect is inherited.
Path / SQL Dialect	The SQL dialects for individual files and directories - if different from the global or project dialect. If nothing is specified in this section, all the <code>.sql</code> and <code>.ddl</code> in your project inherit the project dialect, and all the files that are outside the project - the global dialect. To specify a dialect for a file or directory, click <code>+</code> and select the file or directory in the dialog that opens. Then click <code>✓</code> or the SQL Dialect cell, and select the dialect. The dialects specified explicitly are shown in black. The inherited dialects (unless you close the dialog) are shown in gray italic.

Dialect options

When specifying a dialect, in addition to particular dialects, you can select:

- `<None>` or `<Clear>`. As a result, a dialect from a higher level is inherited.
- `<Generic SQL>`. This means that no particular dialect is specified. As a result, basic SQL92-based coding assistance is provided including completion and highlighting for SQL keywords, and table and column names. Syntax error highlighting is not available. So the file contents are always shown as syntactically correct. Also, automatic code reformatting isn't possible.

Example

Say, most of the SQL script files on your computer are for PostgreSQL. In the current project, you are developing the scripts for Oracle but in one of the directories in your project there are the scripts for MySQL. In such a situation, you'd specify:

- Global SQL dialect: PostgreSQL
- Project SQL dialect: Oracle
- `<PathToMySQLScriptsFolder>`: MySQL

SQL Resolution Scopes

File | Settings | Languages and Frameworks | SQL Resolution Scopes for Windows and Linux

PyCharm | Preferences | Languages and Frameworks | SQL Resolution Scopes for macOS

This page lets you specify the data sources, databases and schemas that should be used to resolve unqualified ("short") names of database object in your SQL files.

Note For this page to be available, the Database Tools and SQL plugin must be enabled. See [Enabling and Disabling Plugins](#).

[Settings](#)

[Example](#)

Settings

ItemDescription

Project mapping	The set of data sources, databases and schemas used by default by all the SQL files in your project to resolve unqualified names of database objects.
-----------------	---

Path /	Mappings for individual files and directories.
--------	--

Resolution Scope	If nothing is specified in this section, all the SQL files in your project use the project mapping.
------------------	---

To specify a different mapping for a file or directory, click **+** and select the file or directory in the dialog that opens. Then click **🗑** or the Resolution Scope cell, and select the necessary data sources, databases and schemas.

The mappings specified explicitly are shown in black. The mappings inherited from a higher level (unless you close the dialog) are shown in gray italic.

Example

You have two data sources, one for your production database and the other one - for your test database. The tables in both databases have the same names but slightly different structures. And you keep the SQL scripts for your production and test databases separately, in two different folders.

In such a situation, you'd map the folder with the production scripts onto the production data source, and the folder with the test scripts onto the test data source.

The other possibility would be to use the project mapping for your production data source and specify the mapping for the test scripts folder separately, or vice versa.

TypeScript

File | Settings | Languages & Frameworks | TypeScript for Windows and Linux

PyCharm | Preferences | Languages & Frameworks | TypeScript for macOS

Ctrl+Alt+S



In this dialog, activate, deactivate, and configure the behaviour of the built-in **TypeScript compiler** that compiles TypeScript code into JavaScript so the code can be executed in a browser and generates sources maps that ensure correspondence between the lines in the generated JavaScript code and the lines in the original TypeScript code.

ItemDescription

Node interpreter	In this field, specify the location of the <code>Node.js</code> executable file. In most cases, PyCharm detects the <code>Node.js</code> executable and fills in the field automatically.
TypeScript version	<p>In this area, specify the version of the compiler to use (PyCharm displays the currently chosen version):</p> <ol style="list-style-type: none">Click Edit.In the Configure TypeScript Compiler dialog box that opens, choose one of the following options:<ul style="list-style-type: none">Detect: if you choose this option, PyCharm searches for a <code>typescript</code> package in the current project. If a <code>typescript</code> package is found, PyCharm uses it. Otherwise the default bundled package is used. This option is chosen by default.Bundled: if you choose this option, PyCharm uses it without attempting to find another <code>typescript</code> package.Custom directory: choose this option to use a custom version of the compiler. In the text box, specify the location of the <code>typescriptServices.js</code>, <code>lib.d.ts</code>, and <code>lib.es6.d.ts</code> files downloaded from https://github.com/Microsoft/TypeScript/.
Use TypeScript Service	<p>Select this check box to get native support from the TypeScript Language Service according to the up-to-date specifications. In this case, syntax and error highlighting is performed based on the annotations retrieved from the TypeScript Language Service while code completion lists contain both suggestions from the TypeScript Language Service and suggestions calculated by PyCharm itself. To get only suggestions from PyCharm, click Configure and clear the Code completion check box in the Service Options dialog box that opens.</p> <p>PyCharm supports integration with the Angular language service developed by the Angular team to improve code analysis and completion for Angular-TypeScript projects. Note that the Angular language service works only with the projects that use Angular 2.3.1 or higher and TypeScript version compatible with it. The Angular language service is activated by default so PyCharm starts it automatically together with the TypeScript service and shows all the errors and warnings in your TypeScript and HTML files both in the editor and in the TypeScript Compiler Tool Window. However if you still need to activate the service, click Configure and select the Use Angular service check box in the Service Options dialog box that opens.</p> <p>By default, the Use TypeScript Service check box is selected.</p> <p>In the Default options field, specify the command line options to be passed to the compiler when the <code>tsconfig.json</code> file is not found. See the list of acceptable options at TSC arguments. Note that, the <code>-w</code> or <code>--watch</code> option (Watch input files) is irrelevant.</p>
Enable TypeScript Compiler	Select this check box to activate the built-in compiler. When the check box is selected, syntax highlighting and code completion are provided based only on the data from the built-in Typescript compiler. After you select this check box, the fields below in the area become active and you can configure the behaviour of the compiler. By default, the check box is cleared.
Track changes	<ul style="list-style-type: none">When this check box is selected, the built-in compiler "wakes up" upon any change to a TypeScript file.When this check box is cleared, the built-in compiler ignores changes to TypeScript files. To re-activate the compiler, open the TypeScript Compiler Tool Window (View Tool Windows TypeScript Compiler), and click the Compile All button  on the toolbar. <p>If you have not opened the TypeScript Compiler Tool Window yet and it is not available from the View menu, choose Help Find Action, then find and launch the TypeScript Compile All action from the list.</p>
Scope	<p>From this drop-down list, choose the scope to apply the compiler in. The available options are:</p> <ul style="list-style-type: none">Project Files: all the files within the project content roots (see Content Root and Configuring Content Roots).Project Production Files: all the files within the project content roots excluding test sources.Project Test Files: all the files within the project test source roots.Open Files: all the files that are currently opened in the editor. <p>VCS Scopes: these scopes are only available if your project is under version control.</p> <ul style="list-style-type: none">Changed Files: all changed files, that is, all files associated with all existing changelists.Default: all the files associated with the changelist <code>Default</code>. <p>Alternatively, click the Browse button and configure a custom scope in the Scopes dialog box that opens. For more details on scopes, see the pages Scopes and Scopes dialog.</p>
Use tsconfig.json	Choose this option to have the built-in compiler analyze the code according to the settings specified in the <code>tsconfig.json</code> file. When you open a project, PyCharm starts searching for a <code>tsconfig.json</code> file in it. If a <code>tsconfig.json</code> file is found, the compiler uses the options specified in it. Otherwise an error is reported.
Set options manually	<p>Choose this option to configure the behaviour of the built-in compiler manually:</p> <ul style="list-style-type: none">Select the Generate source maps check box to generate source maps that set correspondence between lines in your TypeScript code and in the generated JavaScript code, otherwise your breakpoints will not be recognised and processed correctly.Select the Compile main file only check box to have PyCharm compile only a specific file and the files that are referenced from it and ignore all the other files in the project. This may be helpful if you have a dedicated <code>main.ts</code> file which only references other files.Select the Use output path check box to have the built-in compiler store the generated JavaScript files and source map in a custom folder. Specify the path to this folder explicitly or use one of the listed available macros in the format: <code> \${<macro_name> } </code>. The available macros are:<ul style="list-style-type: none"><code>FileDir</code>: the path to the folder where the file is stored.<code>FileRelativeDir</code>: the path to the file directory relative to the project root.<code>FileDirRelativeToProjectRoot</code>: the path to the file directory relative to the project root.<code>ModuleFileDir</code>: the path to the project root folder.<code>SourcePath</code>: the path to the source folder under the content root to which the file belongs, see Configuring Content Roots and Configuring Folders Within a Content Root.

- `FileDirRelativeToSourcePath` : the path to the file relative to the source folder under the content root to the file belongs.

Ctrl+Alt+S



Use this page to activate integration with the [TSLint](#) code verifier and configure its behaviour.

ItemDescription

Enable	Select this check box to have TSLint applied to verify the code in the current project. After that the other controls in the page are enabled.
Node Interpreter	In this field, specify the NodeJS installation home. Type the path to the NodeJS executable file manually, or click the  button and select the location in the dialog box, that opens. If you have appointed one of the installations as default , the field displays the path to its executable file.
TSLint Package	In this field, specify the location of the <code>tslint</code> package installed globally or in the current project, see Using TSLint Code Quality Tool .
Configuration File	In this area, appoint the configuration to use. By default, PyCharm first looks for a <code>tslint.json</code> configuration file. PyCharm starts the search from the folder where the file to be checked is stored, then searches in the parent folder, and so on until reaches the project root. If no <code>tslint.json</code> file is found, TSLint uses its default embedded configuration file. Accordingly, you have to define the configuration to apply either in a <code>tslint.json</code> configuration file, or in a custom JSON configuration file, or rely on the default embedded configuration. <ul style="list-style-type: none">- To have PyCharm look for a <code>tslint.json</code> file, choose the Search for tslint.json option. If no <code>tslint.json</code> file is found, the default embedded configuration file will be used.- To use a custom file, choose the Configuration File option and specify the location fo the file in the Path field. Choose the path from the drop-down list, or type it manually, or click the  button and select the relevant file from the dialog box that opens.
Additional Rules Directory	In this field, specify the location of the files with additional code verification rules. These rules will be applied after the rules from <code>tslint.json</code> or the above specified custom configuration file and accordingly will override them.

Ctrl+Alt+S



The dialog box opens when you select the Use TypeScript Service check box and click Configure next to it on the [TypeScript](#) page. Use the dialog box to configure TypeScript code completion and activate or disable integration with the [Angular language service](#) in [TypeScript-Angular](#) projects.

ItemDescription

- Code completion
- Select this check box to get native support from the [TypeScript Language Service](#) according to the up-to-date specifications. In this case, syntax and error highlighting is performed based on the annotations retrieved from the [TypeScript Language Service](#) while code completion lists contain both suggestions from the [TypeScript Language Service](#) and suggestions calculated by PyCharm itself.
 - Clear the check box to have only suggestions from PyCharm.

By default, the check box is selected.

- Use Angular service
- The check box is only available in [TypeScript-Angular](#) projects when you are using [Angular 2.3.1](#) or higher with a compatible version of [TypeScript](#) and the `@angular/language-service` package is installed in the project root. Select the check box to activate the [Angular language service](#) in your project. The [Angular language service](#) is activated by default so PyCharm starts it automatically together with the [TypeScript service](#) and shows all the errors and warnings in your [TypeScript](#) and [HTML](#) files both in the editor and in the [TypeScript Compiler Tool Window](#).

Tools

File | Settings | Tools for Windows and Linux

PyCharm | Preferences | Tools for macOS

Ctrl+Alt+S



When you select the Tools category in the left-hand pane, its main subcategories are listed in the right-hand part of the dialog.

- [Web Browsers](#)
- [File Watchers](#)
- [External Tools](#)
- [Terminal](#)
- [Database](#)
- [SSH Terminal](#)
- [Diagrams](#)
- [Diff & Merge](#)
- [Docker Machine](#)
- [Python External Documentation](#)
- [Python Integrated Tools](#)
- [Remote SSH External Tools](#)
- [Server Certificates](#)
- [Settings Repository](#)
- [Startup Tasks](#)
- [Tasks](#)
- [Vagrant](#)

Web Browsers

File | Settings | Tools | Web Browsers for Windows and Linux

PyCharm | Preferences | Tools | Web Browsers for macOS

Ctrl+Alt+S



On this page

- Integrate installations of Web browsers with PyCharm, activate or deactivate launching Web browsers from PyCharm .
- Specify whether a browser will be launched by running its executable file or through the default system command .
- Appoint the **default PyCharm browser** in which PyCharm will **open HTML files** upon request by default, that is, when no browser is specified explicitly .

Browsers

In this section, specify which browsers will be available for previewing HTML output. The section shows a **predefined list of browsers**, possibly extended with previously configured **custom browser installations**. Each browser is presented as a separate table row.

PyCharm is shipped with a predefined list of most popular browsers which you may like to install and use. The items are added to the list in advanced and are not based on the information on actually installed browsers. PyCharm presumes that you install browsers according to a standard procedure. Based on this assumption, each browser in this predefined list is assigned an **alias** which stands for the path to its executable file, as PyCharm supposes it to be. If in your actual browser installation the path to the executable file is different, you need to specify it explicitly in the Path field.

In addition to the predefined browsers, you can configure as many custom browser installations as you need using the controls on the toolbar.

ColumnDescription

Active	Select this check box to enable the use the respective browser from PyCharm. The browser will be added to the context menu of the Open in Browse menu item and its icon will be displayed in the Browsers pop-up toolbar. If this check box is cleared, the corresponding browser icon will not appear in the icons toolbar or pop-up menu.
Name	In this column, specify the browser name.
Family	In this column, specify the family to which the browser belongs.
Path	In this column, specify the path to the executable. If the browser was installed according to a standard installation procedure, most likely the alias shown in the Path field points at the right location of the executable file. To specify the path explicitly, click  in the Path field and choose the actual location of the executable file in the dialog box that opens. In the dialog that opens , choose the path to the executable file of the corresponding browser.

Toolbar

ItemDescription

	Click this button to add a custom browser to the list.
	Click this button to delete the selected customer browser from the list. Note that you cannot delete the browsers from the predefined list.
	Click this button to specify a custom profile for Firefox or a browser of the Chrome family. The button is available only when Firefox and Chrome are selected. In the Firefox Settings dialog box, specify the Firefox browser profile to use for previewing output: <ul style="list-style-type: none">– Path to "profiles.ini": in this text box, specify the location of the <code>profiles.ini</code> file, which determines the Firefox profile to be used.– Profile: from this drop-down list, select the desired predefined profile to use. Learn more at Firefox browser profile. In the Chrome Settings dialog box: <ul style="list-style-type: none">– Command line options: In this text box, enter the command line options to launch an instance of Chrome. If you need more space, click  or press <code>Shift+Enter</code> to open the editor box. Learn more about Chrome command line options by opening <code>chrome://flags</code> in Chrome.– Use custom profile directory: Select this check box to define a user-specific Chrome profile to use and specify the location of the <code>chrome-user-data</code> directory, which determines the Chrome profile to be used. Learn more about Chrome profiles at Multi-profiles.
	Use these buttons to move the selected browser up or down in the list. The order of browsers is important for rendering external resources and previewing files with Web contents.
	Click this button to create a copy of the selected browser.

Default Browser

In this section, specify the **default PyCharm browser** that will be used by default for rendering external resources and previewing files with Web contents. This browser will be referred to as Default in the context menu when you choose View | Open in Browser on the main menu or Open in Browser on the context menu of a file.

ItemDescription

Default browser	Select the default browser from the drop-down list. The possible options are: <ul style="list-style-type: none">– System default: Select this option to accept your operating system default Web browser as default for PyCharm.– First listed: Select this option to have PyCharm launch the first browser in the list. Change the order of browsers using the  and  icons on the toolbar.– Custom path: Select this option to specify another Web browser as default for PyCharm. Type the path to the executable file of the browser or click  and select the path in the dialog box that opens.
-----------------	--

Show browser popup – If this check box is selected, the popup window with the enabled browsers appears in the HTML files.

in the editor

– If this check box is not selected, this popup window does not show up, thus helping you read or edit code.

File Watchers

File | Settings | Tools | File Watchers for Windows and Linux

for macOS

Ctrl+Alt+S



The page is available when the **File Watchers** plugin is enabled. The plugin is bundled with PyCharm and activated by default. If the plugin is not activated, enable it on the [Plugins](#) page of the [Settings / Preferences Dialog](#) as described in [Enabling and Disabling Plugins](#).

Use this page to create project **File Watchers** based on predefined PyCharm **File Watcher** templates and thus enable **compilation** in the project.

PyCharm supports integration with various third-party [compilers](#) that run in the background and perform the following:

- Translate **Less**, **Sass**, **SCSS**, and **Stylus** source code into **CSS** code.
- Translate **CoffeeScript** source code into **JavaScript** code, possibly also creating **source maps** to enable debugging.
- Compress **JavaScript** and **CSS** code.

Note that PyCharm does not contain built-in **compilers** but only supports integration with the tools that you have to download and install outside PyCharm.

In PyCharm, these compiler configurations are called **File Watchers**. For each supported compiler, PyCharm provides a predefined **File Watcher** template. Predefined **File Watcher** templates are available at the PyCharm level. To run a compiler against your project files you need to create a project-specific **File Watcher** based on the relevant template, at least, specify the path to the compiler to use on your machine.

You can download a **compiler** of your choice and set it up as a **File Watcher**. However, in this case no predefined template is available so you will have to specify all the settings manually.

To be applicable, a **File Watcher** must be enabled by selecting the check box next to it on the File Watchers page of the Settings dialog box, see [Enabling and Disabling File Watchers](#). After that the **File Watcher** will be invoked automatically upon any changes made to the source code or upon save, depending on whether the Immediate File Synchronization check box is selected or cleared, see [New Watcher Dialog](#).

The output of a **File Watcher** is stored in a separate file. The predefined templates suggest the type of the file depending on the compiler type. By default the output file is created in the same folder as the input file when the **File Watcher** is invoked for the first time, whereupon this file is only updated. You can customize all these settings during **File Watcher** creation.

JavaScript files generated by **File Watchers** are excluded from code completion and refactoring.

In the Project tree view, the output file is shown under the original file which is shown as a node. This is done to improve visibility so you can easier locate necessary files.

File watchers have two dedicated **code inspections**:

- The **File watcher available** inspection is invoked in every file that is recognized as subject for applying a predefined file watcher (Sass, Less, SCSS, or CoffeeScript). If none of the applicable predefined **File Watchers** is associated with the current project, PyCharm suggests to add one.
- The **File watcher problems** inspection is invoked by a running **File Watcher** and highlights errors specific for it.

The File Watchers page consists of two parts:

- A list of **File Watchers** available in the current project. To activate a **File Watcher**, select the check box next to it. If an error occurs while a **File Watcher** is running, the **File Watcher** is automatically disabled.
- A toolbar to manage this list.

ItemTooltip/ Description

and shortcut		
	Add	Click this button to open the Choose template pop-up list and choose the relevant type of File Watcher . After that PyCharm opens the New Watcher dialog box for customizing the predefined File Watcher according to the settings of the current project.
	Edit	Click this button to update the settings of the selected File Watcher in the Edit Watcher dialog box. The update is applied to the current project File Watcher only, it does not affect the predefined PyCharm-level template.
	Remove	Click this button to remove the selected File Watcher . The File Watcher is no longer applied to the files in the current project. Note that this action does not affect the corresponding predefined template which is still available at the PyCharm level.
	Up(Ctrl+Alt+Up) Down (Ctrl+Alt+Down)	Use these buttons to change the order of File Watcher in the list. This determines the order of launching File Watchers , if more than one are enabled.
	Copy	Use this button to create a copy of the selected file watcher.
	Import	Click this button to import an existing file watcher and add it to the list of available file watchers.



Export

Click this button to export the selected watchers to `watchers.xml` file, located under the user's home.

Ctrl+Alt+S



The dialog box opens when you click the Add  or Edit  button on the [File Watchers page](#). Use the dialog box to create a project **File Watcher** based on a predefined PyCharm **File Watcher** template or to edit an existing project **File Watcher**.

Each template contains the settings that are optimal for the selected compiler. So in most cases, all you need is specify the path to the **compiler** executable.

On this page:

- [Name area](#)
- [Watched Files area](#)
- [Watcher Settings Area](#)
- [Other Options area](#)
- [Options area](#)
- [Examples of customizing the behaviour of a compiler](#)

Name area

ItemDescription

Name In this text box, type the name of the **File Watcher**. By default, PyCharm suggests the name of the selected predefined template.

Watched Files area

In this area, specify the type and location of files to be processed by the **File Watcher**.

ItemDescription

File type Use this drop-down list to specify the expected type of input files. The **File Watcher** will consider only files of this type as subject for analyzing and processing. File types are recognised based on [associations between file types and file extensions](#). By default, the field shows the file type in accordance with the chosen predefined **File Watcher** template.

Scope Use this drop-down list to define the range of files the **File Watcher** can be applied to. Changes in these files will invoke the **File Watcher** either immediately or upon save or frame deactivation, depending on the status of the Immediate file synchronization check box. You can choose one of the predefined scopes from the drop-down list:

- Project Files: all the files within the project content roots (see [Content Root](#) and [Configuring Content Roots](#)).
- Project Production Files: all the files within the project content roots excluding test sources.
- Project Test Files: all the files within the project test source roots.
- Open Files: all the files that are currently opened in the editor.

VCS Scopes: these scopes are only available if your [project is under version control](#).

- Changed Files: all changed files, that is, all files associated with all existing changelists.
- Default: all the files associated with the changelist `Default`.

Alternatively, click the Browse button and configure a custom scope in the **Scopes** dialog box that opens.

For more details on scopes, see the pages [Scopes](#) and [Scopes dialog](#).

The Scope setting overrides the Track only root files check box setting: if a dependency is outside the specified scope, the **File Watcher** is not applied to it.

Watcher Settings Area

In this area, configure interaction with the **compiler**: specify the executable file to use, the arguments to pass to it, and customize the default template settings for input and output.

ItemDescription

Program In this text box, specify the location of the **compiler's** executable file (`.exe`, `.cmd`, `.bat`, or other depending on the specific tool. `.jar` archives are also acceptable but defining `PATH` variables for them is not supported.) Do one of the following:

- Type the path explicitly.
- Click the Browse button  to open the Select Path dialog box and navigate to the desired location.
- Click the Insert Macro button to open the [Macros](#) dialog box where you can select the relevant macro from the list.

Arguments In this text box, define the arguments to pass to the **compiler** and thus influence its behaviour. Among other cases, use this text box to change the default output location, that is, specify a custom location where you want the **compiler** to store the files generated during compilation. Note that if you re-define the default output location here you need to clear the Create output file from stdout check box in the Other Options area because otherwise the content of your generated file will be overwritten by the compiler's output stream.

Do one of the following:

- Type the list of arguments in the text box.
- Click the Insert Macro button to open the [Macros](#) dialog box where you can select the desired macro from the list.

When specifying the arguments, follow these rules:

- Use spaces to separate individual arguments.
- If an argument includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, `some "arg" or "some arg"`.
- If an argument includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, `Dmy.prop="\quoted_value\"`.

Output paths to refresh In this text box, specify the files where the **compiler** stores its output: the resulting source code, source maps, and dependencies. In other words, tell PyCharm where it should search for the files generated through compilation. Please note, that changing the value in this text box does not make the **compiler** store its output in another location. To do that, specify the desired output location in the Arguments text box.

Do one of the following:

- Type the output paths manually. Use colons as separators.
- Click the Insert Macro button to open the **Macros** dialog box, where you can select the desired pattern from the list.

Other Options area

ItemDescription

Working directory In this text box, specify the directory to which the **compiler** will be applied. Because the tool is always invoked in the context of a file, the default working directory is the directory of the current file. This setting is specified in all predefined templates through a macros `$FileDir$`. To update this default settings, do one of the following:

- Type the path explicitly in the text box.
- Click the Browse button  to open the Select Path dialog box and navigate to the desired location.
- Click the Insert Macro button to open the **Macros** dialog box, where you can select the desired macro from the list.

Tip If the field is left blank, PyCharm uses the directory of the file where the **File Watcher** is invoked.

Environment variables Use this text box to specify a the `PATH` variable for a tool that is required for starting the compiler but is not referenced in the path to it.

Create output file from stdout

- When this check box is selected, PyCharm reads the native **compiler's** output (`standard output stream (stdout)`) and generates the resulting files from it.
- When the check box is cleared, the **compiler** writes its output directly to the files specified in the Output paths to refresh field.

Tip Some **compilers** generate a `standard output stream (stdout)` file, others do not, which may lead to errors. Therefore it is strongly recommended to preserve the default setting.

Options area

In this area, configure the behaviour of the **File Watcher**.

ItemDescription

Output Filters Click this button to open the **Output Filters dialog** where you can manage the list of filters to distinguish the output of the **File Watcher** from other output. These filters make the basis for:

1. Displaying paths to the **File Watcher** output files as links in error and other messages and logs. When you click such link, the corresponding file is opened in the editor. For example, to get useful error messages displayed, specify the following expression in the Regular expression to match output field of the **Add/Edit Filter Dialog**:

```

$FILE_PATH$: $LINE$ $MESSAGE$

```

2. Error highlighting in the output files.

Immediate file synchronization

- Select this check box to have the **File Watcher** invoked as soon as any changes are made to the source code.
- If the check box is cleared, the **File Watcher** will start upon save (File | Save All, `Ctrl+S`) or when you move focus from PyCharm (upon frame deactivation).

Track only root files When the **File Watcher** is invoked on a file, PyCharm detects all the files in which this file is included. For each of these files, in its turn, PyCharm again detects the files into which it is included. This operation is repeated recursively until PyCharm reaches the files that are not included anywhere **within the specified scope**. These files are referred to as **root files** (do not confuse with **content roots**).

- When this check box is selected, the **File Watcher** runs only against the **root files**.
- When the check box is cleared, the **File Watcher** runs against the file from which it is invoked and against all the files in which this file is included recursively within the specified scope.

This option is available only for Babel, Closure Compiler, Compass, Jade, Less, Sass/SCSS, Stylus, UglifyJS, and YUI Compressor JS.

Show console Use this drop-down list to define when you want the **File Watcher** open its dedicated console with messages. The available options are:

- **Always:** when this option is chosen, the **File Watcher** opens the console when it starts.
- **Error:** when this option is chosen, the **File Watcher** opens the console only if any errors occur in compilation.
- **Never:** choose this option to suppress opening the console under any circumstances.

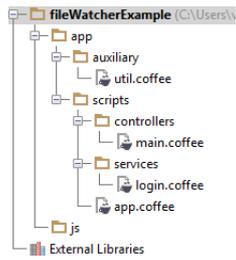
Check file for syntax errors

- Select this check box to have the **File Watcher** ignore update, save, and change focus events in files that are syntactically invalid.
- If this check box is cleared, the **File Watcher** starts anyway upon update, save, or frame deactivation, depending on the status of the Immediate file synchronization check box.

Examples of customizing the behaviour of a compiler

Any **compiler** is an external, third-party tool. Therefore the only way to influence a **compiler** is pass arguments to it just as if you were working in the command line mode. These arguments are specific for each tool. Below are two examples of customizing the default output location for the **CoffeeScript compiler**.

Suppose, you have a project with the following folder structure:



By default, the generated files will be stored in the folder where the original file is. You can change this default location and have the generated files stored in the `js` folder. Moreover, you can have them stored in a flat list or arranged in the folder structure that repeats the original structure under the `app` node.

– To have all the generated files stored in the output `js` folder without retaining the original folder structure under the `app` folder:

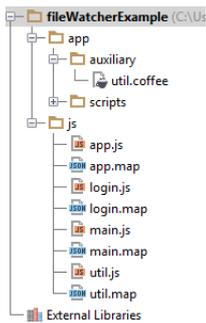
1. In the Arguments text box, type:

```
--output $ProjectFileDir\js\ --compile --map $FileName$
```

2. In the Output paths to refresh text box, type:

```
$ProjectFileDir\js\${FileNameWithoutExtension}.js:$ProjectFileDir\js\${FileNameWithoutExtension}.map
```

As a result, the project tree looks as follows:



– To have the original folder structure under the `app` node retained in the output `js` folder:

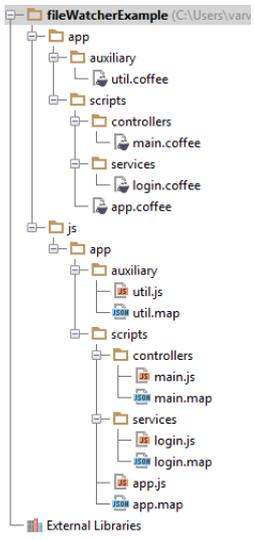
1. In the Arguments text box, type:

```
--output $ProjectFileDir\js\${FileDirRelativeToProjectRoot}\ --compile --map $FileName$
```

2. In the Output paths to refresh text box, type:

```
$ProjectFileDir\js\${FileDirRelativeToProjectRoot}\${FileNameWithoutExtension}.js:$ProjectFileDir\js\${FileDirRelativeToProjectRoot}
```

As a result, the project tree looks as follows:



External Tools

File | Settings | Tools | External Tools for Windows and Linux

PyCharm | Preferences | Tools | External Tools for macOS

Ctrl+Alt+S



Define third-party standalone applications (code generators and analyzers, pre- and post- processors, database utilities, etc.) as external tools to be able to run them from PyCharm.

You can pass contextual information (like the currently selected file, or your project source path) to the external tools, view the tool output, and more.

The tools defined on this page appear as commands in the Tools menu and in various context menus. They can also be assigned keyboard shortcuts (see [Configuring Keyboard Shortcuts](#)).

Toolbar icons

IconDescription

	Add a definition of an external tool. (The Create Tool dialog will open.)
	If an individual tool is selected: delete the tool definition. If a tool group is selected: delete the definitions of all the tools within the selected group.
	Edit the definition of the selected tool. (The Edit Tool dialog will open.)
	Move the selected tool one line up or down within the group. (The order of tools defines the order of items in corresponding menus.)
	Create a copy of the selected definition and then edit that copy. (The Copy Tool dialog will open.)

Check boxes

Use the check boxes to enable or disable the tools and the tool groups. The items that are not currently selected are not available in the Tools and context menus.

Specify the filter for transferring absolute file paths and line numbers in the tool output into hyperlinks.

Item

Description

Name	The name of the filter.
------	-------------------------

Description	The filter description (optional).
-------------	------------------------------------

Regular expression to match output	The text pattern to be matched against the tool output to identify linkable references. The pattern can include text and the following placeholder variables:
------------------------------------	---

- `$FILE_PATH$` - the portion of the output that corresponds to an absolute path to a source file. Required.
- `$LINE$` - a line number reference.
- `$COLUMN$` - a column reference.
- `$MESSAGE$` - a message to display in the log.

The variables are inserted by right-clicking the field and then selecting the necessary item from the list that is shown.

Example

If your tool outputs the lines similar to

```
Error parsing C:\Demos\src\converter\MetersToInches.xml:103 Missing Closing Tag
```

the pattern `$FILE_PATH$:LINE$` will turn `C:\Demos\src\converter\MetersToInches.xml:103` into a hyperlink to the line number `103` in the file `MetersToInches.xml`.

Output Filters Dialog

From the [Create/Edit/Copy Tool dialog](#): Output Filters

This dialog lets you manage the output filters associated with an external tool. (The output filters are used to turn absolute file paths and line numbers in the tool output into hyperlinks.)

ItemDescription

- | | |
|--|---|
|  | Create a new filter. (The Add Filter dialog will open.) |
|  | Delete the selected filter. |
|  | Edit the selected filter. (The Edit Filter dialog will open.) |
|  | Move the selected filter one line up or down in the list. (For each line in the output, the first matching filter is used.) |

Macros Dialog

From the [Create/Edit/Copy Tool dialog](#): Insert macro

Select the macro to be inserted.

ItemDescription

Macros The list of available macros with their descriptions.

Macro preview The value of the selected macro in the current context.

Note In case of multiple projects, opened in the same window, the macro `ModuleSdkPath` expands into an SDK that corresponds to the current project.

Edit the settings for your external tool.

ItemDescription

Name	The name of the tool that appears as a command name in the Tools menu and the context menus. See also, Show in .
Group	The group the tool belongs to. The tool groups correspond to submenus in the Tools menu and the context menus. Select an existing group from the list or type the name for a new group.
Description	The tool description (optional).
Options	
Synchronize files after execution	Make PyCharm aware of changes in the file system when the tool completes its execution.
Open console	Open the console for viewing the tool output such error messages, etc.
Output Filters	Open the Output Filters dialog to manage the output filters associated with the tool. (The output filters are used to turn absolute file paths and line numbers in the tool output into hyperlinks. You'll be able to use those links to open the corresponding files in the editor.)
Show console when a message is printed to standard output stream	Make the output console active and bring it forward when the corresponding event occurs.
Show console when a message is printed to standard error stream	The same as the previous option but for stderr.
Show in	Specify in which menus the command for running the tool should be included. Main menu means the Tools menu. The rest of the options correspond to context menus in various places.
Tool settings	
Program	The path to the executable file to be run. Use  to select the file, Insert macro to open the Macros dialog to select a macro. (Macros are resolved at runtime and let you specify context information such as currently selected file, your project source paths, etc.)
Parameters	The parameters to be passed to the program the way you'd specify them on the command line. Use Insert macro to open the Macros dialog to select a macro. When specifying the parameters, follow these rules: <ul style="list-style-type: none"> - Use spaces to separate individual parameters. - If a parameter includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, <code>some "arg" or "some arg"</code>. - If a parameter includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, <code>-Dmy.prop=\"quoted_value\"</code>.
Working directory	The path to the current working directory for the program. Use  to select the directory, Insert macro to open the Macros dialog to select a macro.

Tip If the specified external tool is a Python script, add `-u` to the Python options. Thus you will be able to get prompt and user input before the script ends.

Terminal

File | Settings | Tools | Terminal for Windows and Linux

PyCharm | Preferences | Tools | Terminal for macOS

Ctrl+Alt+S



Use this dialog to specify settings for the embedded local terminal.

Prerequisites

Before you start working with terminal, make sure that the Terminal plugin is enabled. The plugin is bundled with PyCharm and is activated by default. If the plugin is not activated, enable it on the [Plugins](#) page of the [Settings / Preferences Dialog](#) as described in [Enabling and Disabling Plugins](#).

ItemDescription

Shell path	Specify the shell that will run by default.
Tab name	Specify the default name of a new session tab. Note that a session tab can be renamed.
Close session when it ends	If this check box is selected, the current session ends automatically, when the corresponding process ends (for example, by kill).
Audible bell	If this check box is selected, the console plays the bell sound on incoming escape sequence.
Mouse reporting	If this check box is selected, the embedded local terminal supports mouse pointer.
Copy to clipboard on selection	If this check box is selected, the text selected in the Terminal, is automatically passed to the system clipboard.
Paste on middle mouse button click	If this check box is selected, you can paste the clipboard contents by clicking the middle mouse button.
Override IDE shortcuts	If this check box is selected (this is the default setting), the Terminal tool window handles keyboard shortcuts differently from PyCharm. If this check box is cleared, the PyCharm key bindings are used.
Shell integration	When this check box is selected, the terminal first loads a custom <code>rc</code> config file (located in the <code>terminal</code> folder under <code>plugins</code> of PyCharm distribution) which provides an additional set-up, and then the user's <code>rc</code> file. Note Note that presently shell integration works for Bash/sh (<code>bashrc</code>), zsh(<code>zshrc</code>) and fish shell (<code>config.fish</code>).
Activate virtualenv	For the project interpreter being a virtual environment, with this check box selected, the virtual environment is automatically activated (<code>activate</code> is performed automatically).

Database

File | Settings | Tools | Database for Windows and Linux

PyCharm | Preferences | Tools | Database for macOS

From the [database console](#): 

From the [table editor](#):  | Settings

The pages in the Database section contain the settings related to working with databases and SQL.

- Database
 - [Console](#)
 - [Execute in Console](#)
 - [Quick Documentation](#)
 - [DDL editor](#)
- [Data Views](#)
- [User Parameters](#)
- [CSV Formats](#)

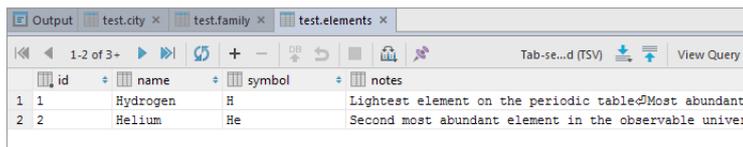
Console

The settings in this section relate to showing various information in [database consoles](#).

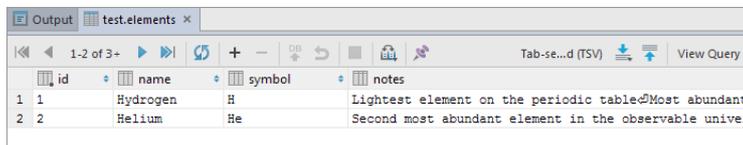
ItemDescription

Show query results in new tab

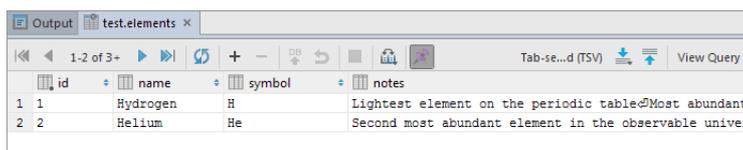
You can select to view query results on individual tabs, or on one and the same tab. If the check box is selected, a new tab with the query result will open each time you run a query (`SELECT`). In this way, you can keep the results of all the queries that you have run.



If this check box is not selected, the same tab is used to show your query results. When you run a query, the information on the tab is updated to show the result, and a new tab doesn't open.

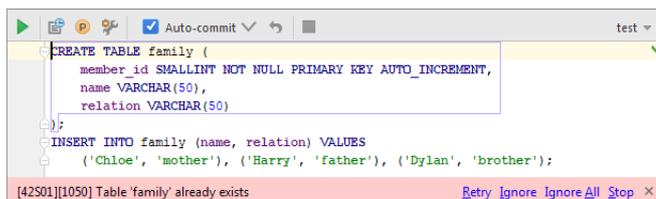


In this case, when you get the result that you want to keep, you can pin the tab to the tool window ( on the toolbar or Pin Tab in the context menu for the tab).



Show error notifications in editor

If this check box is not selected, the information about the errors is shown only in the output pane. If, in addition, you want the error notification bar to appear in the input pane, select the check box. The error notification bar may be particularly useful when running sequences of SQL statements. If an error occurs in such cases, the error notification bar lets you select how to react.



For more information, see [Using the error notification bar](#).

Always review parameters before execution

When you run a statement with parameters, PyCharm memorizes the parameter values. Each next time you execute the statement:

- If this option is on, PyCharm shows you the last used parameter values so that you can change them before actually running the statement.
- If this option is off, PyCharm executes the statement right away without showing you the parameter values.

Track creation and deletion of databases/schemas

When you create a new schema or database, or delete a schema or database (see e.g. [Creating a database or schema](#)):

- If this option is on, the new schema or database is shown in the Database tool window right away. Deleted schemas and databases are immediately removed from the Schemas popup in the Database tool window and from the list on the Schemas tab in the Data Sources and Drivers dialog.

- If this option is off, the new schema or database isn't shown unless you visualize it manually, see [Showing and hiding schemas](#). Deleted schemas and databases that were selected for viewing will stay in the corresponding lists unless deselected.

Execute in Console

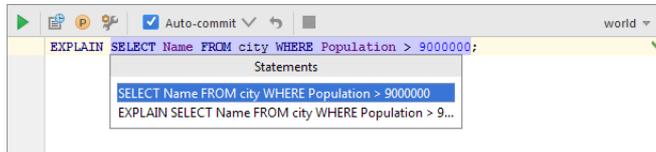
This section contains the options for the Execute command.

ItemDescription

When inside a statement execute

If the cursor is inside a statement, the following options are available:

- Ask what to execute. A list of statements that can be run is shown and you can select the statement or statements.



- Smallest statement. The smallest of the possible statements is executed. For example, when the cursor is inside a subquery, the subquery is executed.
- Largest statement. The largest possible statement is executed. For example, when the cursor is inside a subquery, an outer statement is executed.
- Largest statement or batch. For Transact-SQL (SQL Server and Sybase), the current batch of statements is executed. For all other dialects - the same as the previous option.
- Whole script. All the statements are executed.

otherwise execute

If the cursor is outside of a statement, e.g. on a blank line or within a comment, the following options are available:

- Nothing. None of the statements is executed.
- Whole script. All the statements are executed.
- Everything below caret. All the statements after the cursor position are executed.

for selection execute

If something is currently selected, the following options are available:

- Exactly as one statement. Exactly what is selected is executed as a single statement.
- Exactly as statements. Exactly what is selected is executed. If the selection contains more than one statement, the statements are executed as separate statements.
- Smart expand to script. If there is at least one statement border within the selection, the selection is expanded to form a sequence of valid statements. This sequence is then executed. Otherwise, precisely what is selected is executed.

Quick Documentation

This section contains the settings for the quick documentation view, see [Viewing basic info about an item](#).

ItemDescription

Show first rows When showing quick documentation for a table, include data for a number of first rows.

Number of preview rows The number of rows to be shown for a table in the quick documentation view.

DDL editor

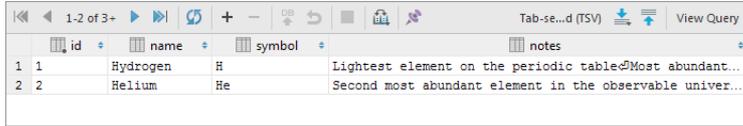
ItemDescription

Show confirmation on close When trying to close the Add Database, the Add Schema or the [Create / Modify Table dialog](#) by clicking Cancel or pressing :
 - If this option is on, you are asked for a confirmation.
 - Otherwise, the dialog closes right away.

The settings on this page define how table data are shown and modified in the [database console](#) and the [table editor](#).

ItemDescription

Result set page size The number of table rows to be shown at a time, on one "page". Here is an example when this number is set to 2:



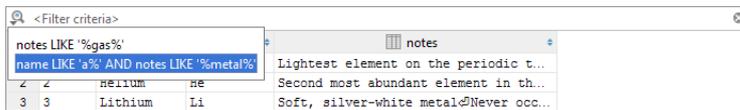
The screenshot shows a database table with columns: id, name, symbol, and notes. The table contains two rows of data:

	id	name	symbol	notes
1	1	Hydrogen	H	Lightest element on the periodic table. Most abundant...
2	2	Helium	He	Second most abundant element in the observable univer...

If you don't want to limit the number of rows displayed simultaneously, specify zero (0).

Result set prefetch size Data from databases are retrieved in chunks. The number in this field defines the number of rows in such chunks. A bigger number means fewer IDE - DB round trips but more memory for storing a chunk.

Filter history size The number of most recently used filtering conditions to memorize for a table in the table editor. Here is an example when this number is set to 2. (The filter history box contains two most recently used conditions.)



The screenshot shows a filter history box with two filter conditions:

- notes LIKE '%gas%'
- name LIKE 'a%' AND notes LIKE '%metal%'

Below the filter history box, a table view is shown with columns: name, symbol, and notes. The table contains three rows of data:

	name	symbol	notes
2	Helium	He	Lightest element on the periodic t...
3	Lithium	Li	Second most abundant element in th...
3	Lithium	Li	Soft, silver-white metal. Never occ...

Max LOB length (bytes) The maximum size of a binary large object to be loaded in bytes.

Data Modification / Submit changes immediately When this options is off, the changes you make to data in a table are accumulated in PyCharm unless you carry out the Submit command (⌘) on the toolbar, Submit in the context menu or (Ctrl+Enter). Before you submit the changes, you can revert them (⌘) on the toolbar, Revert in the context menu or (Ctrl+Z).

When this option is on, the changes are submitted right away.

See [Submitting and reverting changes](#).

Generally, only the question mark (?) is treated as a parameter in SQL statements. On this page, you can specify which other characters and their sequences should be treated as parameters, and in which places.

The patterns for SQL parameters are specified by means of regular expressions.

ItemDescription

Enable in console and SQL files If the check box is selected, the parameter patterns are applied to SQL (in SQL files and database consoles). The usage scope, if necessary, may be limited at the level of [individual patterns](#).
If this check box is not selected, the patterns are not used in SQL files and consoles irrespective of which usage scope is specified for individual patterns.

Enable in string literals with SQL injection If the check box is selected, the parameter patterns are applied to string literals injected with SQL. (See [Using Language Injections](#).) The usage scope, if necessary, may be limited at the level of [individual patterns](#).
If this check box is not selected, the patterns are not used in string literals irrespective of which usage scope is specified for individual patterns.

Parameter patterns The table shows the parameter patterns and their usage scopes. The patterns are specified using regular expressions. Values in parentheses are treated as parameter names. The patterns available initially have the following meanings:

- `\?(\d+)` - a question mark followed by one or more digits, e.g. `?69` in which case `69` would be the parameter name.
- `:(\w+)` - a colon followed by one or more word characters, e.g. `:x`, `:value`, `:parameter_1`.
- `%w+` - `%` followed by one or more word characters, e.g. `%xyz`.
- `\$\{([^\$\\\}]*)\}` - `$`, then `{`, then any character except `$`, `{` or `}` zero or more times, then `}`, e.g. `${}`, `${value}`.
- `\$\(((^\\)+)\)` - `$`, then `(`, then any character except `)` one or more times, then `)`, e.g. `$(x)`.
- `\$(\w+)\$` - `$`, then one or more word characters, then `$` again, e.g. `$x1$`.
- `\#(\w+)\#` - `#`, then one or more word characters, then `#` again, e.g. `#field_3#`.

Use **+** (`Alt+Insert`), **-** (`Alt+Delete`), **↑** (`Alt+Up`) and **↓** (`Alt+Down`) to add, delete and reorder the patterns.

To edit a pattern or its usage scope, click the pattern and use the following controls:

- In scripts. Clear this check box if the pattern shouldn't be used in SQL files and database consoles.
- In literals. Clear this check box if the pattern shouldn't be used in string literals injected with SQL.
- All (the link text may be different). Click the link and deselect the languages in which the pattern shouldn't be used.

This page contains the settings for converting table data into delimiter-separated values formats (e.g. CSV, TSV) and vice versa.

When working on the conversion settings, use the preview in the right-hand part of the page.

ItemDescription

Formats	The list of the available delimiter-separated values formats is shown. Each format is a named set of corresponding conversion settings. Select the format whose settings you want to view or edit. Use  ,  and  to create, delete and reorder the formats;  to create a copy of the selected format.
Value separator	Select or type the character for separating individual values.
Row separator	Select or type the character for separating rows.
Null value text	The text to be used as a value if a cell contains <code>null</code> (an unknown value).
Add row prefix/suffix	Row prefix and suffix are character sequences which in addition to the row separator indicate the beginning and end of a row. If necessary, click the link and specify the row prefix and suffix in the fields that appear.
Quotation	Each line in the area under Quotation is a quotation pattern (see Quote values). A quotation pattern includes: <ul style="list-style-type: none"> – The left quotation character, the one inserted before a value. – The right quotation character, the one inserted after a value; usually, the same as the left quotation character. – An escape method or character for the cases when the quotation character is part of a value. E.g. Escape: duplicate means that if a quotation character occurs within a value, it is doubled. (You can specify your own escape character instead.) <p>If there is more than one pattern, the first of the patterns is used.</p> <p>Use ,  and  to create, delete and reorder the patterns.</p> <p>To start editing an existing pattern, just click the pattern of interest.</p>
Quote values	Specify in which cases the values should be quoted (i.e. enclosed within quotation characters). <ul style="list-style-type: none"> – When needed. A value is quoted only if it contains the value and/or the row separator. – Always. Any value is quoted in its text representation.
Trim whitespaces	If this check box is not selected, the Unicode whitespace characters that precede and follow the value separators are treated as parts of the corresponding values. If this check box is selected, the corresponding whitespace characters are ignored or removed.
First row is header	If this check box is selected, the first row is treated as containing column names. The settings that appear under Header Format have the same meanings as the ones above but are applied to the first row.
First column is header	If this check box is selected, the first column is treated as containing row names.

SSH Terminal

File | Settings | Tools | SSH Terminal for Windows and Linux

PyCharm | Preferences | Tools | SSH Terminal for macOS

Ctrl+Alt+S



Use this dialog box to appoint the a remote Web server or the default remote interpreter to access through the SSH terminal, configure connection with the destination environment, and choose the encoding to use in the SSH terminal.

Make sure the **SSH Remote Run** plugin is enabled. The plugin is bundled with PyCharm and activated by default. If the plugin is not activated, enable it on the [Plugins](#) page of the [Settings / Preferences Dialog](#) as described in [Enabling and Disabling Plugins](#).

ItemDescription

Connection settings	<p>In this section, appoint the a remote Web server or the default remote interpreter to access through the SSH terminal and specify where the connection settings should be taken from:</p> <ol style="list-style-type: none">1. Default remote interpreter: select this option to have commands in the SSH terminal executed on the same host, where the default remote interpreter runs.2. Current Vagrant: select this option to have the commands in the SSH Terminal executed on the currently running Vagrant virtual machine. For details, see Vagrant: Working with Reproducible Development Environments.3. Deployment server: select this option to have the commands in the SSH Terminal executed on the local or remote Web server accessible through one of the server access configurations. From the drop-down list, choose the server access configuration that specifies the destination environment and the settings to establish connection to it.<ul style="list-style-type: none">– Select server on every run: if this option is selected, you will have to choose the desired server access configuration from the pop-up window, every time you choose Tools Start SSH Session on the main menu.– If the desired server access configuration does not appear in the drop-down list, click the link Configure Servers, and define one in the Deployment page. For details, see Configuring Synchronization with a Web Server.
Default encoding	<p>From this drop-down list, select the desired encoding to be used in the SSH terminal.</p>

Diagrams

File | Settings | Tools | Diagrams for Windows and Linux

PyCharm | Preferences | Tools | Diagrams for macOS

Ctrl+Alt+S



Use this page to configure the default visibility settings and layout for diagrams.

On this page:

- [Content pane](#)
- [Controls](#)

Content pane

Select the check boxes next to the elements to be shown on diagrams.

ItemDescription

Show Difference

Details If this check box is selected, all the specified details of the elements will be shown in the UML class diagram for a revision. If this check box is not selected, only node elements will be included in diagram.

DB Diagrams

Key columns For the primary key columns to be shown when a diagram opens, select this check box.
When viewing a diagram in the editor, use on the toolbar to show or hide the corresponding columns.

Columns For the columns other than the primary key columns to be shown when a diagram opens, select this check box.
When viewing a diagram in the editor, use on the toolbar to show or hide the corresponding columns.

Python Class Diagram

Class Select the check boxes below to show members within the node elements.
Elements In diagram, use the toolbar buttons for methods, for fields, and for the inner classes.

Django Model Dependency Diagram

Element Select the check boxes below to show members within the node elements.
In diagram, use the toolbar button for fields.

Google App Engine Model Dependency Diagram

Fields Select the check boxes below to show fields within the node elements.
In diagram, use toolbar button .

SQLAlchemy Model Dependency Diagram

Fields Select the check boxes below to show fields within the node elements.
In diagram, use toolbar button .

Note More nodes appear in this pane depending on the installed and enabled plugins.

Controls

ItemDescription

Default layout Select the desired layout from the drop-down list. Node elements in newly created diagrams will be arranged according to the selected layout.

Default scope Select scope from the drop-down list. Specifying a scope helps you avoid showing in diagram the unnecessary hierarchies. You can define scopes for your project in the [Scopes page](#) of the Settings dialog.

Fit content after layout If this check box is selected, then after applying a layout selected on the diagram context menu, all diagram elements will be resized to fit into the current diagram area. In diagram, use the toolbar button.

Do relayout when new elements were added If this check box is selected, diagram layout will be performed automatically after adding new elements.

Enable colors If this check box is selected, relationship links will be shown colored.

Diff & Merge

File | Settings | Tools | Diff & Merge for Windows and Linux

PyCharm | Preferences | Tools | Diff & Merge for macOS

Ctrl+Alt+S



On this page, specify the default behavior of the [Differences viewer](#).

ItemDescription

Diff

Context lines Use the slider to specify the amount of context lines that will not collapse, when in a Differences viewer you click the button  to [collapse the unchanged fragments](#).

Go to the next file after reaching last change If this check box is selected, PyCharm will suggest to click  /  once more and compare other files

Merge

Automatically apply non-conflicting changes If this check box is selected, the interactive merge tool automatically merges all non-conflicting changes. This is an equivalent to clicking  in the [Merge](#) dialog.

Highlight modified lines in gutter Select this check box if you want added/modified lines (relative to the base revision) to be highlighted in the gutter of the Merge dialog.

Ctrl+Alt+S



If necessary, specify external tools for comparing files and folders, and associated settings.

ItemDescription

Use external diff tool	Select this check box to have PyCharm use an external tool for comparing files or folders.
Path to executable	In this text box, specify the path to the executable file of the desired external diff tool. Use the Browse button  , if necessary. This field is only available, when Use external diff tool check box is selected.
Use by default	Select this check box to have PyCharm use the specified external tool for comparing files or folders by default. If this check box is not selected, PyCharm will use the built-in diff tool. To invoke an external tool, click the button  on the toolbar of the Differences viewer . This field is only available, when Use external diff tool check box is selected.
Parameters	Use this field to set the diff tool parameters. Note Note that different diff tools have different parameters. You need to specify all the necessary parameters in proper order. This field is only available, when Use external diff tool check box is selected.
Use external merge tool	Select this check box to have PyCharm use an external merge tool. In the text box below, specify the path to the executable file of the desired external tool. Use the Browse button  , if necessary.
Path to executable	In this text box, specify the path to the executable file of the desired external merge tool. Use the Browse button  , if necessary. This field is only available, when Use external merge tool check box is selected.
Parameters	Use this field to set the merge tool parameters. Note Note that different merge tools have different parameters. You need to specify all the necessary parameters in proper order. This field is only available, when Use external merge tool check box is selected.

Docker Machine

File | Settings | Tools | Docker Machine for Windows and Linux

PyCharm | Preferences | Tools | Docker Machine for macOS

This page appears in the Settings/Preferences dialog, when the Docker integration plugin is enabled.

The plugin is bundled with PyCharm and is activated by default. If it is disabled, you can manually [enable the plugin](#).

Specify the settings for working with [Docker Machine](#) or [Docker Toolbox](#).

ItemDescription

Docker Machine executable `docker-machine` or an actual path to `docker-machine.exe` (normally located in the Docker Toolbox installation folder).
The default setting `docker-machine` is fine if:

- The actual name of the executable file is `docker-machine`.
- The path to the directory where the file is located is included in the environment variable `Path`.

Detect Set Docker Machine executable to `docker-machine`, try to detect the executable file and, if a success, show its version.

Python External Documentation

Warning! This page only appears when Python Plugin is installed and enabled!

File | Settings | Tools | Python External Documentation for Windows and Linux

PyCharm | Preferences | Tools | Python External Documentation for macOS

Ctrl+Alt+S



In this section:

- [Python External Documentation](#)

- [Add/Edit Documentation URL](#)

Python External Documentation

ItemDescription

Module Names	This column shows the names of the modules, whose documentation you want to have visible in browser on invoking View External Documentation, or pressing Shift+F1 .
--------------	--

URL Pattern	This column shows existing patterns of the URLs to the external documentation, or its local address. If external documentation resides locally, specify the local path to it .
-------------	--

	Click this button to add to the list a new module and its URL pattern or local address.
--	---

	Click this button to change the name and/or URL pattern of the selected module.
--	---

Double-clicking an entry in the table produces same result.

	Delete the selected module from the list.
--	---

Add/Edit Documentation URL

ItemDescription

Module name	Type module name in the text field.
-------------	-------------------------------------

URL pattern	In this text field, create the desired pattern, using plain text and macros from the Available Macros field. Note that documentation can also reside locally.
-------------	---

Insert	Click this button to add the selected macro to the pattern.
--------	---

Python Integrated Tools

Warning! This page only appears when Python Plugin is installed and enabled!

File | Settings | Tools | Python Integrated Tools for Windows and Linux

PyCharm | Preferences | Tools | Python Integrated Tools for macOS

Ctrl+Alt+S



Use this page to configure requirements management file, default test runner, and documentation strings treatment.

ItemDescription

Package requirements file Type the name of the [requirements file](#), or click the browse button, and select the desired requirements file from file system using the [Select Path](#) dialog.

Default test runner Select the test run/debug configuration that PyCharm will suggest every time you choose Run on the context menu of a test case. Refer to the section [Testing Frameworks](#) for details.

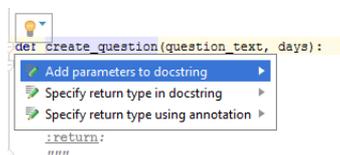
Docstring format Select the format of the documentation strings to be recognized by PyCharm. Depending on the selected docstring format, PyCharm will generate the stub documentation comments and render text in the [Quick Documentation](#) lookup:

- **Plain**: on pressing [Enter](#) or Space after opening quotes, an empty stub is generated; quick documentation shows as plain text.
- **reStructuredText**: on pressing [Enter](#) or Space after opening quotes, stub doc comment is generated according to [reStructuredText](#) format; the quick documentation is rendered by Docutils.
- **Epytext**: on pressing [Enter](#) or Space after opening quotes, stub doc comment is generated according to the [epytext](#) format; quick documentation is rendered by [epydoc](#).
- **NumPy**: on pressing [Enter](#) or Space after opening quotes, stub doc comment is generated according to the [NumPy](#) format; the quick documentation is rendered by [Napoleon](#) and Docutils.
- **Google**: on pressing [Enter](#) or Space after opening quotes, stub doc comment is generated according to [Google](#) format; the quick documentation is rendered by [Napoleon](#) and Docutils.

All types of docstrings feature:

- Proper generation of docstrings
- Updates after applying intention actions and quick-fixes
- Coding assistance
- Autocompletion for section headers

Note that the information provided in the docstrings, is used for code insight.



Analyze Python code in docstrings If this check box is selected, PyCharm highlights the code examples and performs syntax checks and code inspections.

If this check box is not selected, the code fragments inside docstrings are not analyzed.

Sphinx working directory Specify here the path to the directory that contains `*.rst` files.

Note For recognizing custom roles, point to the directory with `conf.py`.

Treat *.txt files as reStructuredText If this check box is selected, the files with `*.txt` extension will be highlighted same way, as the files with `*.rst` extension.

Remote SSH External Tools

File | Settings | Tools | Remote SSH External Tools for Windows and Linux

PyCharm | Preferences | Tools | Remote SSH External Tools for macOS

Define remote applications that require SSH access as external tools to be able to run them from PyCharm.

You can pass contextual information (like the currently selected file, or your project source path) to the external tools, view the tool output, and more.

The tools defined on this page appear as commands in the Tools menu and in various context menus. They can also be assigned keyboard shortcuts (see [Configuring Keyboard Shortcuts](#)).

Toolbar icons

Icon	Description
	Add a definition of an external tool. (The Create Tool dialog will open.)
	If an individual tool is selected: delete the tool definition. If a tool group is selected: delete the definitions of all the tools within the selected group.
	Edit the definition of the selected tool. (The Edit Tool dialog will open.)
	Move the selected tool one line up or down within the group. (The order of tools defines the order of items in corresponding menus.)
	Create a copy of the selected definition and then edit that copy. (The Copy Tool dialog will open.)

Check boxes

Use the check boxes to enable or disable the tools and the tool groups. The items that are not currently selected are not available in the Tools and context menus.

Edit the settings for your external tool.

ItemDescription

Name	The name of the tool that appears as a command name in the Tools menu and the context menus. See also, Show in .
Group	The group the tool belongs to. The tool groups correspond to submenus in the Tools menu and the context menus. Select an existing group from the list or type the name for a new group.
Description	The tool description (optional).
Options	
Synchronize files after execution	Make PyCharm aware of changes in the file system when the tool completes its execution.
Open console	Open the console for viewing the tool output such error messages, etc.
Output Filters	Open the Output Filters dialog to manage the output filters associated with the tool. (The output filters are used to turn absolute file paths and line numbers in the tool output into hyperlinks. You'll be able to use those links to open the corresponding files in the editor.)
Show console when a message is printed to standard output stream	Make the output console active and bring it forward when the corresponding event occurs.
Show console when a message is printed to standard error stream	The same as the previous option but for stderr.
Show in	Specify in which menus the command for running the tool should be included. Main menu means the Tools menu. The rest of the options correspond to context menus in various places.
Connection settings	In this section, appoint the to access through the SSH terminal and specify where the connection settings should be taken from: <ol style="list-style-type: none"> Default remote interpreter: select this option to have commands in the SSH terminal executed on the same host, where the default remote interpreter runs. Current Vagrant: select this option to have the commands in the SSH Terminal executed on the currently running Vagrant virtual machine. For details, see Vagrant: Working with Reproducible Development Environments. Deployment server: select this option to have the commands in the SSH Terminal executed on the local or remote Web server accessible through one of the server access configurations. From the drop-down list, choose the server access configuration that specifies the destination environment and the settings to establish connection to it. <ul style="list-style-type: none"> Select server on every run: if this option is selected, you will have to choose the desired server access configuration from the pop-up window, every time you choose Tools Start SSH Session on the main menu. If the desired server access configuration does not appear in the drop-down list, click the link Configure Servers, and define one in the Deployment page. For details, see Configuring Synchronization with a Web Server.
Tool settings	
Program	The path to the executable file to be run. Use 📄 to select the file, Insert macro to open the Macros dialog to select a macro. (Macros are resolved at runtime and let you specify context information such as currently selected file, your project source paths, etc.)
Parameters	The parameters to be passed to the program the way you'd specify them on the command line. Use Insert macro to open the Macros dialog to select a macro. When specifying the parameters, follow these rules: <ul style="list-style-type: none"> Use spaces to separate individual parameters. If a parameter includes spaces, enclose the spaces or the argument that contains the spaces in double quotes, for example, <code>some "arg"</code> or <code>"some arg"</code>. If a parameter includes double quotes (e.g. as part of the argument), escape the double quotes by means of the backslashes, for example, <code>Dmy.prop="quoted_value"</code>.
Working directory	The path to the current working directory for the program. Use 📄 to select the directory, Insert macro to open the Macros dialog to select a macro.

Tip If the specified external tool is a Python script, add `-u` to the Python options. Thus you will be able to get prompt and user input before the script ends.

Server Certificates

On this page:

- [Access](#)
- [Overview](#)
- [Table of controls](#)

Access

File | Settings | Tools | Server Certificates for Windows and Linux

PyCharm | Preferences | Tools | Server Certificates for macOS

Ctrl+Alt+S



Overview

PyCharm provides its own storage for trusted certificates. Use this page to manage this storage.

Table of controls

ItemShortcutDescription

Accept non-trusted certificates automatically		Select this option if you want non-trusted certificates (i.e. the certificates that are not added to the list) to be accepted automatically, without sending a request to the server.
---	--	---

+	Alt+Insert	Add a trusted server certificate to the list. Select the certificate file in the dialog that opens . The certificate file should have an extension <code>.crt</code> , <code>.cer</code> or <code>.pem</code> .
---	------------	---

For a trusted certificate, the certificate information is shown in the lower part of the page.

-	Alt+Delete	Remove the selected trusted certificate from the list.
---	------------	--

Settings Repository

File | Settings | Tools | Settings Repository for Windows and Linux

PyCharm | Preferences | Tools | Settings Repository for macOS

Ctrl+Alt+S 

This page appears in the Settings/Preferences dialog, when the Settings Repository plugin is enabled.

The plugin is bundled with PyCharm and is activated by default. If it is disabled, you can manually [enable the plugin](#).

Use this page to configure the **Settings Repository** feature that allows you to share your IDE settings between different instances of PyCharm (or other IntelliJ platform-based) products installed on different computers.

Tip The settings you are going to share must be stored in a Git repository.

ItemDescription

Auto Sync Select this check box, if you want your local settings to be automatically synchronized with the settings stored in the repository every time you perform an Update Project or a Push operation, or when you close your project or exit PyCharm.
If this option is disabled, you can manually update your settings by choosing VCS | Sync Settings from the main menu.

Read-only sources Use this section to configure additional repositories containing any types of settings you want to share, including live templates, file templates, schemes, deployment options, etc.
These repositories cannot be overwritten or merged, just used as a source of settings as is.

Use the following controls to manage the read-only repositories:

 Click this button to add the URL of the GitHub repository that contains the settings you want to share.

 Click this button to remove the selected repository from the list.

 Click this button to edit the URL of the selected source.

 Use these buttons to move up/down in the list.

 Click this button to clone the selected URL.

Startup Tasks

On this page, create a list of run/debug configurations to be launched automatically on the project start. This may be helpful if you run some [Grunt](#) or [Gulp.js](#) tasks or [npm scripts](#) on a regular basis. All you need is just add the run/debug configurations that launch such tasks or scripts to the list of [startup tasks](#).

On this page:

- [Access](#)
- [Table of Controls](#)

Access

File | Settings | Tools | Startup Tasks for Windows and Linux

PyCharm | Preferences | Tools | Startup Tasks for macOS

Ctrl+Alt+S



Table of Controls

ItemTooltipDescription

	Run Configuration	This read-only field shows the names of the run configurations to be launched on the project start.
	Shared	When this check box is selected, the corresponding task is available for other team members. Note that you can share only those run configurations that are already marked as shared in their definitions. The shared run/debug configurations are kept in separate <code>xml</code> files under <code>.idea\runConfigurations</code> folder, while the local run/debug configurations are kept in the <code>.idea\workspace.xml</code> .
	Add	Click this button to add one of the run/debug configurations that are currently defined in the project to the list of tasks to be executed on the project start. Choose the relevant configurations from the list or choose Edit Configurations to open the Run/Debug Configurations dialog and define a required configuration in it.
	Remove	Click this button to delete the selected task from the list.
	Edit	Click this button to open the Run/Debug Configurations dialog with the settings from the selected configuration and update them as required.

Tasks

This feature is supported in the Professional edition only.

File | Settings | Tools | Tasks for Windows and Linux

PyCharm | Preferences | Tools | Tasks for macOS

Ctrl+Alt+S



Use this page to set up the general options for [tasks and context management](#).

ItemDescription

Changelist name format Type here the format of a changelist to be created. The available placeholders are:

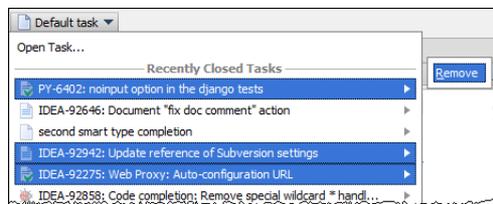
```
{id},{summary},{number}, {project}
```

Task history length Specify the number of tasks to be stored in history.

Connection timeout Sets the maximum time for a connection to be established. A value of zero means the timeout is not used. The default setting is five seconds (5000).

Always display task combo in toolbar If this check box is selected, the combo box of tasks appears in the main toolbar. Use this combo box to [switch between tasks](#), and remove the unnecessary tasks:

toolbar



If this check box is not selected, then the tasks combo box hides, when the only open task is the default one.

Save context on commit If this check box is selected, a new task with the name of the committed changelist is created upon commit. The new task will be available in the tasks combo on the main toolbar.

Enable issue cache – Select this check box to have PyCharm synchronize with the issue tracking system in the background on a regular basis, No matter whether you actually request on information from your issue tracker or not, PyCharm will connect to your issue tracking system according to the specified frequency and refresh the cached issues. The advantage of this approach is that when you need to switch to a task, the up-to-date information is already at your disposal so you do not need to wait till PyCharm establishes connection with the tracker and retrieves the information.

Tip This configuration is especially recommended when working with rather "slow" issue tracking systems.

– Clear this check box to have PyCharm connect to the issue tracking system only when you actually need information on issues, clear the Enable issue cache check box.

Update (N) issues each (M) minutes Specify how often PyCharm should synchronize with your issue tracking system and refresh the cached issues.

Servers

This feature is supported in the Professional edition only.

File | Settings | Tools | Tasks | Servers for Windows and Linux

PyCharm | Preferences | Tools | Tasks | Servers for macOS

Ctrl+Alt+S



Use this page to associate your account in the bug tracking system you use with the PyCharm project. The page is divided into the following sections:

- [Configured servers](#)
- [Server options](#)
 - [General tab](#)
 - [Commit Message tab](#)
 - [Server Configuration tab](#)

Configured servers

Item ShortcutDescription

+ **Alt+Insert** Click to add a new server of one of the supported types. For example:

- [YouTrack](#).
Make sure that REST API in your [YouTrack](#) server is enabled.
- [JIRA](#)
- [GitHub](#)
- Generic

Click **+** to explore the list of the supported servers.

- **Alt+Delete** Click to remove the selected server.

Server options

General tab

Item AvailableDescription for

Share URL All servers If this check box is selected, the server URL of your issue tracking system, specified in the field below becomes available to the other team members.

Server URL All servers Specify here the server URL of your issue tracking system.

Note Important note for [YouTrack](#) users:

If you are using free instance of YouTrack InCloud, make sure to specify <http://csitename/youtrack>. If you are using a paid instance, specify <https://csitename/youtrack>.

Username / Password All except for Lighthouse, Trello, GitLab, GitHub Specify here login name and password for your account.

Note Important note for [Sprintly](#) users:

The Password required here is the API Key from your Sprintly profile (not your Sprintly password). Refer to the [Sprintly documentation](#) for details.

Repository GitHub Specify here the name of the desired Git repository.

Search YouTrack Specify here the default search string.

Project ID Lighthouse, Pivotal Tracker, Redmine Specify here the name of your project in the tracker.

API Token Lighthouse, Pivotal Tracker Specify here the desired API token for the project in question.

Use proxy All servers Select this check box to use the specified proxy server.

Proxy settings All servers Click this button to set up [proxy settings](#).

Use HTTP authentication JIRA Select this check box to enable HTTP authentication.

Test All servers Click this button to make sure the connection is established.

<Template variables> More fields may be added to this tab depending upon the selected or cleared check box Show on first tab in the Template Variables dialog in the [Server Configuration](#) tab.

Commit Message tab

ItemDescription

Add commit message If this check box is selected, a commit message for a changelist, created from an issue in a bug tracking system, is formed according to the syntax defined below.

Message Use the placeholders to define the commit message syntax. The available placeholders are `{id}`, `{summary}`, `{number}`, `{project}`.

Server Configuration tab

This tab only shows for the following servers:

- Generic
- [Asana](#)
- [Assembla](#)
- [Sprint.ly](#)

ItemDescription

Login URL	<p>Specify here the resource for authentication (if any). The request to this resource will take place every time the issues are retrieved from server.</p> <p>This field is disabled, if the check box Use HTTP authentication in the General tab is selected.</p> <p>The resource consists of a URL with possible template variables:</p> <ul style="list-style-type: none">- <code>{serverUrl}</code> : URL of the server, specified in the General tab.- <code>{password}</code>, <code>{username}</code> : correspond to the settings specified in the General tab.- <code>{max}</code> : maximum number of issues to be retrieved from the server. <p>Also it contains the combo-box with the request type: GET or POST.</p>
Tasks list URL	<p>This resource enables obtaining the list of issues from the server and is requested every time the user opens tasks.</p> <p>The resource consists of a URL with possible template variables:</p> <ul style="list-style-type: none">- <code>{serverUrl}</code> : URL of the server, specified in the General tab.- <code>{password}</code>, <code>{username}</code> : correspond to the settings specified in the General tab.- <code>{max}</code> : maximum number of issues to be retrieved from the server. <p>Also it contains the combo-box with the request type: GET or POST.</p>
Single task URL	<p>This resource makes it possible to obtain a more detailed information about a specific issue, using its ID.</p> <p>This field is mandatory, when the check box Each task in separate request is selected.</p> <p>The resource consists of a URL with possible template variables:</p> <ul style="list-style-type: none">- <code>{serverUrl}</code> : URL of the server, specified in the General tab.- <code>{password}</code>, <code>{username}</code> : correspond to the settings specified in the General tab.- <code>{max}</code> : maximum number of issues to be retrieved from the server. <p>Also it contains the combo-box with the request type: GET or POST.</p>
Response type	<p>This section contains three radio buttons:</p> <ul style="list-style-type: none">- XML: click this radio button to return the server response in XML format. So doing, the selectors are described in the format XPath.- JSON: click this radio button to return the server response in JSON format. So doing, the selectors are described in the format JSONPath.- Text: click this radio button to retrieve issues fields using regular expressions.
Each task in separate request	<p>Select this check box to make the field Single task URL mandatory.</p>
Table of selectors	<p>These selectors help specifying how exactly the detailed issue information will be extracted from the server response.</p> <p>The selectors have the following meanings:</p> <ul style="list-style-type: none">- tasks: path in the server response to the descriptions of specific issues (mandatory selector).- id: path to the unique ID within the description of a specific issue (mandatory selector).- summary: path to the title/header of an issue inside its description (mandatory selector).- description: path to the detailed description of an issue (optional selector).- updated/created: path to the time of updating or creating the issue respectively, in the format ISO-8601, inside its description (optional selector).- closed: path to a flag that defines whether a specific issue is opened or closed (optional selector).- issueUrl: path to a URL inside the issue description. It is possible to open information about this issue in your browser. <p>Depending on the choice of response type, these selectors are described in XPath or JSONPath format, or as regular expressions.</p>
Reset to Defaults	<p>Click this button to discard all settings and return to the default values.</p>
Manage Template Variables...	<p>Click this button to show the dialog box Template Variables with the list of existing template variables. Use + and - buttons to manage the list. Note the check boxes in the columns Show on first tab and Hide.</p> <ul style="list-style-type: none">- If a check box in the column Show on first tab next to a custom template variable is selected, this custom variable is displayed in the General tab.- If a check box in the column Hide next to a custom template variable is selected, the value of this variable is shown dots. Otherwise, it is shown as is. <p>Note that default variables cannot be deleted or changed!</p>
Test	<p>Click this button to make sure the settings are correct.</p>

Vagrant

This feature is supported in the Professional edition only.

File | Settings | Vagrant for Windows and Linux

PyCharm | Preferences | Vagrant for macOS

Ctrl+Alt+S



Use this page to:

- Enable [Vagrant](#) support in PyCharm by specifying the Vagrant executable file.
- Specify the location of the [VagrantFile](#) that determines creation of **virtual boxes (instances)** by provisioning predefined **Vagrant base boxes** through the `vagrant up` command.
- Handle the list of **Vagrant base boxes** to use in creation of **virtual boxes (instances)**.

Before you start working with **Vagrant**, make sure that:

1. [Vagrant](#) is downloaded and installed.
2. Before you start working with Vagrant, make sure that the Vagrant plugin is enabled. The plugin is bundled with PyCharm and is activated by default. If the plugin is not activated, enable it on the [Plugins](#) page of the [Settings / Preferences Dialog](#) as described in [Enabling and Disabling Plugins](#).

ItemDescription

Vagrant executable	Specify the fully qualified address of the executable file: <code>vagrant.bat</code> for Windows, <code>vagrant</code> for Unix and macOS. Type the path manually, or click the browse button and locate the desired file in the Select vagrant executable dialog box.
Instance folder	<p>Specify here the fully qualified path to the directory, where the task <code>vagrant init</code> has been executed, and the <code>Vagrantfile</code> is initialized and stored. Note that you can create an instance folder in any location, for example, in a project root. When a new remote interpreter is created, this project root location will be suggested by default as the instance folder.</p> <p>A <code>Vagrantfile</code> is a configuration file that defines the instance (virtual machine) you need. The file contains the virtual IP address, port mappings, and the memory size to assign. The file can specify which folders are shared and which third-party software should be installed. According to the <code>Vagrantfile</code> your instance (virtual machine) is configured, provisioned against the relevant Vagrant base box, and deployed on your computer. A <code>Vagrantfile</code> is created through the <code>vagrant init</code> command.</p> <p>When creation of an instance (virtual machine) is invoked either through the <code>vagrant up</code> command or through the Tools Vagrant Up menu option, PyCharm looks for the <code>Vagrantfile</code> in the directory specified in the Instance folder field. For more information, see http://docs.vagrantup.com/v2/vagrantfile/.</p> <p>You can create a <code>Vagrantfile</code> in any directory and appoint it as instance folder. If the field is empty, PyCharm will treat the project root as the instance folder and look for a <code>Vagrantfile</code> in it.</p>

Provider Use this field to specify the [provider](#) to be used by `vagrant up` command. If this field is left blank, the default provider is used.

Environment variable Click the ellipsis button or press `Shift+Enter` to specify the shell variables to be used to configure the providers' behavior.

Boxes and Plugins tabs

Boxes This list shows the predefined [Vagrant base boxes](#) available in PyCharm. Each item presents a **Vagrant base box** on which Vagrant configures and launches its **instances (virtual machines)**. The entries of this list correspond to the output of the command `vagrant box list`.

 `Alt+Insert` Click this button to download a new base box. This command corresponds to `vagrant box add <name> <URL>`. By default, PyCharm suggests the URL to the `lucid32` box. For details, see [Creating and Removing Vagrant Boxes](#).

 `Alt+Delete` Click this button to remove the selected **Vagrant base box**. So doing, the box and the nested files are physically deleted from the disk. This command corresponds to `vagrant box remove <name>`. For details, see [Creating and Removing Vagrant Boxes](#).

Plugins Use this table to view and change the list of available plugins.

 `Alt+Insert` Click this button to install a new Vagrant plugin.

 `Alt+Delete` Click this button to remove the selected plugin.

 Click this button to update the selected plugin.

 Use this button to attach a license to the selected plugin.

Keyboard Shortcuts and Mouse Reference

Using shortcuts is a major way to maximum efficiency and productivity of PyCharm. This part lists keystroke combinations and their functions for the Default keymap defined in [Keymap](#) dialog box.

Warning! The key combinations documented in this part may fail to perform the function described, if you are using a [customized keymap](#).

To get a printable copy of the default keymap, choose Help | Default Keymap Reference on the main menu.

Tip Note that in certain operating systems the key and mouse combinations may not work as described here. In this case, it's necessary to tweak the operating system's keymap. For example, if you are using Ubuntu, mind the windows manager, whose [shortcuts conflict](#) with that of PyCharm.

In this part:

- [Keyboard Shortcuts By Category](#)
- [Keyboard Shortcuts By Keystroke](#)

Keyboard Shortcuts By Category

In this part you can find reference information about the keyboard shortcuts grouped by functional categories:

- [Advanced Editing](#)
- [Basic Editing](#)
- [Code Folding](#)
- [Running and Debugging](#)
- [General](#)
- [Search](#)
- [Navigation Between Bookmarks](#)
- [Navigation Between IDE Components](#)
- [Navigation In Source Code](#)
- [Refactoring](#)

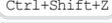
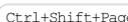
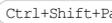
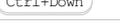
Advanced Editing

Function Shortcut Use this shortcut to...

Comment with Line Comment	Ctrl+Slash	Comment/uncomment current line or selected block with line comments.
Comment with Block Comment	Ctrl+Shift+Slash	Comment/uncomment code with block comments.
Quick Documentation	Ctrl+Q / Alt+Button2 Click	Show a pop-up window with the documentation for the symbol at the caret.
Pop-up Hecor	Ctrl+Shift+Alt+H	Show the Highlighting level pop-up window to configure highlighting in the current file.
Parameter Info	Ctrl+P	Show parameters of the method call at the caret.
Context Info	Alt+Q	Show the current method or class declaration when it is not visible.
Error Description	Ctrl+F1	Show an error or warning description at the caret.
External Documentation	Shift+F1	Open browser with the documentation for the selected item.
Override Methods...	Ctrl+O	Override base class methods in the current class.
Surround with...	Ctrl+Alt+T	Surround selected code fragment with <code>if</code> , <code>do</code> , tags or other constructs.
Basic Code Completion	Ctrl+Space Alt+Slash	Code completion for any class, method or variable.
SmartType Code Completion	Ctrl+Shift+Space	Code Completion filtering the lookup list basing on expected type.
Expand Word	Alt+Slash	Goes through the names of classes, methods, keywords and variables in the current visibility scope.
Insert Live Template...	Ctrl+J	Show a pop-up list of starting with a specified prefix.
Surround with Live Template...	Ctrl+Alt+J	Surround the selection with one of the templates.
Next Template Variable	Tab	In templates: move the caret to the next template variable.
Previous Template Variable	Shift+Tab	In templates: move the caret to the previous template variable.

Basic Editing

Function Shortcut Use this shortcut to...

Enter		Depending on the context: <ul style="list-style-type: none">- In a lookup list: select an item.- In the editor: enter a new line and set the caret at its beginning.- In the editor: enter a new line and set the caret at its beginning. On pressing  , PyCharm adds backslash character to avoid syntactical errors.
Tab		In the editor: <ul style="list-style-type: none">- With selection: indent selected lines.- Without selection: insert a tab symbol (or corresponding number of space characters). In a lookup list: <ul style="list-style-type: none">- No code after the caret: select an item.- Some code after the caret: select an item and substitute the code after the caret with it.
Delete		Depending on the context: <ul style="list-style-type: none">- In the editor: delete selected symbol/block.- In a usage view: exclude a selected item.- In other views: remove selected items.
Backspace		Delete a character to the left of the caret.
Undo		Undo last operation.
Redo		Redo last undone operation.
Cut		Cut a current line or a selected code block to the Clipboard.
Copy		Copy a current line or a selected code block to the Clipboard.
Paste		Paste from the Clipboard to the caret location.
Paste from History		Paste selected entry from the Clipboard to the caret location.
Up		Move the caret one line up.
Up with Selection		Move the caret one line up selecting the text.
Down		Move the caret one line down.
Down with Selection		Move the caret one line down selecting the text.
Left		Move the caret one character to the left.
Left with Selection		Move the caret one character to the left selecting the text.
Right		Move the caret one character to the right.
Right with Selection		Move the caret one character to the right selecting the text.
Go to Page Bottom		Move the caret down to the page bottom.
Go to Page Bottom with Selection		Move the caret down to the page bottom, selecting the text.
Go to Page Top		Move the caret up to the page top.
Go to Page Top with Selection		Move the caret up to the page bottom, selecting the text.
Page Down		Move the caret one page down.
Page Down with Selection		Move the caret one page down, selecting the text.
Page Up		Move the caret one page up.
Page Up with Selection		Move the caret one page up, selecting the text.
Scroll Down		Scroll the text one line down.
Scroll to Center		Scroll a line at caret to the center of the screen.
Scroll Up		Scroll the text one line up.
Move to Line End		Move the caret to the end of line.
Move to Line End with Selection		Move the caret to the end of line, selecting the text.
Move to Line Start		Move the caret to the beginning of line.
Move to Line Start with Selection		Move the caret to the beginning of line, selecting the text.
Move to Next Word		Move the caret to the next word.
Move to Next Word with Selection		Move the caret to the next word, selecting it.
Move to Previous Word		Move the caret to the previous word.

Move to Previous Word with Selection	Ctrl+Shift+Left	Move the caret to the previous word, selecting it.
Move to Text End	Ctrl+End	Move the caret to the end of text.
Move to Text End with Selection	Ctrl+Shift+End	Move the caret to the end of text, selecting it.
Move to Text Start	Ctrl+Home	Move the caret to the beginning of text.
Move to Text Start with Selection.	Ctrl+Shift+Home	Move the caret to the beginning of text, selecting it.
Select All	Ctrl+A	Select the entire text opened in the editor.
Delete Line at Caret	Ctrl+Y	Delete the line where the caret is currently located.
Delete to Word End	Ctrl+Delete	Delete the word starting from the current caret location up to the word end.
Delete to Word Start	Ctrl+Backspace	Delete the word starting from the current caret location up to the word start.
Toggle Insert/Overwrite	Insert	Toggle insert/overwrite modes.
Duplicate Line or Block	Ctrl+D	Duplicate selected block or the line at the caret.
Toggle Case	Ctrl+Shift+U	Toggle case of the selected text block.
Move to Code Block End	Ctrl+Close Bracket	Move the caret to the current code block end, highlighting the block limits.
Move to Code Block End with Selection	Ctrl+Shift+Close Bracket	Move the caret to the current code block end, selecting the code beginning from the initial caret location.
Move to Code Block Start	Ctrl+Open Bracket	Move the caret to the current code block start, highlighting the block limits.
Move to Code Block Start with Selection	Ctrl+Shift+Open Bracket	Move the caret to the current code block start, selecting the code beginning from the initial caret location.
Start New Line	Shift+Enter	Start a new line after the current one positioning the caret in accordance with the current indentation level.
Start New Line Before Current One	Ctrl+Alt+Enter	Start a new line before the current one.
Join Lines	Ctrl+Shift+J	Concatenate the selected lines into one or concatenate the line where the caret is currently located with the next line.
Split Line	Ctrl+Enter	Split the selected line at the point where the caret is located, leaving the caret at the end of the first line. This shortcut splits a line without adding a backslash.
Select Word at Caret	Ctrl+W	Select successively increasing code blocks starting from the current caret location.
Unselect Word at Caret	Ctrl+Shift+W	Remove sequentially the selection made by the action.
Indent Selection	Tab	Move the selected block to the next indentation level.
Unindent Selection	Shift+Tab	Move the selected block to the previous indentation level.
Auto-Indent Lines	Ctrl+Alt+I	Indent the current line or selected block according to the General settings.

Code Folding

CommandShortcutDescription

Expand	Ctrl+NumPad Plus	Expand the current collapsed fragment
Collapse	Ctrl+NumPad -	Collapse the current folding region
Expand Recursively	Ctrl+Alt+NumPad Plus	Expand the current folded fragment and all the subordinate collapsed folding regions within that fragment
Collapse Recursively	Ctrl+Alt+NumPad -	Collapse the current folding region and all the subordinate folding regions within it
Expand All	Ctrl+Shift+NumPad Plus	Expand all collapsed fragments within the selection, or, if nothing is selected, expand all the collapsed fragments in the current file
Collapse All	Ctrl+Shift+NumPad -	Collapse all folding regions within the selection, or, if nothing is selected, collapse all the folding regions in the current file
Expand to level 1, 2, 3, 4 or 5	Ctrl+NumPad *, 1 Ctrl+NumPad *, 2 Ctrl+NumPad *, 3 Ctrl+NumPad *, 4 Ctrl+NumPad *, 5	Expand the current fragment and all the nested fragments up to the specified level
Expand all to level 1, 2, 3, 4 or 5	Ctrl+Shift+NumPad *, 1 Ctrl+Shift+NumPad *, 2 Ctrl+Shift+NumPad *, 3 Ctrl+Shift+NumPad *, 4 Ctrl+Shift+NumPad *, 5	Expand all the collapsed fragments in the file up to the specified nesting level
Fold Selection / Remove region	Ctrl+Period	Collapse the selected fragment and create a custom folding region for it to make it "foldable" / Expand the current fragment and remove the corresponding custom folding region to make the fragment "unfoldable"

Running and Debugging

Function Shortcut Use this shortcut to...

Run	Shift+F10	Run a program.
Choose configuration and run	Shift+Alt+F10	Quickly select run/debug configuration and run or edit it.
Rerun	Ctrl+F5	Repeat execution with the same settings, with the same tab of the Run tool window having the focus.
Rerun without losing the focus in the editor	Shift+F10	Repeat execution with the same settings, with the same tab of the editor having the focus.
Debug	Shift+F9	Debug a program.
Choose configuration and debug	Shift+Alt+F9	Quickly select run/debug configuration and debug or edit it.
Step Over	F8	Step to the next line in the current file. See Stepping Through the Program .
Step Into	F7	Step to the next executed line. See Stepping Through the Program .
Smart Step Into	Shift+F7	Select the method to step in, if the current line contains multiple method call expressions. See Choosing a Method to Step Into .
Step Out	Shift+F8	Step to a first executed line after returning from the current method. See Stepping Through the Program .
Force Step Over	Shift+Alt+F8	Run until the next line in this method or file, skipping the methods referenced at the current execution point and ignoring breakpoints. See Stepping Through the Program .
Force Step Into	Shift+Alt+F7	Steps into the method called in the current execution point even if this method is to be skipped. See Stepping Through the Program .
Run to Cursor	Alt+F9	Run to the line where the caret is located. See Stepping Through the Program .
Force Run To Cursor	Ctrl+Alt+F9	Run to the line where the caret is located, ignoring breakpoints. See Stepping Through the Program .
Resume Program	F9	Resume program execution.
Stop Program	Shift+F2	Terminate a debugging session.
Evaluate Expression	Alt+F8	Evaluate an arbitrary expression.
Quick Evaluate Expression	Ctrl+Alt+F8	Evaluate an arbitrary expression without calling Evaluate Expression dialog.
Toggle Breakpoint	Ctrl+F8	Toggle breakpoint at the current line.
View Breakpoints	Ctrl+Shift+F8	View/manage all breakpoints.

General

Function **Shortcut** **Use this shortcut to...**

Close Active Tab	Ctrl+Shift+F4	Close an active tab in a tool window (for example, Find tool window).
Close Editor	Ctrl+F4	Close an active editor.
Edit Source	F4	Open an editor for the selected item or items and give focus to the last opened file.
Escape	Escape	Depending on the context: <ul style="list-style-type: none">– In the editor: close pop-up windows, terminate search, or remove highlighting.– In a tool window: return focus to the editor.
Export to Text File	Alt+O	Export contents of a tool window to a text file. This feature applies to the Version Control Tool Window , Messages Tool Window , and other tool windows that provide the export button  on the window toolbar.
New...	Alt+Insert	Create a new class, file or directory. See Populating Modules .
Save All	Ctrl+S	Save all files and settings.
Select Next Tab	Alt+Right	When several tabs are open in the editor or a view, open the next tab to the right (or first tab if the current one is the last).
Select Previous Tab	Alt+Left	When several tabs are open in the editor or a view, open the next tab to the left (or last tab if the current one is the first).
Show Intention Action	Alt+Enter	Display intention actions (if any) for the code where the caret is currently located, or the selected GUI component in a form.
Synchronize	Ctrl+Alt+Y	Detect all externally changed files and reload them from disk.
View Source	Ctrl+Enter	Depending on the context: <ul style="list-style-type: none">– In Tool Windows: Open an Editor tab or tabs for the selected item or items, respectively.– In the editor: Intelligently split the current line into 2 lines, shifting quotes, etc. as necessary.

Search

Function **Shortcut** **Use this shortcut to...**

Find	Ctrl+F	Initiate text search .
Replace	Ctrl+R	Initiate text search and replace .
Search for next/ previous occurrence	F3 / Shift+F3	Navigate to the next/previous occurrence of a selected word in the editor.
Find Word at Caret	Ctrl+F3	Search in the editor for the word where the caret is currently located.
Incremental Search	Ctrl+F	Initiate text search .
Find in Path	Ctrl+Shift+F	Initiate search for a text string in the specified scope .
Replace in Path	Ctrl+Shift+R	Initiate search and replace in the specified scope .
Find Usages	Alt+F7	Initiate search for usages of the selected symbol in the specified scope .
Find Usages in File	Ctrl+F7	Initiate search for usages of the selected symbol in the current file .
Highlight Usages in File	Ctrl+Shift+F7	Highlight usages of a symbol at caret.
Show Usages	Ctrl+Alt+F7	Show usages of a symbol at caret in a pop-up window. Use list of found usages to jump to the desired location.
Find Action	Ctrl+Shift+A	Find an action, bypassing menus. See Finding Actions .

Navigation Between Bookmarks

Function **Shortcut** **Use this shortcut to...**

Go to Bookmark <number>

Ctrl+Number

Navigate to a [numbered bookmark](#) with the corresponding number.

Toggle Bookmark

F11

Turn anonymous bookmark on or off.

Toggle Bookmark with Mnemonic

Ctrl+F11

Turn bookmark with mnemonic on or off.

Show Bookmarks

Shift+F11

Open [Bookmarks](#) dialog to manage existing bookmarks and navigate between them.

Navigation Between IDE Components

In this section you can find keyboard shortcuts for navigation between:

- [Views and Windows](#).
- [Differences](#).

Views and Windows

Function **Shortcut** **Use this shortcut to...**

Select Target	Alt+F1	Move focus from the current file, class, method or reference to a data source, to a view suggested in the Select Target pop-up menu. Refer to Navigating Between IDE Components .
Collapse all	Ctrl+NumPad -	Collapse all nodes in a tree view.
Expand all	Ctrl+NumPad Plus	Expand all nodes in a tree view.
Switcher	Ctrl+Tab	Navigate between files opened in the editor, and tool windows.
Open tool window	Alt+Number	Open a tool window with the specified number.
Hide Active Window	Shift+Escape	Hide the currently active tool window.
Jump to Last Window	F12	Activate the last focused tool window.

Differences

Function **Shortcut** **Use this shortcut to...**

Move to Next Difference	F7	Navigate to the next difference in view .
Move to Previous Difference	Shift+F7	Navigate to the previous difference in view .

Navigation In Source Code

Function **Shortcut** **Use this shortcut to...**

File Structure Pop-up	Ctrl+F12	Display the Structure pop-up window for quick navigation through the current file.
Select target	Alt+F1	Move focus from the current file, class, method or reference to a data source table to a view suggested in the Select Target pop-up menu. See Navigating Between IDE Components .
Recent Files	Ctrl+E	Show the list of recently opened files .
Recently Changed Files	Ctrl+Shift+E	Show the list of recently updated files .
Type Hierarchy	Ctrl+H	Browse hierarchy for the selected class class .
Navigate to Class	Ctrl+N	Navigate directly to a class in project by specifying its name in a pop-up dialog box.
Navigate to File	Ctrl+Shift+N	Navigate directly to a file in project by specifying its name in a pop-up dialog box.
Navigate to Recently Opened File	Ctrl+E	Show the list of recently opened files .
Navigate to Recently Changed File	Ctrl+Shift+E	Show the list of recently updated files .
Navigate to Line	Ctrl+G	Navigate to any line in the current file by specifying its number.
Navigate to Declaration	Ctrl+B	Navigate to declaration of a symbol at caret.
Navigate to Implementation	Ctrl+Alt+B	Navigate to implementation of the item at caret.
Navigate to Type Declaration	Ctrl+Shift+B	Navigate to a type declaration of a symbol at caret, the symbol being a variable or a method call.
Navigate to Super Method	Ctrl+U	Navigate to a super method declaration of a method under the caret.
Navigate to Test/Test Subject	Ctrl+Shift+T	Navigate to a test for the class at caret, if any, or navigate from a test to a test subject.
Navigate to Related Symbol	Ctrl+Alt+Home	Navigate between files with complicated relationships between them . For example, use this shortcut to navigate between views and templates.
Navigate to Next Method	Alt+Down	Navigate to the next method declaration in the active editor tab.
Navigate to Previous Method	Alt+Up	Navigate to the previous method declaration in the active editor tab.
Navigate to Opening Brace	Ctrl+Open Bracket	Navigate to the start of the current code block.
Navigate to Closing Brace	Ctrl+Close Bracket	Navigate to the end of the current code block.
Back	Ctrl+Alt+Left	Undo last navigation operation. Note On an macOS computer, you can also use the three-finger right-to-left swipe gesture.
Forward	Ctrl+Alt+Right	Redo last undone navigation operation. Note On an macOS computer, you can also use the three-finger left-to-right swipe gesture.
Navigate to Previous Occurrence	Ctrl+Alt+Up	Navigate to a previous found item .
Navigate to Next Occurrence	Ctrl+Alt+Down	Navigate to a next found item .
Last Edit Location	Ctrl+Shift+Backspace	Move through the most recent change points.
Navigate to Next Highlighted Error	F2	Navigate to the next found error/warning.
Navigate to Previous Highlighted Error	Shift+F2	Navigate to the previous found error/warning.

Refactoring

Function **Shortcut** **Use this shortcut to...**

Rename	Shift+F6	Rename the selected file, class, field, method, etc. and change all references to it accordingly.
Change Method Signature	Ctrl+F6	Change the signature of the selected method and update all the corresponding method calls.
Move	F6	Move the selected class, package or static member to another package or class and update all the corresponding references.
Copy	F5	Create a copy of the selected class, file or directory in the same or different directory or package.
Clone		Create a copy of the selected class in the same package.
Safe Delete	Alt+Delete	Delete the selected class, method or field checking its usages.
Extract Method	Ctrl+Alt+M	Turn the selected code fragment into a method.
Extract Variable	Ctrl+Alt+V	Create a new variable and use the selected expression as its value.
Extract Field	Ctrl+Alt+F	Create a new field and use the selected expression as its value.
Extract Constant	Ctrl+Alt+C	Create a new constant (static final field) and use the selected expression as its value.
Extract Parameter	Ctrl+Alt+P	Turn the selected expression into a new method parameter.
Inline	Ctrl+Alt+N	Inline the selected method or variable.

Keyboard Shortcuts By Keystroke

In this part you can find reference information on keyboard shortcuts grouped by keystroke:

- [Alt](#)
- [Alt+Shift](#)
- [Ctrl](#)
- [Ctrl+Alt](#)
- [Ctrl+Shift](#)
- [Function Keys](#)
- [Insert, Delete and Navigation Keys](#)
- [Shift](#)
- [Ctrl+Alt+Shift](#)

To view a full list of available shortcuts, navigate to File | Settings and click Keymap under IDE Settings.

Alt

This section lists and describes the keyboard shortcuts that include the [Alt](#) key:

- [Alt+Alphanumeric keys](#)
- [Alt+Navigation keys](#)
- [Alt+Function \(F\) keys](#)

Alt+Alphanumeric keys

ShortcutFunction Use this shortcut to...

Alt+O	Export to Text File	Export a tool window's content to a text file.
Alt+Q	Context Info	Show the current method or class declaration when it is not visible.
Alt+Number	Open tool window	Open a tool window with the corresponding number.
Alt+Slash	Code completion / Expand word	Expand string at caret to any word in the visible scope that starts with the same characters.
Alt+Back Quote	VCS operations	Show quick list with the most required version control commands.

Alt+Navigation keys

ShortcutFunction Use this shortcut to...

Alt+Delete	Safe Delete	Delete selected class/method/field checking its usages.
Alt+Enter	Show Intention Action	Display intention actions (if any) for a code where the caret is currently located.
Alt+Home	Activate Navigation Bar	Bring focus to the Navigation bar .
Alt+Insert	Create new entity	Depending on the context: <ul style="list-style-type: none">– In the navigation views: create a new class, file or directory, using the New pop-up menu.
Alt+Down	Navigate to Next Method	Navigate to the next method declaration in the active editor tab.
Alt+Left	Select Previous Tab	Depending on the context: <ul style="list-style-type: none">– When several tabs are opened in the editor or a view, open the next tab to the left (or the last tab if the current one is the first).– In the Differences Viewer for Files invoked from the Update Project Info tab of the Version Control tool window, compare the local copy of the previous file with its update from the server.
Alt+Right	Select Next Tab	Depending on the context: <ul style="list-style-type: none">– When several tabs are opened in the editor or a view, open the next tab to the right (or the first tab if the current one is the last).– In the Differences Viewer for Files invoked from the Update Project Info tab of the Version Control tool window, compare the local copy of the next file with its update from the server.
Alt+Up	Navigate to Previous Method	Navigate to the previous method declaration in the active editor tab.

Alt+Function (F) keys

ShortcutFunction Use this shortcut to...

Alt+F1	Select Target	Move focus from the current file, class, method or reference to a data source table to a view suggested in the Select Target pop-up menu. See Navigating Between IDE Components .
Alt+F7	Find Usages	Initiate search for usages .
Alt+F8	Evaluate Expression	Debugger: Evaluate an arbitrary expression .
Alt+F9	Run to Cursor	Debugger: Run to the line where the caret is located.

Alt+Shift

This section lists and describes the keyboard shortcuts that include the **Shift+Alt** keys:

ShortcutFunctionUse this shortcut to...

Shift+Alt+F7	Force Step Into	Step into the method called in the current execution point, even if this method is to be skipped.
Shift+Alt+F8	Force Step Over	Run until the next line in this method or file, skipping the methods referenced at the current execution point and ignoring breakpoints.
Shift+Alt+F9	Debug	Quickly select run/debug configuration and debug/edit it.
Shift+Alt+F10	Run	Quickly select run/debug configuration and run/edit it.

Ctrl

This section lists and describes the keyboard shortcuts that include the `Ctrl` key:

- [Ctrl+Alphanumeric keys](#)
- [Ctrl+Navigation keys](#)
- [Ctrl+Symbol keys](#)
- [Ctrl+Numpad keys](#)
- [Ctrl+Function \(F\) keys](#)

Ctrl+Alphanumeric keys

Shortcut**Function****Use this shortcut to...**

<code>Ctrl+A</code>	Select All	Select the entire text in the active editor.
<code>Ctrl+B</code>	Navigate to Declaration	Navigate directly to an element's declaration from any usage.
<code>Ctrl+C</code>	Copy	Copy selected text to the Clipboard.
<code>Ctrl+D</code>	Duplicate Line or Block	Duplicate selected block or line at caret.
<code>Ctrl+E</code>	Recent Files Recent find usages	Show the list of recently opened files . When the Find tool window has the focus, use this shortcut to show the list of recent find usages results .
<code>Ctrl+F</code>	Find	Initiate text search in the editor.
<code>Ctrl+G</code>	Navigate to Line	Navigate to a line with the specified number in the current file.
<code>Ctrl+H</code>	Type Hierarchy	Browse hierarchy for the selected class.
<code>Ctrl+J</code>	Insert Live Template	Show a pop-up list of Live Templates starting with a specified prefix.
<code>Ctrl+M</code>	Scroll to Center	Scroll a line at caret to the center of the screen.
<code>Ctrl+N</code>	Navigate to Class	Jump to a class in the project with the specified name.
<code>Ctrl+O</code>	Override Methods	Override base class methods in the current class.
<code>Ctrl+P</code>	Parameter Info	Show parameters of the method call at the caret.
<code>Ctrl+Q</code>	Quick documentation	Show a pop-up window with documentation for the symbol at caret. In the Database tool window , show a pop-up window that displays the <code>create table</code> query for the database table at the caret and the first 10 rows of the table.
<code>Alt+Mouse Button2</code>		
<code>Ctrl+R</code>	Replace	Call the Replace Text dialog box.
<code>Ctrl+S</code>	Save All	Save all files and settings.
<code>Ctrl+U</code>	Navigate to Super Method	Navigate to a super method declaration of a method at caret
<code>Ctrl+V</code>	Paste	Paste from the Clipboard.
<code>Ctrl+W</code>	Select Word at Caret	Successively select expanding blocks of text, starting from the word at caret. (Use this shortcut repeatedly to select expressions.)
<code>Ctrl+X</code>	Cut	Cut to the Clipboard.
<code>Ctrl+Y</code>	Delete Line at Caret	Delete a word starting from the current caret location up to the end of word.
<code>Ctrl+Z</code>	Undo	Undo last operation.
<code>Ctrl+Shift+Z</code>	Redo	Redo last undone operation.
<code>Ctrl+Number</code>	Navigate to bookmark	Navigate to a numbered bookmark with corresponding number.

Ctrl+Navigation keys

Shortcut**Function****Use this shortcut to...**

<code>Ctrl+Tab</code>	Switcher	Navigate between the files opened in the editor, and tool windows.
<code>Ctrl+Backspace</code>	Delete to Word Start	Delete a word starting from the current caret location up to the word start.
<code>Ctrl+Delete</code>	Delete to Word End	Delete a word starting from the current caret location up to the word end.
<code>Ctrl+End</code>	Move to Text End	Move the caret to the end of text.
<code>Ctrl+Enter</code>	Split Line or Open Item	Depending on the context: - In the editor: intelligently split the current line into 2 lines, shifting quotes, etc. as necessary.

– In the Tool Windows: Open an Editor tab or tabs for the selected item or items, respectively.

Ctrl+Home	Move to Text Start	Jump to the beginning of the text.
Ctrl+C	Copy	Copy a current line or a selected code block to the Clipboard.
Ctrl+Space	Basic Code Completion	Complete code for any class, method or variable.
Ctrl+Page Down	Navigate to Page Bottom	Move the caret down to the page bottom.
Ctrl+Page Up	Navigate to Page Top	Move the caret up to the page top.
Ctrl+Down	Scroll Down	Move line at caret one down, preserving syntactical correctness.
Ctrl+Left	Move to Previous Word	Move the caret to the previous word.
Ctrl+Right	Move to Next Word	Move the caret to the next word.
Ctrl+Up	Scroll Up	Move line at caret one up, preserving syntactical correctness.
Ctrl+Shift+Up		
Ctrl + Ctrl + Home / End		Select text from the caret position to the beginning/end of the current line.

Ctrl+Symbol keys

ShortcutFunctionUse this shortcut to...

Ctrl+Open Bracket	Move to Code Block Start	Move the caret to the beginning of the current code block, highlighting its limits.
Ctrl+Close Bracket	Move to Code Block End	Move the caret to the end of the current code block, highlighting its limits.
Ctrl+Slash	Comment with Line Comment	Comment/uncomment current line or selected block with line comments.
Ctrl+Numpad/		
Ctrl+=	Expand All	Expand all folding blocks.
Ctrl+NumPad Plus		
Ctrl+NumPad -	Collapse All	Collapse all folding blocks.

Ctrl+Numpad keys

ShortcutFunctionUse this shortcut to...

Ctrl+Numpad/	Comment with Line Comment	Comment/uncomment current line or selected block with line comments.
Ctrl+Slash		
Ctrl+NumPad Plus	Expand All	Expand all folding blocks.
Ctrl+=		
Ctrl+NumPad -	Collapse All	Collapse all folding blocks.

Ctrl+Function (F) keys

ShortcutFunctionUse this shortcut to...

Ctrl+F1	Error Description	Show an error or warning description at the caret.
Ctrl+F3	Find Word at Caret	Search in the editor for the word where the caret is currently located.
Ctrl+F6	Change Method Signature	Refactor a selected method signature and update all references.
Ctrl+F7	Find Usages in File	Initiate search for usages .
Ctrl+F8	Toggle Breakpoint	Toggle breakpoint at caret.
Ctrl+F9	Make Project	Compile all modified and dependent files in a project.
Ctrl+F11	Toggle Bookmark with mnemonic.	Turn bookmark with mnemonic on or off.
Ctrl+F12	File Structure Pop-up	Show the current file structure in the File Structure pop-up window for quick navigation.

Ctrl+Alt

This section lists and describes the keyboard shortcuts that include the `Ctrl+Alt` keys:

- [Ctrl+Alt+Alphanumeric keys](#)
- [Ctrl+Alt+Navigation keys](#)
- [Ctrl+Alt+Function \(F\) keys](#)

Ctrl+Alt+Alphanumeric keys

ShortcutFunctionUse this shortcut to...

<code>Ctrl+Alt+B</code>	Navigate to Implementation	Navigate to implementation of an item at the caret.
<code>Ctrl+Alt+C</code>	Extract Constant	Replace selected expression with a constant (static final field) (Refactoring).
<code>Ctrl+Alt+F</code>	Extract Field	Put the selected expression result into a field (Refactoring).
<code>Ctrl+Alt+H</code>	Call Hierarchy	Browse call hierarchy for the selected method. See page Viewing Structure and Hierarchy of the Source Code
<code>Ctrl+Alt+I</code>	Auto-indent Lines	Indent current line or selected block according to the Code Style settings .
<code>Ctrl+Alt+J</code>	Surround with Live Template	Surround the selection with one of the Live Templates.
<code>Ctrl+Alt+M</code>	Extract Method	Create a method from the selected code (Refactoring).
<code>Ctrl+Alt+N</code>	Inline	Inline the selected method/variable (Refactoring).
<code>Ctrl+Alt+P</code>	Extract Parameter	Turn the selected expression into a method parameter (Refactoring).
<code>Ctrl+Alt+T</code>	Surround with	Surround selected code fragment with <code>if</code> , <code>while</code> , <code>try/catch</code> , or another construct.
<code>Ctrl+Alt+V</code>	Extract Variable	Put selected expression result into a variable (Refactoring). See page Extract Variable .
<code>Ctrl+Alt+Y</code>	Synchronize	Detect all externally changed files and reload them from disk.

Ctrl+Alt+Navigation keys

ShortcutFunctionUse this shortcut to...

<code>Ctrl+Alt+Enter</code>	Start new line before current one	Start a new line before the current one.
<code>Ctrl+Alt+Down</code>	Navigate to Next/Previous Occurrence	Navigate to the next/previous found item.
<code>Ctrl+Alt+Up</code>		
<code>Ctrl+Alt+Left</code>	Back	Undo last navigation operation. See page Navigating to Navigated Items Note On an macOS computer, you can also use the three-finger right-to-left swipe gesture.
<code>Ctrl+Alt+Right</code>	Forward	Redo last undone navigation operation. See page Navigating to Navigated Items Note On an macOS computer, you can also use the three-finger left-to-right swipe gesture.
<code>Ctrl+Alt+Home</code>	Navigate to Related Symbol	Navigates between files with the various relationships. See Navigation In Source Code .

Ctrl+Alt+Function (F) keys

ShortcutFunctionUse this shortcut to...

<code>Ctrl+Alt+F6</code>	Switch to another coverage suite.	Open the Coverage Suites pop-up window and select the desired suite to run.
<code>Ctrl+Alt+F7</code>	Show usages	Show usages of a symbol at the caret. See page Viewing Usages of a Symbol
<code>Ctrl+Alt+F8</code>	Quick Evaluate Expression	Evaluate an arbitrary expression without calling Evaluate Expression dialog box.
<code>Ctrl+Alt+F9</code>	Force Run To Cursor	Run to the line where the caret is located, ignoring breakpoints. See page Stepping Through the Program .

Ctrl+Shift

This section lists and describes the keyboard shortcuts that include the `Ctrl+Shift` keys:

- [Ctrl+Shift+Alphanumeric keys](#)
- [Ctrl+Shift+Navigation keys](#)
- [Ctrl+Shift+Symbol keys](#)
- [Ctrl+Shift+Numpad keys](#)
- [Ctrl+Shift+Function \(F\) keys](#)

Ctrl+Shift+Alphanumeric keys

ShortcutFunctionUse this shortcut to...

<code>Ctrl+Shift+A</code>	Find Action	Find an action, bypassing menus. See Finding Actions .
<code>Ctrl+Shift+B</code>	Navigate to Type Declaration	Navigate to type declaration of a variable or a method call at caret.
<code>Ctrl+Shift+E</code>	Navigate to Recently Changed File	Show the list of recently updated files .
<code>Ctrl+Shift+F</code>	Find in Path	Initiate text search in the specified path .
<code>Ctrl+Shift+H</code>	Method Hierarchy	Browse hierarchy for the selected class.
<code>Ctrl+Shift+J</code>	Join Lines	Concatenate selected lines into one or concatenate a line where the caret is currently located with the next line.
<code>Ctrl+Shift+N</code>	Navigate to File	Jump to the specified file in project.
<code>Ctrl+Shift+R</code>	Replace in Path	Initiate text replacement in the specified path .
<code>Navigate to Test/Test Subject</code>	Navigate to a test for the class at caret, if any, or create a new test class. For a test class, navigate to its test subject.	
<code>Ctrl+Shift+U</code>	Toggle Case	Toggle case of the selected text fragment.
<code>Ctrl+Shift+V</code>	Paste from History	Paste from recent Clipboards. See page Cutting, Copying and Pasting
<code>Ctrl+Shift+W</code>	Deselect Word at Caret	Remove sequential selection made by the Select Word at Caret action.
<code>Ctrl+Shift+Z</code>	Redo	Redo the last Undo operation .

Ctrl+Shift+Navigation keys

ShortcutFunctionUse this shortcut to...

<code>Ctrl+Shift+End</code>	Move to Text End with Selection	Select text from the current caret position to the end of text, and move caret to the end of text. See page Selecting Text in the Editor .
<code>Ctrl+Shift+Home</code>	Move to Text Start with Selection	Select text from the current caret position to the start of text, and move caret to the start of text. See page Selecting Text in the Editor .
<code>Ctrl+Shift+Right</code>	Move to Word End with Selection	Select text from the current caret position to the end of word, and move caret to the end of word. See page Selecting Text in the Editor .
<code>Ctrl+Shift+Left</code>	Move to Word Start with Selection	Select text from the current caret position to the beginning of the current word, and move caret to the beginning of this word. See page Selecting Text in the Editor .
<code>Ctrl+Shift+V</code>	Paste from History	Paste from recent Clipboards. See page Cutting, Copying and Pasting
<code>Ctrl+Shift+Space</code>	SmartType Code Completion	Complete code, filtering the lookup list based on an expected type.
<code>Ctrl+Shift+Page Down</code>	Navigate to Page Bottom with Selection	Move the caret down to the page bottom selecting the text. See page Selecting Text in the Editor .
<code>Ctrl+Shift+Page Up</code>	Navigate to Page Top with Selection	Move the caret up to the page bottom selecting the text. See page Selecting Text in the Editor .
<code>Ctrl+Shift+Down</code>	Move Line Down	Move line at caret one down, preserving syntactical correctness. See page Adding, Deleting and Moving Code Elements .
<code>Ctrl+Shift+Up</code>	Move Line Up	Move line at caret up, preserving syntactical correctness. See page Adding, Deleting and Moving Code Elements .
<code>Ctrl+Shift+Backspace</code>	Last Edit Location	Jump to the place of the last editing.

Ctrl+Shift+Symbol keys

ShortcutFunctionUse this shortcut to...

<code>Ctrl+Shift+Open Bracket</code>	Move to Code Block Start with Selection	Move the caret to the beginning of the current code block, selecting the code from the initial caret location. See page Selecting Text in the Editor .
<code>Ctrl+Shift+Close Bracket</code>	Move to Code Block End with Selection	Move the caret to the end of the current code block, selecting the code from the initial caret location. See page Selecting Text in the Editor .

Ctrl+Shift+Slash	Comment with Block Comment	Comment/uncomment code with block comments. See page Commenting and Uncommenting Blocks of Code .
Ctrl+NumPad Plus	Expand All	Expand all folding blocks. See page Code Folding .
Ctrl+Shift+NumPad -	Collapse All	Collapse all folding blocks. See page Code Folding .

Ctrl+Shift+Numpad keys

ShortcutFunctionUse this shortcut to...

Ctrl+Shift+Numpad/	Comment with Block Comment	Comment/uncomment code with block comments. See page Commenting and Uncommenting Blocks of Code .
Ctrl+Shift+Numpad+	Expand All	Expand all folding blocks. See page Code Folding .
Ctrl+NumPad Plus		
Ctrl+Shift+Numpad-	Collapse All	Collapse all folding blocks. See page Code Folding .

Ctrl+Shift+Function (F) keys

ShortcutFunctionUse this shortcut to...

Ctrl+Shift+F4	Close Active Tab	Close an active tab in a tool window. See page PyCharm Editor .
Ctrl+Shift+F7	Highlight Usages in File / Highlight Method Exit Points	Highlight usages of a symbol where the caret is currently located. If the caret is placed on one of the method's exit points, like <code>return</code> , all method exit points are highlighted.
Ctrl+Shift+F8	View Breakpoints	View/manage all breakpoints/watchpoints

Function Keys

This section describes default mappings for the function (F) keys.

ShortcutFunction Use this shortcut to...

F1	Help	Invoke reference page.
F2	Activate in-place editing	In a GUI Designer form, enable in-place editing of the name of a selected UI component.
F3	Search for next/previous occurrence	Navigate to the next/previous occurrence of a selected word in the editor.
Shift+F3		
F4	Edit Source	Depending on the context: <ul style="list-style-type: none">– In Tool Windows: Open an Editor tab or tabs for the selected item or items (including GUI forms), and give focus to the last opened file.– On the context menus of the modules in the Project tool window, Dependency Viewer, and Module Dependencies tool window: open the Modules structure.
F5	Copy	Create a copy of a selected class/file/directory in the same or a different package.
F6	Move	Move a selected class/package/static member to another package/class and correct all references.
F7	Step Into	Step to the next executed line (during debugging).
F8	Step Over	Step to the next line in the current file (during debugging).
F9	Resume Program	Resume program execution (during debugging).
F11	Toggle Bookmark	Turn anonymous bookmark on or off.
F12	Jump to Last Window	Activate a last focused tool window.

Insert, Delete and Navigation Keys

This section describes default mappings for the `Insert`, `Delete` and the navigation keys.

ShortcutFunction Use this shortcut to...

<code>Delete</code>	Delete	Depending on the context: <ul style="list-style-type: none">- In the editor: delete selected symbol/block.- In the Find tool window: exclude items from the search results.- In the Version Control tool window: delete an item from a changelist.- In other views: remove the selected item or items.
<code>Down</code>	Move down	Move the caret one line down.
<code>End</code>	Move to Line End	Move the caret to the end of line.
<code>Home</code>	Move to Line Start	Move the caret to the beginning of line.
<code>Insert</code>	Toggle Insert/Overwrite	Toggle Insert/Overwrite modes in the editor. The shape of the cursor changes according to the current mode.
<code>Left</code>	Move left	Move the caret one character to the left.
<code>Page Down</code>	Page down	Move the caret one page up.
<code>Page Up</code>	Page up	Move the caret one page up.
<code>Right</code>	Move right	Move the caret one character to the right.
<code>Tab</code>		In the editor: <ul style="list-style-type: none">- With any selection: indent selected line(s).- Without any selection: insert a tab symbol (or corresponding number of space.characters). In a lookup list: <ul style="list-style-type: none">- No code after the caret in the editor: select an item (like <code>Enter</code>)- Some code after the caret in the editor: select an item and replace the code after the caret with it.
<code>Up</code>	Move up	Move the caret one line up.

Shift

This section lists and describes the keyboard shortcuts that include the **Shift** key:

- [Shift+Navigation keys](#)
- [Shift+Function \(F\) keys](#)

Shift+Navigation keys

ShortcutFunctionUse this shortcut to...

Shift+Down	Down with Selection	Move the caret one line down selecting the text.
Shift+End	Move to Line End with Selection	Move the caret to the end of line, selecting text.
Shift+Enter	Start New Line	Start a new line after the current one, positioning the caret in accordance with the current indentation level (equal to sequential pressing End, Enter).
Shift+Escape	Hide Active Window	Hide the currently active tool window.
Shift+Home	Move to Line Start with Selection	Move the caret to the beginning of line, selecting the text.
Shift+Left	Left with Selection	Move the caret one character to the left selecting the text.
Shift+Page Down	Page Down with Selection	Move the caret one page down selecting the text.
Shift+Page Up	Page Up with Selection	Move the caret one page up selecting the text.
Shift+Right	Right with Selection	Move the caret one character to the right selecting the text.
Shift+Tab	Unindent Selection	Move selected block to the previous indent level.
Shift+Up	Up with Selection	Move the caret one line up selecting the text.

Shift+Function (F) keys

ShortcutFunctionUse this shortcut to...

Shift+F1	External Documentation	Open browser with the documentation for the selected item. Refer to Viewing Inline Documentation for details.
Shift+F2	One of the following: <ul style="list-style-type: none">- Navigate to Previous Highlighted Error.- Stop Program.	Depending on whether you are editing or debugging: <ul style="list-style-type: none">- When editing: Navigate to the previous found error/warning.- When debugging: Terminate the debugging session.
F3 / Shift+F3	Search for next/previous occurrence	Jump to the next/previous occurrence of the selected word in the editor.
Shift+F6	Rename	Rename a statement and correct all references. (Refactoring).
Shift+F7	Move to Previous Difference/Smart Step Into	Move to a previous difference in a view./ Select the method to step in, if the current line contains multiple method call expressions. (Debugger).
Shift+F8	Step Out	Step to the first executed line after returning from a current method.
Shift+F9	Debug	Debug application.
Shift+F10	Run	Run application.
Shift+F11	Show Bookmarks	Open Bookmarks dialog to manage existing bookmarks and navigate between them.
Shift+F12	Restore Default layout	Restore the default PyCharm layout (tool windows positions, buttons location and order). To restore the default layout, check the option Store Current Layout as Default in the Window menu.

Ctrl+Alt+Shift

This section lists and describes the keyboard shortcuts that include the `Ctrl+Shift+Alt` keys.

Shortcut Function Use this shortcut to...

<code>Ctrl+Shift+Alt+C</code>	Copy Relative Path	Copy a reference (a relative path) of a symbol to the Clipboard.
<code>Ctrl+Shift+Alt+N</code>	Go to Symbol	Navigate to a symbol with the specified name.
<code>Ctrl+Shift+Alt+H</code>	Pop up Hector	Open the Highlighting level pop-up window.
<code>Ctrl+Shift+Alt+U</code>	Show Uml Diagram	Open UML Class diagram for a class or package.
<code>Ctrl+Shift+Alt+V</code>	Paste Simple	Paste the last entry from the Clipboard as plain text.
<code>Ctrl+Shift+Alt+L</code>	Show Reformat File Dialog	Show reformatting dialog .
<code>Ctrl+Shift+Alt+I</code>	Run Inspection by Name	Execute an inspection by its name .
<code>Ctrl+Shift+Alt+Insert</code>	New Scratch File	Create a new scratch file with the selected language.

Tool Windows Reference

View | Tool Windows

The Tool Windows Reference contains detailed information about the functionality, controls and menus of the PyCharm [tool windows](#).

In this section:

- [Project](#) ([Alt+1](#))
- [Favorites](#) ([Alt+2](#))
- [Find](#) ([Alt+3](#))
- [Run](#) ([Alt+4](#))
- [Debug](#) ([Alt+5](#))
- [TODO](#) ([Alt+6](#))
- [Structure](#) ([Alt+7](#))
- [Hierarchy](#) ([Alt+8](#))
- [Dependency Viewer](#)
- [Inspection Results](#)
- [Version Control](#)

Application Servers

View | Tool Windows | Application Servers

For this tool window to be available, there must be a server [run/debug configuration](#) in your [project](#), or a cloud user account must be registered in PyCharm.

All the available functions are accessed by means of the toolbar icons and context menu commands.

- [Icons and commands for server run configurations](#)
- [Icons and commands for server artifacts](#)
- [Icons and commands for cloud user accounts](#)
- [Icons and commands for cloud apps](#)
- [Deployment status icons](#)

Icons and commands for server run configurations

IconCommandDescription

	Run/Connect	Start the selected run/debug configuration in the run mode. For a local configuration, normally, the corresponding server will be started. For a remote configuration, PyCharm will connect to the server.
	Debug	Start the selected run/debug configuration in the debug mode.
	Stop/Disconnect	Stop the selected run/debug configuration. For a local configuration, normally, the corresponding server will be stopped. For a remote configuration, PyCharm will disconnect from the server.
	Deploy All	Deploy all the artifacts associated with the selected run/debug configuration.
	Edit Configuration	Edit the settings for the selected run/debug configuration.
	Artifacts	Edit the deployment list for the selected run/debug configuration.

Icons and commands for server artifacts

IconCommandDescription

	(Re)deploy	Deploy or redeploy the selected artifact.
	Undeploy	Undeploy the selected artifact.
	Remove	Remove the selected artifact from the corresponding deployment list and undeploy the artifact from the server.

Icons and commands for cloud user accounts

IconCommandDescription

	Connect	Connect (log on) to the corresponding cloud platform.
	Disconnect	Disconnect (log off) from the corresponding cloud platform.
	Edit Configuration	Edit your cloud user account settings.
	Deploy	Deploy your app by means of a cloud deployment run/debug configuration.
	Debug	Deploy your app and start debugging it by means of a cloud deployment run/debug configuration.

Icons and commands for cloud apps

IconCommandDescription

	(Re)deploy	Deploy or redeploy the selected app.
	Undeploy	Undeploy the selected app.
	Debug	Start debugging the selected application.
	Edit Configuration	Edit the settings for an associated cloud deployment run/debug configuration.

Deployment status icons

IconStatus

	Unknown
	Deployed
	Undeployed

Coverage Tool Window

View | Tool Windows | Coverage

In this section:

- [Toolbar](#)
- [Context menu](#)

Toolbar

ItemDescription

	Click this button to go up one level.
	When this button is pressed, all the packages are displayed as a single-level view.
	When this button is pressed, source code of the class selected in the tool window, automatically opens in a separate editor tab, and gains the focus.
	When this button is pressed, when source code of certain class gets the focus in the editor, the corresponding node is automatically highlighted in the tool window.
	<p>Click this button to generate a code coverage report and save it to the specified directory. See Generating Code Coverage Report for details.</p> <p>The button is not available when the tests are executed on Karma because a coverage report is actually generated on the disk every time Karma tests are run. The format of a coverage report can be configured in the configuration file, for example:</p>
<pre>// karma.conf.js module.exports = function(config) { config.set({ ... // optionally, configure the reporter coverageReporter: { type : 'html', dir : 'coverage/' } ... });};</pre>	
<p>The following <code>type</code> values are acceptable:</p> <ul style="list-style-type: none">- <code>html</code> produces a bunch of HTML files with annotated source code- <code>lcovonly</code> produces an <code>lcov.info</code> file- <code>lcov</code> produces HTML + <code>.lcov</code> files. This format is applied by default.- <code>cobertura</code> produces a <code>cobertura-coverage.xml</code> file for easy Hudson integration- <code>text-summary</code> produces a compact text summary of coverage, typically to the console- <code>text</code> produces a detailed text table with coverage for all files	
	Click this button to close the tool window.
	Click this button to show reference.

Context menu

ItemShortcutDescription

Jump to Source		Choose this command to open the selected file in the editor.
----------------	---	--

Database Tool Window

View | Tool Windows | Database

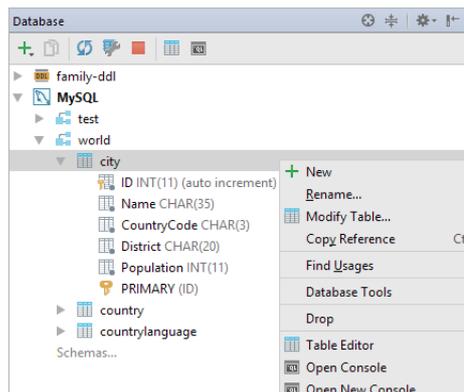
On this page:

- [Overview of the tool window](#)
- [Toolbar icons, context menu commands and shortcuts](#)
- [View options](#)
- [Icons for data sources and their elements](#)

See also, [Working with the Database tool window](#).

Overview of the tool window

The Database tool window provides access to functions for working with databases and DDL [data sources](#). It lets you view and modify data structures in your databases, and perform other associated tasks.



The available data sources are shown as a tree of data sources, schemas, tables and columns. If no data sources are currently defined, use the [New](#) command ([Alt+Insert](#)) to create a data source.

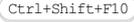
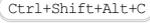
Most of the functions in this window are accessed by means of the toolbar icons or context menu commands. (If the toolbar is not currently shown, click  on the title bar and select Show Toolbar.) Many of the commands have keyboard shortcuts. If the toolbar is hidden, the [Synchronize](#) and [Open Console](#) commands can be accessed by means of the title bar icons ( and  respectively).

 on the title bar provides access to view options.

Toolbar icons, context menu commands and shortcuts

IconCommandShortcutDescriptionAvailable for

Icon	Command	Shortcut	Description	Available for
			Collapse all the nodes.	All node types
	New	Alt+Insert	Create a new data source, database, schema, database console, table, column, index, or a primary or foreign key. The list of options depends on which element is currently selected. See also, Creating a data source , Creating a database or schema , Creating and opening a new database console , Creating a table, a column, an index, or a primary or foreign key and Data Sources and Drivers Dialog .	DB data sources and their elements. If a DDL data source is selected, you can only choose to create another data source.
	Duplicate	Ctrl+D	Create a copy of the selected data source. Specify the properties of the data source in the Data Sources and Drivers dialog that opens.	DB and DDL data source nodes
	Synchronize	Ctrl+Alt+Y	Update the view of the selected element (i.e. synchronize the view of the element with its actual state in the database). See also, Auto sync .	DB data sources and their elements
	Properties		Open the Data Sources and Drivers dialog to manage your data sources and their settings.	All node types
	Disconnect	Ctrl+F2	Close the database connection for the selected DB data source or data sources. (The names of the data sources with active database connections are shown in bold.)	DB data sources with active connections and their elements
	Table Editor	F4	Open the selected database table in the Table Editor . See also, Working with the Table Editor .	Tables and table columns in DB data sources
	View Editor,	F4	Open the definition of the selected view, function,	Corresponding elements in

	Routine Editor,		procedure or package in the editor.	DB data sources
	Package Editor			
	Edit Source		Open the associated DDL file in the editor.	Tables and table columns in DDL data sources
	Open Console		Open the default database console for the corresponding DB data source.	DB data sources and their elements (tables and table columns)
	Rename		Rename the selected data source, table or column. Specify the new name in the dialog that opens. See also, Renaming items .	All node types
	Modify Table, Modify Column, Modify Index, Modify Key, Modify Foreign Key		Edit the definition of the selected table, column, index, or primary or foreign key. See also, Modifying the definition of a table, column, index, or a primary or foreign key .	Corresponding elements in DB data sources
	Copy Reference		Copy the fully qualified name of the selected data source, table or column to the clipboard.	All node types
	Find Usages		Find the usages of (references to) the selected item (data source, table or column) in your source files and libraries.	All node types
	Database Tools Hide Schemas		Hide the selected schemas. See Showing and hiding schemas .	Schemas in DB data sources
	Database Tools Manage Shown Schemas		Open the Schemas popup for the current DB data source. See Showing and hiding schemas .	DB data sources and their elements
	Database Tools Forget Cached Schema		Use this command in problematic cases such as when your data structures start to display incorrectly, fail to synchronize, etc. As a result, PyCharm deletes the information it has accumulated about your database. To check if this has eliminated the problem, use the Synchronize command.	DB data sources
	Database Tools Copy Settings		Copy the settings for the selected data source onto the clipboard.	DB data sources
	Database Tools Drop Primary Key		Remove the primary key constraint for the current table.	Tables and columns in DB data sources
	Database Tools Drop Foreign Key		Remove the foreign key constraint.	Columns with the foreign key constraint in DB data sources
	Database Tools Truncate		Remove all the rows in the selected table.	Tables in DB data sources
	Drop or Remove		Remove the selected item.	All node types
	Open New Console		Create and open a new database console for the corresponding DB data source.	DB data sources and their elements
	Generate and Copy DDL		Generate DDL definitions for the selected data source, schema, table, view, stored procedure or function, and copy those definitions onto the clipboard.	All node types except columns
	Open DDL in Console		Open a DDL definition of the selected table or view in a database console .	Tables and views in DB data sources
	Compare		Select two data sources, schemas or tables and then use this command to compare table structures for the selected items. The comparison results are shown in the differences viewer .	DB and DDL data sources and tables
	Dump Data to File(s)		Save data for the selected tables and views in files. Select the output format (e.g. SQL Inserts, Tab-separated (TSV), JSON-Closure.json.clj). See also, Saving data in files in various forms and formats .	DB data sources, and schemas, tables and views within them

Dump Data with "mysqldump" or Dump with "pg_dump"	Run mysqldump or pg_dump for the selected items. See Creating database backups with mysqldump or pg_dump .	MySQL and PostgreSQL data sources, and schemas, tables and views within them
Import Data from File	Import a text file containing delimiter-separated values (CSV, TSV, etc.) into your database. If a schema is currently selected, PyCharm will create a new table for the data that you are importing. If a table is selected, PyCharm will try to add the data to the selected table. See Importing delimiter-separated values into a database .	Schemas, tables and columns in DB data sources. For columns, the result will be the same as for schemas
Color Settings	Set or change the color for the selected element or elements. (The Database Color Settings dialog will open.)	All node types
Scripted Extensions / Generate POJOs.cj	Generate a Java entity class for the selected table. In the dialog that opens, specify the directory in which the <code>.java</code> class file should be generated.	Tables
Scripted Extensions / Go to Scripts Directory	Switch to the directory where the <code>Generate POJOs.cj</code> example script file is located. See also, Extending the functionality of database tools .	All node types
Diagrams	<code>Ctrl+Shift+Alt+U</code> <code>Ctrl+Alt+U</code> View a UML class diagram for the selected data source or table. Select: <ul style="list-style-type: none"> Show Visualisation to open the diagram on a separate editor tab. Show Visualisation Popup to see the diagram in a pop-up window. 	DB and DDL data sources and tables
View Quick Documentation (in the main menu)	<code>Ctrl+Q</code> View basic information about the selected element. For example, the info about a table includes the names of the data source, database, schema and the table itself, the table definition (<code>CREATE TABLE</code>) and, if appropriate, the first 10 rows. To close the documentation pop-up, press <code>Escape</code> .	All node types

View options

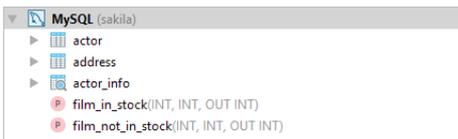
The view options, generally, define what is shown in the tool window and how. To view or change these options, click  on the title bar.

OptionDescription

Group Schema This option defines how schema elements are shown. When on, there are separate nodes for tables, views and stored routines (shown as folders). Tables, views and routines (procedures and functions) are shown as elements of the corresponding groups.



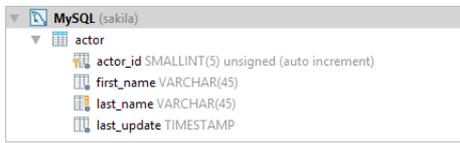
When off, there is no explicit grouping for tables, views, and routines. Tables and views are followed by procedures and functions.



Group Contents This option defines how table elements are shown. When on, there are separate nodes for columns, indexes, primary and foreign key constraints, and triggers (shown as folders). The elements appear in the corresponding groups.

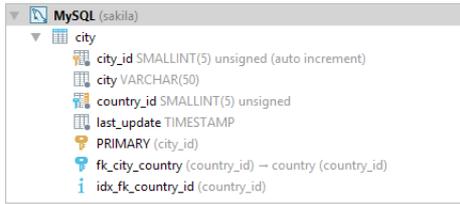


When off, there is no such grouping and, generally, only columns are shown for tables.

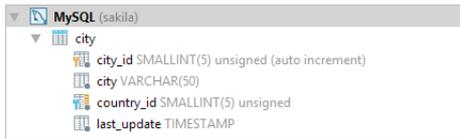


Show Keys and etc.

When this option is on, the primary and foreign key constraints, and indexes are shown as separate elements.



Otherwise, there are no separate elements for the keys and indexes.



The option is unavailable when the [Group Contents](#) option is on.

Show Empty Groups

If the [Group Schema](#) or the [Group Contents](#) option is on, you can select to show or hide empty groups, i.e. the categories that contain no elements. The Show Empty Groups option is on:

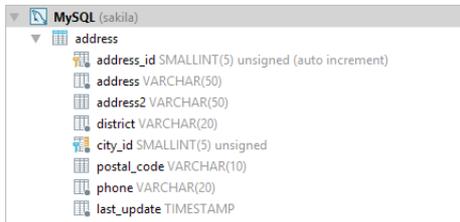


The Show Empty Groups option is off:

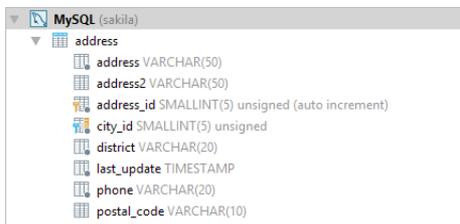


Sort Alphabetically

When this option is off, columns, generally, are unsorted.



When this option is on, the columns are ordered alphabetically.



Show Toolbar

Select or deselect this option to show or hide the toolbar.

The rest of the options are common for all the tool windows, see [Viewing Modes](#).

Icons for data sources and their elements

IconDescription

	DB data source. Also, DBMS-specific icons are used:
	DB2
	Derby
	H2

 [HSQLDB](#)

 [MySQL](#)

 [Oracle](#)

 [PostgreSQL](#)

 [SQL Server](#)

 [SQLite](#)

 [Sybase](#)

	DB data source with the read-only status, e.g.  for Derby.
	DDL data source
	Database
	Schema
	Table
	View
	Column
	A <code>NOT NULL</code> column
	Column with a primary key
	Column with a foreign key
	Column with an index
	Primary key
	Foreign key
	Index
	Trigger
	Stored procedure or function

Data Sources and Drivers Dialog

To access this dialog from the [Database tool window](#):  or  on the toolbar

Use this dialog to manage your [data sources](#) and database drivers, and their settings.

A driver here is understood as a collection that includes [database driver](#) files, and also default options and settings for creating a DB data source.

The names of new, yet unsaved items are shown in the left-hand pane in green. New items are saved when clicking Apply or OK.

- [Left-hand pane](#)
- [DB data source settings](#)
- [DDL data source settings](#)
- [Driver settings](#)
- [Problems](#)

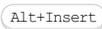
See also, [Managing Data Sources](#).

Left-hand pane

Shown in this pane are your data sources and [drivers](#). When you select an item, its settings are shown in the right-hand part of the dialog.

Use the toolbar icons, context menu commands and associated keyboard shortcuts to manage your data sources and drivers, and also to perform other, related tasks.

Icon and ShortcutDescription command

		Use this icon, command or shortcut to create a new data source or driver. Select: <ul style="list-style-type: none">- The name of DBMS to create a DB data source associated with the corresponding DBMS. If you have created your own drivers, you can also select a driver to use it as the basis for creating the data source. See DB data source settings.- DDL Data Source to create a DDL data source. See DDL data source settings.- Database Driver to create a driver. See Driver settings.
		Use this icon, command or shortcut to remove the selected item or items from the list.
		Use this icon, command or shortcut to create a copy of the selected data source or driver.
		Use this icon, command or shortcut to view or edit the settings for the driver associated with the selected DB data source.
		Use this icon or command to move the selected DB data source to the global or project level. (Global data sources are available in all your projects.)
Change Driver		If more than one driver is available for the selected DB data source, use this command to associate the data source with a different driver.
Reset Changes		Use this command or shortcut to undo the changes made to the selected item.
Load Sources		For the selected DB data source or data sources, PyCharm will load source code of database objects for the category of schemas that you select. The alternative way of setting this option - for each data source individually - is by using Load sources for on the Options tab .
Show Driver Usages		Use this command or shortcut to find the DB data sources that use the selected driver. The found data sources are shown in the Used By popup which lets you navigate to anyone of the found data sources.
		Use these icons to switch between the items you've been working with.

DB data source settings

- [Name](#)
- [General tab](#)
- [SSH/SSL tab](#)
- [Schemas tab](#)
- [Options tab](#)
- [Advanced tab](#)

Name

ItemDescription

Name Use this field to edit the name of the data source.

General tab

Shown on this tab are mainly the database connection settings.

The user interface is adjustable: the set of available controls depends on which option is selected in the list to the right of the [URL field](#).

ItemDescription

File	<p>If your database is a local file, specify the path to that file.</p> <p> lets you select an existing database file.</p> <p> lets you create a new database file.</p>
Path	<p>If your database is a local file or folder, specify the path to that file or folder.</p> <p> lets you select the database file or folder.</p> <p>Create database. Select this option to create a new database. (This option may be unavailable.)</p>
Host	<p>Specify the hostname (domain name) or the IP address of the computer on which the database is located. If the database is on your local computer, specify <code>localhost</code> or <code>127.0.0.1</code>.</p> <p>If you are using SSH, the database host must be accessible by the specified domain name or IP address from the computer on which the SSH proxy runs. See SSH/SSL tab.</p>
Port	<p>Specify the database port number.</p>
Database	<p>Specify the name of the target database or schema.</p>
User	<p>Specify the name of the database user (i.e. your database user account name).</p>
Password	<p>Specify the password for the database user.</p>
Remember password	<p>Select this check box if you want PyCharm to remember the password. See Passwords.</p>
URL	<p>Shown in this field is the URL that PyCharm will use to connect to the database. The user interface for specifying the URL is different depending on which option is selected in the list to the right:</p> <ul style="list-style-type: none">– URL only. This option, generally, is for editing the database connection URL directly. When you select this option, only the following fields are available: User, Password and URL. <p>You should edit the URL right in the field. Your user name and password, if necessary, are specified in the corresponding fields, or within the URL in the format appropriate for the JDBC driver that you are using.</p> <ul style="list-style-type: none">– When using any other of the options (the options are DBMS-specific), PyCharm forms the database connection URL automatically using the info in the fields above the URL field. In all such cases, normally, you don't need to edit the URL (though you can if you want).
Test Connection	<p>Click this button to make sure that the database connection settings are correct and PyCharm can communicate with the target database.</p>
Driver	<p>Click the <driver name> link to switch to the settings for the associated driver.</p>
Read-only	<p>Select this check box if you want to protect the data source from accidental data modifications. As a result, you won't be able to modify the data in the Table Editor.</p> <p>Whether data modifications will be possible by means of the consoles depends on the DBMS: PyCharm will try to set the database connection status to read-only. All the rest depends on the database driver, i.e. whether and to which extent the driver supports the read-only status.</p> <p>For example, if you define your MySQL data source as read-only, the driver won't let you switch schemas in associated database consoles (see e.g. Selecting the default schema or database). In that case, you should use the Database field to specify your current schema or database.</p>
Auto commit	<p>The default setting for the Auto-commit option in the table editor and the database console.</p>
Auto sync	<p>If this option is off, the view of the data source in the Database tool window is synchronized with the actual state of the database only when you perform the Synchronize command ( or Ctrl+Alt+Y).</p> <p>If this option is on, the view of the data source is automatically updated:</p> <ul style="list-style-type: none">– When you change the data source settings. (Technically, when you click OK in the Data Sources and Drivers dialog.)– When you run DDL SQL statements in the database consoles associated with the data source. <p>Note that auto sync is performed for the overall database and, thus, may be time-consuming. So auto-synchronization is more suitable for "small" databases. If your database is "big", it's recommended that you sync its state manually (e.g. Ctrl+Alt+Y), and only for the appropriate database parts such as separate tables.</p>

SSH/SSL tab

If the target database should be accessed using SSH or SSL, select the corresponding check box and specify the associated settings.

– [SSH](#)

– [SSL](#)

ItemDescription

SSH	
Use SSH tunnel	<p>Select this check box to set up and use an SSH tunnel for accessing a remote database via an SSH proxy.</p>
Copy from	<p>If there is already a data source for which the necessary SSH settings are specified, you can copy those settings from that data source. Click the link and select the data source to copy the settings from.</p>
Proxy host	<p>Specify the hostname (domain name) or IP address of the SSH proxy server that you are using. The SSH proxy server host must be accessible by the specified hostname or IP address from your local computer.</p>
Port	<p>Specify the port on which your SSH proxy server accepts SSH connections. The port number <code>22</code> suggested by PyCharm is the standard port used by SSH</p>

servers. Change this number if your SSH proxy server uses a different port.

Proxy user	Specify the name of the SSH proxy user.
Auth type	Specify the user authentication type used by your SSH proxy. Select: <ul style="list-style-type: none">– Password for password-based authentication.– Key pair (Open SSH) for key-based authentication.
Proxy password	For password-based authentication: specify the password for the SSH proxy user. See also, Remember password .
Private key file	For key-based authentication: specify the path to the file where the corresponding private key is stored. Type the path in the field, or click  (Shift+Enter) and select the file in the dialog that opens .
Passphrase	For key-based authentication: specify the passphrase for the private key if the key is locked with the passphrase.
Remember password	Select this check box if you want PyCharm to remember the password or the passphrase. See Passwords .
SSL	All the files specified in this section should be in PEM format. Which of the files you have to specify, depends on the SSL properties of your user account.
Use SSL	Select the check box to use SSL when connecting to the server.
Copy from	If there is already a data source for which the necessary SSL settings are specified, you can copy those settings from that data source. Click the link and select the data source to copy the settings from.
CA file	Specify the path to SSL Certificate Authority (CA) certificate file. If used, this must be the same certificate as used by the server. Type the path in the field, or click  (Shift+Enter) and select the file in the dialog that opens .
Client certificate file	Specify the path to your (client) public key certificate file. Type the path in the field, or click  (Shift+Enter) and select the file in the dialog that opens .
Client key file	Specify the path to your (client) private key file. Type the path in the field, or click  (Shift+Enter) and select the file in the dialog that opens .

Schemas tab

Select the databases and [schemas](#) to be shown in the Database tool window.

ItemDescription

	Refresh the list of the databases and schemas.
 test	Specify the text for filtering the list. Only the databases and schemas whose names contain the specified text will be shown.
Pattern	An alternative to selecting the databases and schemas by means of the check boxes. <code>*.*</code> would mean all schemas in all databases. To get the info about the syntax to be used, place the cursor into the field and press  .

Options tab

The settings on this tab relate to filtering database objects, loading source code, etc.

ItemDescription

Object filter	You can limit the set of tables and other database objects shown in the Database tool window by specifying a filter. The filter is applied to "short" (i.e. unqualified) names of the database objects. Examples: <code>f.*</code> Only the objects whose names start with <code>f</code> will be shown. <code>table:[gh].*</code> The tables whose names start with <code>g</code> or <code>h</code> and all the objects in other categories will be shown. <code>view:new_.* routine:-[ps].*</code> The views whose names start with <code>new_</code> , the routines whose names start with the letters other than <code>p</code> or <code>s</code> , and all the objects in the categories other than views and routines will be shown.
Plan table	For Oracle: The name of the table that should be used to store the EXPLAIN PLAN output information.
Introspect using JDBC metadata	You may want to select this check box (if available) to try to fix the problems with retrieving the database structure information from your database (e.g. when the schemas existing in your database or the database objects below the schema level are not shown in the Database tool window). This option defines which of the following alternative introspectors PyCharm is using to retrieve the info about the database objects (DB metadata): <ul style="list-style-type: none">– A native introspector (may be unavailable for certain DBMSs). This introspector uses DBMS-specific tables and views as a source of metadata. In addition to "standard" info, it can retrieve DBMS-specific details and thus produce a more precise picture of your database objects.– A JDBC-based introspector (available for all the DBMSs). This introspector uses the metadata provided by the corresponding JDBC driver. It can retrieve only standard info about the database objects and their properties. <p>The JDBC-based introspector should be used only when the native introspector fails (if this is the case, select the check box) or is not available (in such a case the check box is missing).</p> <p>(The native introspector may fail, for example, when your DB server version is older than the minimum version supported by PyCharm, when you are using Amazon Redshift because it "pretends" that it's a Postgres while in fact it isn't, etc.)</p>
Load sources for	PyCharm will load source code of database objects for the selected category of schemas. You can change this setting for several data sources at once. To do that, select the data sources of interest in the left-hand pane of the dialog. Then, in the context menu , point to Load Sources and select the necessary option.

Advanced tab

On this tab, you can configure the database connection properties, and also specify the options and environment variables for the database driver JVM.

ItemDescription

Name - Value	<p>The set of connection options passed to the database driver as key - value pairs at its start.</p> <p>When you select a cell in the Name column, the description of the corresponding option is shown underneath the table.</p> <p>To find an option of interest, just start typing its name.</p> <p>To start editing a value, click or double-click the corresponding Value field, or press F2.</p> <p>To add a row, start editing the values in the last row, where <user defined> and <value> are shown. A new row will be added to the table automatically.</p>
VM Options	<p>The options for the JVM in which the database driver runs. (The driver is started as a separate process in its own JVM.)</p> <p>Example. For certain Oracle Database versions (e.g. for version 9), there may be connection problems when you and your database server are in different time zones. Specifying the time offset for your timezone may help, e.g. <code>-Duser.timezone=UTC+03:00</code>.</p> <p>Alternatively, you can try setting the variable <code>oracle.jdbc.timezoneAsRegion</code> to <code>false</code> in the Name - Value table.</p>
VM Environment	<p>Environment variables for the database driver JVM.</p> <p>Example. Sometimes, when working with Oracle, your data and/or error messages don't display correctly. Many of such problems are encoding-related and can be solved by appropriately setting the <code>NLS_LANG</code> variable, e.g. <code>NLS_LANG=Russian_CIS.CL8MSWIN1251</code>. For more information, see e.g. Oracle NLS_LANG FAQ.</p> <p>To start editing the variables, click .</p>

For additional information, refer to your DBMS documentation.

DDL data source settings

A DDL data source is defined by its name, and can reference one or more DDL files and/or another data source (a parent data source).

ItemDescription

Name	Use this field to edit the data source name.
DDL Files	<p>Use the controls in this area to compose the list of files that contain the necessary DDL definitions.</p> <ul style="list-style-type: none">-  (Alt+Insert). Use this icon or shortcut to add a DDL SQL file or files to the data source definition. In the dialog that opens, select the necessary file or files.-  (Alt+Delete). Use this icon or shortcut to remove the selected file or files from the list.-  (Alt+Up). Use this icon or shortcut to move the selected file one line up in the list.-  (Alt+Down). Use this icon or shortcut to move the selected file one line down in the list.
Extend	<p>If necessary, select another data source as a parent. As a result, the data source whose properties you are editing will "inherit" all the DDL definitions from its parent.</p> <p>If the parent data source is not needed, select <none>.</p>

Driver settings

- [Name](#)
- [Settings tab](#)
- [Advanced tab](#)

Name

ItemDescription

Name The name of the [driver](#).

Settings tab

Shown on this tab are mainly the defaults for the [General tab](#).

ItemDescription

Class	The fully qualified name of the driver class to be used.
Dialect	<p>The SQL dialect associated with the corresponding data sources. Via the data sources this dialect will "propagate" to the database console.</p> <p>In addition to particular dialects, also the following option is available:</p> <ul style="list-style-type: none">- <Generic SQL>. Basic SQL92-based support is provided including completion and highlighting for SQL keywords, and table and column names. Syntax error highlighting is not available. So all the statements in the input pane are always shown as syntactically correct.
Auto commit	The default setting for the auto commit option .
Auto sync	The default setting for the auto sync option .
JDBC drivers	The JDBC driver to be used to interact with a database. You can download and use a driver from the PyCharm driver repository (see Use provided driver) or specify the driver that you already have available on your computer (see Additional).

Use provided driver. If the check box is selected, the driver from the repository is used.

To download and use the latest driver version, click the red <driver name> [latest] link. (If the link isn't red, the latest driver version has already been downloaded.)

You can also specify that you want to use the latest available driver or the driver with a particular version number. To do that, right-click the link, point to the driver name, and select Latest or the version number. If the selected version has not been yet downloaded, it will be downloaded automatically.

Additional. The files specified in this area are used in addition to the downloaded driver if the Use provided driver check box is selected, or instead of the downloaded driver otherwise.

Say, you want to use the driver that is already available on your computer. In that case, you should clear the Use provided driver check box, click **+** and select the driver files (usually one or more `.jar` files) in the [dialog that opens](#).

URL templates

The templates used to construct the database URL. The text in curly brackets represents variables, e.g.

- `{host}` the domain name or IP address of the database host.
- `{port}` the database port number.
- `{database}` the name of the database or schema.

Optional fragments are in square brackets, e.g. `[:{port}]`.

Template names correspond to the names of the options in the [URL options list](#).

Advanced tab

Shown on this tab are the default settings for the [Advanced tab](#).

ItemDescription

Name - Value	The default set of connection options passed to the database driver as key - value pairs at its start. To start editing a value, double-click the corresponding Value field.
--------------	---

To add a row, start editing the values in the last row, where <user defined> and <value> are shown. A new row will be added to the table automatically.

VM Options	The default options for the JVM in which the database driver runs. (The driver is started as a separate process in its own JVM.)
------------	--

Problems

If potential problems are detected, there is a number to the right of Problems. In that case, if you click Problems, you'll see the list of problems as well as controls for fixing them.

Database Color Settings Dialog

From the [Database tool window](#):

Select the element or elements of interest, and select Color Settings from the context menu.

Select the [color](#), and specify how this color should be used (see [Shared](#) and [Override recursively](#)). Use the check boxes in the [Appearance Settings](#) section to enable or disable the database colors in various places in the UI.

ItemDescription

Color Click the color that you want to use for the element or elements selected in the Database tool window. If the color that you want is missing, click Custom and specify the color in the dialog that opens.

The color which is currently selected is marked with its name, e.g. Blue.



If you don't want to use any color, click [No Color](#).

Shared Select the check box if you want to share the change of color you are about to make with your team. (The database color settings are shared through version control.)

In technical terms, the new color (for the selected element or elements) will be stored in

- `.idea/databaseColors.xml` or the `.ipr` file if the check box is selected.
- `.idea/workspace.xml` or the `.iws` file if the check box is not selected.

The `workspace.xml` or the `.iws` file, normally, is not shared through version control while the rest of PyCharm configuration files are.

For more information on PyCharm configuration files and their sharing, see [Project](#).

Override recursively If the check box is selected, the selected color will be applied to the element itself and all its subordinate elements recursively. If the check box is not selected, the colors set individually for the subordinate elements won't change.

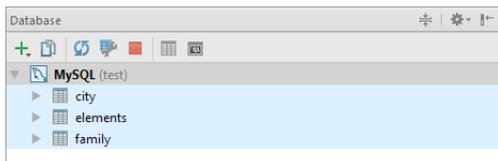
Appearance Settings

Enable database colors Clear the check box to (temporarily) disable the database colors everywhere in the UI.

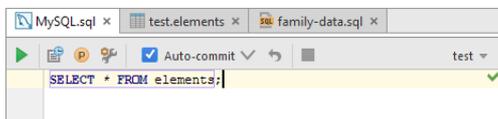
In Database tool window Select the check box to use the database colors in the Database tool window. When this option is off, the tool window looks like this:



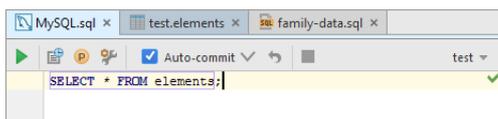
When this option is on, the tool window may look as shown below. (In this example, the blue color is set for the corresponding data source.)



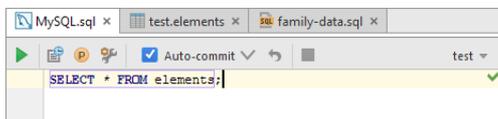
In editor tabs Select the check box to use the database colors for editor tabs (when working with the table editor and the input pane of a database console). When this option is off, the tabs look like this:



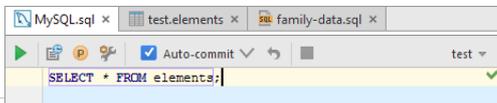
When this option is on, the tabs have the same color as the corresponding data source or table (e.g. blue):



In console editors and grids Select the check box to use the database colors for the editing area in database consoles and table editors. When this option is off, the editing area looks like this:

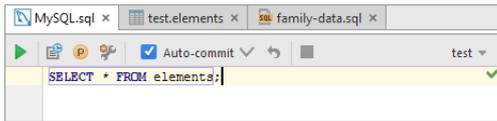


When this option is on, the background of the editing area has the same color as the corresponding data source (e.g. blue).

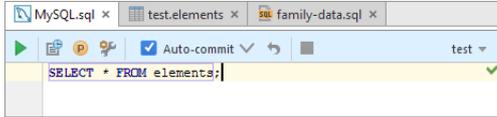


In toolbars

Select the check box to use the database colors for the toolbars of database consoles and table editors.
When this option is off, the toolbar looks like this:



When this option is on, the toolbar background has the same color as the corresponding data source (e.g. blue).



No Color

Click this button, if you don't want to use any color for the selected element or elements.

Database Console

From the [Database tool window](#) (for any node within a DB data source):

-  on the title bar if the toolbar is hidden
-  on the toolbar if the toolbar is shown
- Open Console or Open New Console from the context menu
- `Ctrl+Shift+F10`

From the [Scratches view](#) of the [Project tool window](#):

- View | Jump to Source
- Jump to Source from the context menu
- `F4`

On this page:

- [Overview](#)
- [Input pane](#)
- [Toolbar of the Database Console tool window](#)
- [Output pane](#)
- [Result pane](#)
- [Parameters pane](#)

See also, [Working with Database Consoles](#).

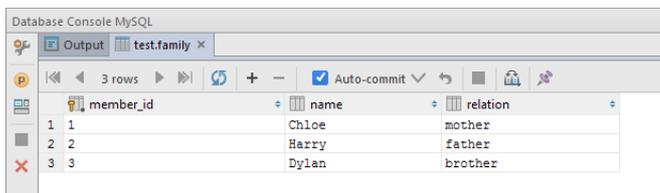
Overview

Database consoles let you compose and execute SQL statements for databases defined in PyCharm as data sources. They also let you analyze and modify the retrieved data.

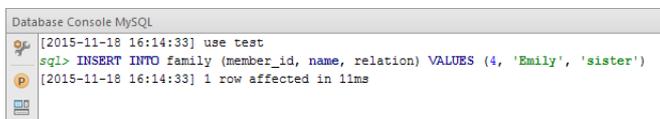
The input pane of a database console opens as a separate editor tab. This is where you compose your SQL statements.



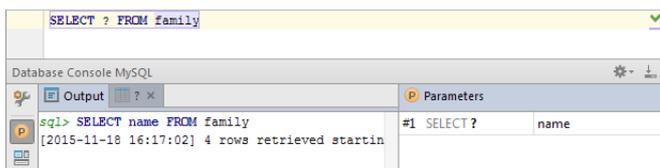
When you execute your first statement () , the Database Console tool window opens. If the executed statement retrieves data (e.g. `SELECT`), there are two panes in the tool window shown on the Output and the Result tabs. (The tab showing retrieved data may be labeled Result # or, if appropriate, the table name may be shown.)



Otherwise, only the output pane is shown.



Additionally, you can open the Parameters pane () to manage parameters in SQL statements.



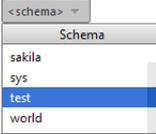
Input pane

Use the input pane to compose and execute your SQL statements as well as to perform other, associated tasks.

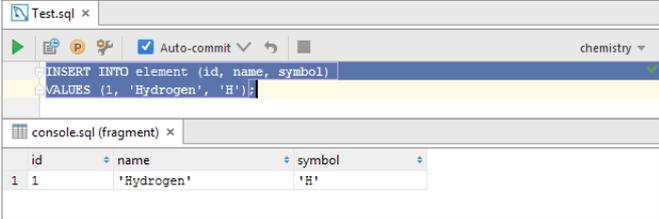
The available functions are accessed by means of the toolbar icons, keyboard shortcuts and context menu commands.

- [Toolbar icons and shortcuts](#)
- [Most useful context menu commands](#)

Toolbar icons and shortcuts

 Execute	Ctrl+Enter	Use this icon or shortcut to execute the selected (highlighted) SQL statement or statements. If nothing is selected, the current statement is executed. See also, Run 'console.sql' , Execute in Console and Executing an SQL statement .
 Browse Console History	Ctrl+Alt+E	Use this icon or shortcut to open a dialog that shows all the statements that you have run for the corresponding data source. See also, Executing auto-memorized statements .
 View Parameters		Use this icon to open or close the Parameters pane .
 Settings		Use this icon to open the Database page of the Settings dialog to view or edit the settings for your database consoles, the Table Editor and the Database tool window.
Auto-commit		Use this check box to turn the autocommit mode for the database connection on or off. In the autocommit mode, each SQL statement is executed in its own transaction that is implicitly committed. Consequently, the SQL statements executed in this mode cannot be rolled back. If the autocommit mode is off, transactions are committed or rolled back explicitly by means of the commit or rollback command. Each commit or rollback starts a new transaction which provides grouping for a series of subsequent SQL statements. In this case, the data manipulations in the transaction scope are committed or rolled back all at once when the transaction is committed or rolled back.
 Commit		If the autocommit mode is off, use this icon to commit the current transaction.
 Rollback		If the autocommit mode is off, use this icon to roll back the current transaction.
 Cancel Running Statements	Ctrl+F2	Use this icon or shortcut to terminate execution of the current statement or statements.
<schema>		Select the default schema or database, or, for PostgreSQL, form the schema search path. (This control may be unavailable).  See also, Selecting the default schema or database and Controlling the schema search path for PostgreSQL .

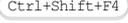
Most useful context menu commands

Edit as Table		If an <code>INSERT</code> statement is currently selected: Open the editor for working with the data in table format.  See also, Editing data for INSERT statements in table format .
Change Dialect (<CurrentDialect>)		Use this command to change the SQL dialect being used. Select the necessary dialect from the list. In addition to particular dialects, also the following option is available: – <Generic SQL>. Basic SQL92-based support is provided including completion and highlighting for SQL keywords, and table and column names. Syntax error highlighting is not available. So all the statements in the input pane are always shown as syntactically correct. See also, Changing the SQL dialect .
Explain Plan		Show an execution plan (a.k.a. explain plan) for the current statement. The result is shown in a mixed tree/table format on a dedicated Plan tab.
Explain Plan (Raw)		Show an execution plan (a.k.a. explain plan) for the current statement. The result is shown in table format. (Technically, <code>EXPLAIN <CURRENT_STATEMENT></code> or similar statement is executed.)
Execute	Ctrl+Enter	Execute the current statement or the sequence of selected statements.
Execute to File		Execute the current statement (e.g. <code>SELECT</code>) and save the result in a text file. Select the output format, and specify the file location and name.
Run 'console.sql'	Ctrl+Shift+F10	Use this command or shortcut to execute all the statements contained in the console.
Diagrams	Ctrl+Shift+Alt+U Ctrl+Alt+U	If the cursor is within the name of a schema: Open a UML class diagram for the schema. – Show Visualisation (<code>Ctrl+Shift+Alt+U</code>). The diagram opens on a separate editor tab. – Show Visualisation Popup (<code>Ctrl+Alt+U</code>). The diagram opens in a pop-up window.

Toolbar of the Database Console tool window

To hide or show the toolbar, click  on the title bar and select Show Toolbar.

ItemShortcutDescription

 Settings		Use this icon to open the Database page of the Settings dialog to view or edit the settings for your database consoles, the Table Editor and the Database tool window.
 Enable SYS.DBMS_OUTPUT		For Oracle: use this icon or shortcut to enable or disable showing the contents of the DBMS_OUTPUT buffer in the output pane.
 View Parameters		Use this icon to open or close the Parameters pane .
 Browse Console History		Use this icon or shortcut to open a dialog that shows all the statements that you have run for the corresponding data source. See also, Executing auto-memorized statements .
 Restore Layout		Use this icon to restore the original tool window layout (after the rearrangements that you have made).
 Cancel Running Statements		Use this icon or shortcut to terminate execution of the current statement or statements.
 Close		Use this icon or shortcut to close the tool window.

Output pane

This pane shows the SQL statements that you have run as well as information about other operations performed in the console. These include turning the autocommit mode on or off, committing or rolling back a transaction, etc.

The information about the errors that occur is also shown in this pane.

For most of the events the following information is provided:

- Timestamp, that is, when the event took place.
- For data definition and data manipulation operations - how many rows were affected (e.g. added, changed or deleted). For data retrieval operations - how many rows were retrieved.
- Duration in milliseconds.

The summary info is also shown on the status bar.

Use the following context menu commands:

- Copy () to copy the text selected in the output pane to the clipboard.
- Compare with Clipboard to compare the text selected in the output pane with the contents of the clipboard.
- Clear All to clear all the contents of the output pane.

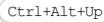
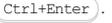
Result pane

This pane shows the data retrieved from the database in table format. You can sort, add, edit and remove the data as well as perform other, associated tasks.

- [Main functions](#)
- [Using the header row](#)

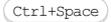
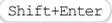
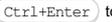
Main functions. Most of the functions in the Result pane are accessed by means of controls on the toolbar, context menu commands for the data cells, and associated keyboard shortcuts.

ItemShortcutDescription

		These icons and corresponding commands are for switching between the result set pages, i.e. the pages that show the retrieved data. A fixed number of rows shown simultaneously is referred to as a result set page . If this number is less than the number of rows that satisfy the query, only a subset of all the rows is shown at a time. In such cases, you can use  to switch between the subsets. (If all the rows are currently shown, these icons and the corresponding commands are inactive.) The result set page size is set on the Database page of the Settings dialog.
 First Page		Use this icon or command to switch to the first of the result set pages to see the first series of rows.
 Previous Page		Use this icon, command or shortcut to switch to the previous result set page to see the previous series of rows.
 Next Page		Use this icon, command or shortcut to switch to the next result set page to see the next series of rows.
 Last Page		Use this icon or command to switch to the last of the result set pages to see the last series of rows.
 Reload Page		Use this icon, command or shortcut to refresh the current table view. Use this function: <ul style="list-style-type: none"> - Synchronizē the data shown with the actual contents of the database. - Apply the Result set page size setting after its change.
 Add New Row		Use this icon, command or shortcut to add a new row to the table. Complete entering a value into a cell by pressing  . To save the new row, select Submit New Row from the context menu or press  .

If inappropriate in the current context (i.e. for the table currently shown), this function is not available.

See also, [Adding a row](#).

		Use this icon, command or shortcut to delete the selected row or rows. Rows are selected by clicking the cells in the column where the row numbers are shown. To select more than one row, use mouse clicks in combination with the  key. If inappropriate in the current context (i.e. for the table currently shown), this function is not available.
		Submit the changes to the database server. See Submitting and reverting changes .
		Revert the selected changes. See Submitting and reverting changes .
		Use this icon or shortcut to terminate execution of the current query.
		Use this icon or command to pin the tab to the tool window to keep the query result. See also, Pinning the Result tab .
 Tab S... (TSV) Data Extractor: <current_format>		Use this button or command to open a menu in which you can select an output format for your data. In addition to output formats, there are also the following options and commands: <ul style="list-style-type: none">– Allow Transposition. For delimiter-separated values formats (TSV, CSV): If the table is shown transposed and you are copying selected cells or rows to the clipboard (e.g. ) , the selection is copied transposed (as shown) if the option is on and non-transposed (as in the original table) otherwise.– Skip Generated Columns (SQL). For SQL INSERTs and UPDATES: When copying or saving data (Copy, Dump Data To File, Dump Data To Clipboard), don't include auto-increment fields.– Add Table Definition (SQL). For SQL INSERTs and UPDATES: When copying or saving data, add the table definition (CREATE TABLE).– Configure CSV Formats. Open the CSV Formats Dialog that lets you manage your delimiter-separated values formats (e.g. CSV, TSV).– Go to Scripts Directory. Switch to the directory where the scripts that convert table data into various output formats are stored.
		Use this command to copy the table data onto the clipboard.
		Use this command to save the table data in a file. In the dialog that opens, specify the location and name of the file.
		Export the data to another table, schema or database. Select the target schema (a new table will be created) or table (the data will be added to the selected table). In the dialog that opens, specify the data mapping info and the settings for the target table.
View Query		Use this button to view the query which was used to generate the table. To close the pane where the query is shown, press  .
		This icon provides access to the following commands: <ul style="list-style-type: none">– Transpose. Turn the transposed table view on or off. (In the transposed view, the rows and columns are interchanged. So, the rows are shown as columns and vice versa.)– Reset View. Restore the initial table view after reordering or hiding the columns, or sorting the data.– Auto-commit. Turn the autocommit mode on or off. When the autocommit mode is on, each change of a value, or adding or deleting a row - when submitted to the database server - is implicitly committed and cannot be rolled back. When the autocommit mode is off, all the changes you have submitted to the server can be explicitly committed or rolled back by means of the Commit or the Rollback command.– Settings. Open the Database page of the Settings dialog to view or edit the settings for the Database console, the Table Editor and the Database tool window.
Edit		Use this command or shortcut to start editing a value in the selected cell or cells. (Alternatively, you can double-click the cell or simply start typing.) To open the value completion suggestion list, press  . To enter the modified value, press  . To cancel editing, press  . See also, Modifying cell contents and Modifying values in a number of cells at once .
Edit Maximized		Maximize the selected cell and start editing a value in it. When working in a maximized cell, use  to start a new line and  to enter the value. To restore an initial value and quit the editing mode, press  . See also, Modifying cell contents .
Set DEFAULT		If appropriate: Set the default value or values.
Set NULL		If appropriate: Replace the value or values with <code>null</code> .
Load File		If appropriate: Load a file into the field.
Clone Row		Use this command or shortcut to create a copy of the selected row.
Quick Documentation		Use this command or shortcut to open the quick documentation view. To close the view, press  . For more information, see Using the quick documentation view .
Transpose		Turn the transposed table view on or off. Alternatively, use  Transpose.
Commit		Commit the current transaction. See also, Auto-commit .
Rollback		Roll back the current transaction. See also, Auto-commit .

Go To Row	Ctrl+G	Use this command or shortcut to switch to a specified row. In the dialog that opens, specify the row number to go to.
Go To Related Data	F4	Use this command or shortcut to switch to a related record. The command options are a combination of those for Go To Referenced Data and Go To Referencing Data . The command is not available if there are no related records.
Go To Referenced Data	Ctrl+B	Use this command or shortcut to switch to a record that the current record references. If more than one record is referenced, select the target record in the pop-up that appears. The command is not available if there are no referenced records.
Go To Referencing Data	Alt+F7	Use this command or shortcut to see the records that reference the current record. In the pop-up that appears there are two categories for the target records: <ul style="list-style-type: none"> – First Referencing Row. All the rows in the corresponding table will be shown and the first of the rows that references the current row will be selected. – All Referencing Rows. Only the rows that reference the current row will be shown. The command is not available if there are no records that reference the current one.
Copy	Ctrl+C	Copy the selection onto the clipboard. See also, Copying and pasting data: data types are converted if necessary .
Paste	Ctrl+V	Paste the contents of the clipboard into the table. See also, Copying and pasting data: data types are converted if necessary .
Save LOB		Use this command to save the large object (LOB) currently selected in the table in a file.
	Alt+J, Shift+Alt+J, Ctrl+W	See Selecting cells and ranges: using unobvious techniques .

Using the header row. In the Result pane, you can use the cells in the header row (i.e. the row where column names are shown) for:

- [Sorting data](#)
- [Reordering columns](#)
- [Hiding and showing columns](#)

You can sort table data by any of the columns by clicking the cells in the header row.

Each cell in this row has a sorting marker in the right-hand part and, initially, a cell may look something like this: . The sorting marker in this case indicates that the data is not sorted by this column.

If you click the cell once, the data is sorted by the corresponding column in the ascending order. This is indicated by the sorting marker appearance: . The number to the right of the marker (1 on the picture) is the sorting level. (You can sort by more than one column. In such cases, different columns will have different sorting levels.)

When you click the cell for the second time, the data is sorted in the descending order. Here is how the sorting marker indicates this order: .

Finally, when you click the cell for the third time, the initial state is resorted. That is, sorting by the corresponding column is canceled: .

Here is an example of a table where data are sorted by two of its columns.

	id	name	relation
1	3	Dylan	brother
2	6	Jack	brother
3	1	Harry	father
4	2	Chloe	mother
5	5	Alice	sister
6	4	Emily	sister

To restore the initial "unsorted" state for the table, click  and select Reset View.

To reorder columns, use drag-and-drop for the corresponding cells in the header row. To restore the initial order of columns, click  and select Reset View.

	member_id	relation	name
1	5	sister	Alice
2	1	mother	Chloe
3	3	brother	Dylan
4	4	sister	Emily
5	2	father	Harry

To hide a column, right-click the corresponding header cell and select Hide column.

To show a hidden column:

1. Do one of the following:

- Right-click any of the cells in the header row and select Column List.

- Press **Ctrl+F12**.

In the list that appears, the names of hidden columns are shown struck through.

	member_id	name
1	5	Alice
2	1	Chloe
3	3	Dylan
4	4	Emily
5	2	Harry
6	6	Jack

2. Select (highlight) the column name of interest and press `Space`.

3. Press `Enter` or `Escape` to close the list.

To show all the columns, click  and select Reset View.

See also, [Using the Structure view to sort data, and hide and show columns.](#)

Parameters pane

The Parameters pane shows the parameters detected in the input pane and lets you edit their values. To open or close this pane, use  on the toolbar.

To start editing a value, switch to the corresponding table cell and start typing. To indicate that you have finished editing a value, press `Enter` or switch to a different cell. To quit the editing mode and restore an initial value, press `Escape`.

When you select a row in the table, the corresponding parameter is highlighted in the input pane.

See also, [Executing parameterized statements.](#)

Table Editor

From the [Database tool window](#) (for any table within a DB data source):

-  on the toolbar (if the toolbar is not currently hidden)
- Table Editor from the context menu
- 

Use the Table Editor to manipulate the table data, and to perform other, associated tasks.

There are two tabs in the Table Editor:

- Data. On this tab, the table you are working with is shown.
- DDL. Initially, the `CREATE TABLE` statement used to create the table is shown on this tab. You can edit the statement and then run it  on the toolbar or , Use  or  to regenerate the `CREATE TABLE` statement for the current state of the table in the database.

Below, the features of the Data tab are discussed.

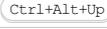
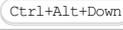
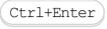
- [Toolbar controls, context menu commands for data cells and keyboard shortcuts](#)
- [Using the table header row: sorting data, reordering and hiding columns](#)

See also, [Working with the Table Editor](#).

Toolbar controls, context menu commands for data cells and keyboard shortcuts

Most of the functions on the Data tab are accessed by means of controls on the toolbar, context menu commands for the data cells, and associated keyboard shortcuts.

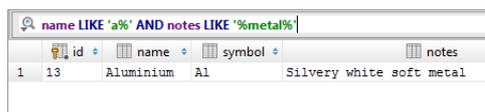
ItemShortcutDescription

		These icons and corresponding commands are for switching between the result set pages, i.e. the pages that show the table data. A fixed number of rows shown simultaneously is referred to as a result set page . If this number is less than the number of rows in the table, only a subset of all the rows is shown at a time. In such cases, you can use  ,  ,  and  to switch between the subsets. (If all the rows are currently shown, these icons and the corresponding commands are inactive.) The result set page size is set on the Database page of the Settings dialog.
	First Page	Use this icon or command to switch to the first of the result set pages to see the first series of rows.
	Previous Page 	Use this icon, command or shortcut to switch to the previous result set page to see the previous series of rows.
	Next Page 	Use this icon, command or shortcut to switch to the next result set page to see the next series of rows.
	Last Page	Use this icon or command to switch to the last of the result set pages to see the last series of rows.
	Reload Page 	Use this icon, command or shortcut to refresh the current table view. Use this function to: <ul style="list-style-type: none">- Synchronize the data shown with the actual contents of the database.- Apply the Result set page size setting after its change.
	Add New Row 	Use this icon, command or shortcut to add a new row to the table. Complete entering a value into a cell by pressing  . To save the new row, select Submit New Row from the context menu or press  .
	Delete Rows 	Use this icon, command or shortcut to delete the selected row or rows. Rows are selected by clicking the cells in the column where the row numbers are shown. To select more than one row, use mouse clicks in combination with the  key.
	Submit 	Submit the changes to the database server. See Submitting and reverting changes .
	Revert 	Revert the selected changes. See Submitting and reverting changes .
	Cancel Query 	Use this icon or shortcut to terminate execution of the current query.
	Data Extractor: <current_format>	Use this button or command to open a menu in which you can select an output format for your data. In addition to output formats, there are also the following options and commands: <ul style="list-style-type: none">- Allow Transposition. For delimiter-separated values formats (TSV, CSV): If the table is shown transposed and you are copying selected cells or rows to the clipboard (e.g. ), the selection is copied transposed (as shown) if the option is on and non-transposed (as in the original table) otherwise.- Skip Generated Columns (SQL). For SQL INSERTs and UPDATES: When copying or saving data (Copy, Dump Data To File, Dump Data To Clipboard), don't include auto-increment fields.- Add Table Definition (SQL). For SQL INSERTs and UPDATES: When copying or saving data, add the table definition (CREATE TABLE).- Configure CSV Formats. Open the CSV Formats Dialog that lets you manage your delimiter-separated values formats (e.g. CSV, TSV).- Go to Scripts Directory. Switch to the directory where the scripts that convert table data into various output formats are stored.

 Dump Data To Clipboard	Use this command to copy the table data onto the clipboard.
 Dump Data To File	Use this command to save the table data in a file. In the dialog that opens, specify the location and name of the file.
 Export to Database	Export the data to another table, schema or database. Select the target schema (a new table will be created) or table (the data will be added to the selected table). In the dialog that opens, specify the data mapping info and the settings for the target table.
View Query	Use this button to view the query which was used to generate the current table view. To close the pane where the query is shown, press <code>Escape</code> .
	<p>This icon provides access to the following commands:</p> <ul style="list-style-type: none"> – Transpose. Turn the transposed table view on or off. (In the transposed view, the rows and columns are interchanged. So, the rows are shown as columns and vice versa.) – Reset View. Restore the initial table view after reordering or hiding the columns, or sorting the data. – Sort via ORDER BY. Turn the corresponding option on or off. If the Sort via ORDER BY option is on, all the sorting operations that you perform are reflected in the corresponding <code>SELECT</code> statement (an <code>ORDER BY</code> clause is added or modified) which is executed immediately. As a result, the data for the whole table is sorted by the corresponding database system. <p>Don't turn this option on if you want to keep interactions with the database to a minimum (e.g. when the table is very big or the database connection is "slow").</p> <p>If this option is off, the data is sorted "locally" by PyCharm and only for the rows currently shown. – Row Filter. Show or hide the filter box. – Auto-commit. Turn the autocommit mode on or off. When the autocommit mode is on, each change of a value, or adding or deleting a row - when submitted to the database server - is implicitly committed and cannot be rolled back. <p>When the autocommit mode is off, all the changes you have submitted to the server can be explicitly committed or rolled back by means of the Commit or the Rollback command.</p> – Settings. Open the Database page of the Settings dialog to view or edit the settings for the Database console, the Table Editor and the Database tool window. </p>

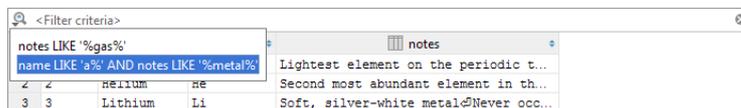
 <Filter criteria> 

Specify filtering conditions for the table. (If the filter box is not currently shown, click  on the toolbar and select Row Filter.) The filtering conditions are specified as in a `WHERE` clause but without the word `WHERE`, e.g. `name LIKE 'a%' AND notes LIKE '%metal%'`. Within the `LIKE` expressions, the SQL wildcards can be used: the percent sign (`%`) for zero or more characters and underscore (`_`) for a single character.



To apply the conditions currently specified in the box, press `Enter`. To cancel filtering, click , or delete the contents of the filter box and press `Enter`.

To reapply a memorized filter, click  and select the filter in the list. See also, [Filter history size](#).



Edit F2

Use this command or shortcut to start editing a value in the selected cell or cells. (Alternatively, you can double-click the cell or simply start typing.)

To open the value completion suggestion list, press `Ctrl+Space`. To enter the modified value, press `Enter`. To cancel editing, press `Escape`.

See also, [Modifying cell contents](#) and [Modifying values in a number of cells at once](#).

Edit Maximized Shift+Enter

Maximize the selected cell and start editing a value in it.

When working in a maximized cell, use `Enter` to start a new line and `Ctrl+Enter` to enter the value. To restore an initial value and quit the editing mode, press `Escape`.

See also, [Modifying cell contents](#).

Set DEFAULT Ctrl+Alt+D

If appropriate: Set the default value or values.

Set NULL Ctrl+Alt+N

If appropriate: Replace the value or values with `null`.

Load File

If appropriate: Load a file into the field.

Clone Row Ctrl+D

Use this command or shortcut to create a copy of the selected row.

Quick Documentation Ctrl+Q

Use this command or shortcut to open the quick documentation view. To close the view, press `Escape`. For more information, see [Using the quick documentation view](#).

Transpose

Turn the transposed table view on or off. Alternatively, use  | Transpose.

Commit

Commit the current transaction. See also, [Auto-commit](#).

Rollback

Roll back the current transaction. See also, [Auto-commit](#).

Go To | Row Ctrl+G

Use this command or shortcut to switch to a specified row. In the dialog that opens, specify the row number to go to.

Go To | Related Data F4

Use this command or shortcut to switch to a related record. The command options are a combination of those for [Go To |](#)

Referenced Data and Go To | Referencing Data.

The command is not available if there are no related records.

Go To Referenced Data	Ctrl+B	Use this command or shortcut to switch to a record that the current record references. If more than one record is referenced, select the target record in the pop-up that appears. The command is not available if there are no referenced records.
Go To Referencing Data	Alt+F7	Use this command or shortcut to see the records that reference the current record. In the pop-up that appears there are two categories for the target records: – First Referencing Row. All the rows in the corresponding table will be shown and the first of the rows that references the current row will be selected. – All Referencing Rows. Only the rows that reference the current row will be shown. The command is not available if there are no records that reference the current one.
Filter by		Use this command to access quick filtering options. The options include those for the current column name and depend on the value in the current cell.
Copy	Ctrl+C	Copy the selection onto the clipboard. See also, Copying and pasting data: data types are converted if necessary.
Paste	Ctrl+V	Paste the contents of the clipboard into the table. See also, Copying and pasting data: data types are converted if necessary.
Save LOB		Use this command to save the large object (LOB) currently selected in the table in a file.
	Alt+J, Shift+Alt+J, Ctrl+W	See Selecting cells and ranges: using unobvious techniques.

Using the table header row: sorting data, reordering and hiding columns

Use the cells in the header row (i.e. the row where column names are shown) for:

- [Sorting data](#)
- [Reordering columns](#)
- [Hiding and showing columns](#)

You can sort table data by any of the columns by clicking the cells in the header row.

Each cell in this row has a sorting marker in the right-hand part and, initially, a cell may look something like this:  name . The sorting marker in this case indicates that the data is not sorted by this column.

If you click the cell once, the data is sorted by the corresponding column in the ascending order. This is indicated by the sorting marker appearance:  name  1. The number to the right of the marker (1 on the picture) is the sorting level. (You can sort by more than one column. In such cases, different columns will have different sorting levels.)

When you click the cell for the second time, the data is sorted in the descending order. Here is how the sorting marker indicates this order:  name  1.

Finally, when you click the cell for the third time, the initial state is restored. That is, sorting by the corresponding column is canceled:  name .

Here is an example of a table where data are sorted by two of its columns.

	id	name	relation
1	3	Dylan	brother
2	6	Jack	brother
3	1	Harry	father
4	2	Chloe	mother
5	5	Alice	sister
6	4	Emily	sister

To restore the initial "unsorted" state for the table, click  and select Reset View. See also, [Sort via ORDER BY.](#)

To reorder columns, use drag-and-drop for the corresponding cells in the header row. To restore the initial order of columns, click  and select Reset View.

	member_id	relation	name
1	5	sister	lice
2	1	mother	hloe
3	3	brother	ylan
4	4	sister	mily
5	2	father	arry

To hide a column, right-click the corresponding header cell and select Hide column.

To show a hidden column:

1. Do one of the following:
 - Right-click any of the cells in the header row and select Column List.
 - Press .

In the list that appears, the names of hidden columns are shown struck through.

	member_id	name
1	5	Alice
2	1	Chloe
3	3	Dylan
4	4	Emily
5	2	Harry
6	6	Jack

2. Select (highlight) the column name of interest and press `Space`.

3. Press `Enter` or `Escape` to close the list.

To show all the columns, click  and select Reset View.

See also, [Using the Structure view to sort data, and hide and show columns.](#)

Debug Tool Window

View | Tool Windows | Debug

Alt+5

In this topic:

- [Overview](#)
- [Debug toolbar](#)
- [Stepping toolbar](#)
- [Hide/restore toolbar](#)
- [Moving tabs and areas](#)
- [Context menu of a tab](#)

Overview

The Debug tool window becomes available when you start [debugging](#).

It displays the output generated by the debugging session for your application. If you are debugging multiple applications, the output for each application is displayed in a separate tab named after the corresponding [run/debug configuration](#).

For each application, there are the following nested tabs:

- [Console](#): displays system information and error messages, and the console input and output of your application.
- [Debugger](#) tab divided into the following areas:
 - [Frames/Threads](#)
 - [Variables](#)
 - [Watches](#)
- [Elements](#): appears if you are using Chrome browser for debugging.

Each area has a [context menu](#) that allows you to configure its behavior and navigate between tabs.

Each of the tabs and areas can be [hidden/restored](#), or [moved](#) to a location of your choice.

Debug toolbar

Item
Tooltip **Description**
and
Shortcut

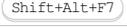
	Rerun	Click this button to stop the current application and run it again. When an application is stopped, this button toggles to  .
		
	Debug	When the current application is stopped, click this button to debug it again. When an application is running, this button toggles to  .
		
	Resume Program	When an application is paused, click this button to resume program execution.
		
	Pause Program	Click this button to pause program execution. Note that the button is not available for Run/Debug Configuration: Node JS , Run/Debug Configuration: Node JS Remote Debug , and Run/Debug Configuration: NodeUnit .
		
	Stop	Click this button to terminate the current process externally by means of the standard <code>shutdown</code> script. Clicking the button once invokes <code>soft kill</code> allowing the application to catch the <code>SIGINT</code> event and perform graceful termination (on Windows, the <code>Ctrl+C</code> event is emulated). After the button is clicked once, it is replaced with  indicating that subsequent click will lead to force termination of the application, e.g. on Unix <code>SIGKILL</code> is sent.
		
	View Breakpoints	Click this button to open the Breakpoints dialog box where you can configure breakpoints behavior.
		
	Mute Breakpoints	Use this button to toggle breakpoints status. When the button  is pressed in the toolbar of the Debug tool window, all the breakpoints in a project are muted, and their icons become grey:  .
		You can temporarily mute all the breakpoints in a project to execute the program without stopping at breakpoints.
	Restore Layout	Click this button abandon changes to the current layout and return to the default state.
	Settings	Click this button to open the menu with the following options available: <ul style="list-style-type: none">– Show Values Inline: select this option to enable the Inline Debugging feature that allows viewing the values of variables right next to their usage in the editor.– Sort Values Alphabetically: select this option to sort the values in the Variables pane in the alphabetical order.

- Unmute Breakpoints on Session Finish: select this option to re-enable all disabled breakpoints after the debugging session has been finished.
- Show Return Values: select this option to display the return values of the executed functions in the current frame while stepping.
- Simplified Variables View: Select this option to include the following names into the Special Variables group:
 - all variables with the dunder names.
 - all instances of `function`, `classobj` and `module`.
 - all the IPython internal variables, if debug console with IPython has been started.

	Pin	Click this button to pin or unpin the currently selected tab.
	Close	Click this button to close the selected tab.
		
	Help	Click this button to open the corresponding help page.
		

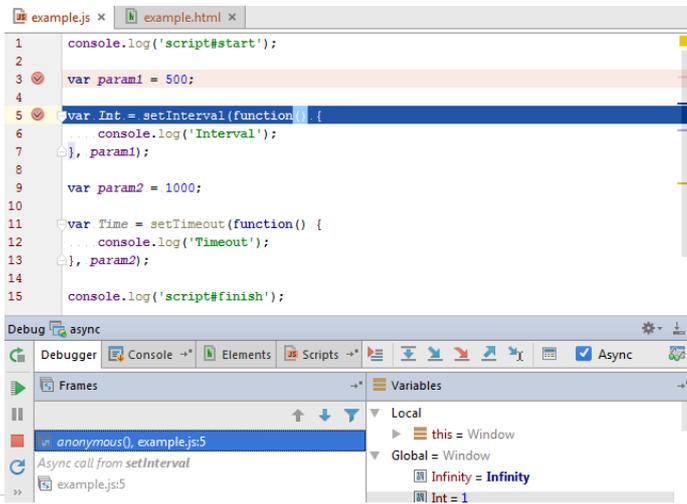
Stepping toolbar

ItemTooltip Description and Shortcut

	Show Execution Point	Click this button to highlight the current execution point in the editor and show the corresponding stack frame in the Frames pane.
		
	Step Over	Click this button to execute the program until the next line in the current method or file, skipping the methods referenced at the current execution point (if any). If the current line is the last one in the method, execution steps to the line executed right after this method.
		
	Step Into	Click this button to have the debugger step into the method called at the current execution point.
		
	Step Into My Code	Click this button to skip stepping into library sources and keep focused on your own code.
		
	Step Out	Click this button to have the debugger step out of the current method, to the line executed right after it.
		
	Drop frame	Interrupts execution and returns to the initial point of method execution. In the process, it drops the current method frames from the stack.
	Run to Cursor	Click this button to resume program execution and pause until the execution point reaches the line at the current cursor location in the editor. No breakpoint is required. Actually, there is a temporary breakpoint set for the current line at the caret, which is removed once program execution is paused. Thus, if the caret is positioned at the line which has already been executed, the program will be just resumed for further execution, because there is no way to roll back to previous breakpoints. This action is especially useful when you have stepped deep into the methods sequence and need to step out of several methods at once. If there are breakpoints set for the lines that should be executed before bringing you to the specified line, the debugger will pause at the first encountered breakpoint.
		
	Evaluate Expression	Click this button to open the Evaluate Expression dialog.
		

- Async
- When this check box is selected, PyCharm recognizes breakpoints inside asynchronous code and stops at them and lets you step into asynchronous code. As soon as a breakpoint inside an asynchronous function is hit or you step into asynchronous code, a new element `Async call from <caller>` is added in the Frames pane of the Debugger tab. PyCharm displays a full call stack, including the caller and the entire way to the beginning of the asynchronous actions.
 - When the check box is cleared, PyCharm does not recognize and therefore skips breakpoints in the asynchronous code and does not allow you to step into it.

The image below shows an example of a JavaScript debugging session.



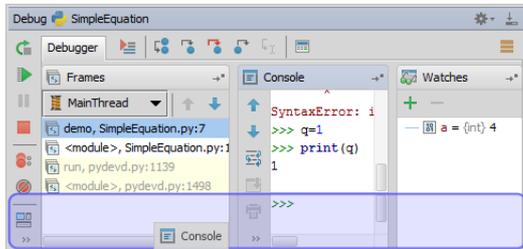
Hide/restore toolbar

IconTooltipDescription

	Hide	Click this button, located in the upper-right corner of the Debug Console, Watches, Treads, Frames, or Variables pane to hide the corresponding area. When an area is hidden, its icon appears in the right corner of the Debugger.
	Restore 'Console' view	Click this button to make the Console area visible. This button becomes available after clicking Hide . With the Async check box selected, the debugger will stop at line3 (breakpoint). On clicking Step into, the debugger will move to line9.
	Restore 'Frames' view	Click this button to make the Frames area visible. This button becomes available after clicking Hide .
	Restore 'Watches' view	Click this button to make the Watches area visible. This button becomes available after clicking Hide .
	Restore 'Threads' view	Click this button to make the Threads area visible. This button becomes available after clicking Hide .
	Restore 'Variables' view	Click this button to make the Variables area visible. This button becomes available after clicking Hide .

Moving tabs and areas

If you are unhappy with the default layout of the Debug tool window, you can always move the tabs and areas. To do that, just drag a tab or an area to the desired location. The possible target gets highlighted:



Drop the tab or area in the highlighted location.

To restore the default layout of tabs and area, click in the Debug toolbar.

Context menu of a tab

Use the context menu of the **Frames/Threads**, **Variables** or **Watches** areas to configure the behavior of these areas or navigate between tabs.

ItemDescription

Hide	Click this button to hide the corresponding area
Close Others	Click this button to hide all tabs except for the Console and Debugger tabs.
Focus On Startup	If this option is selected, the selected area gets the focus when you start a debugging session.
Focus On Breakpoint	If this option is selected, the selected area gets the focus when a breakpoint is reached.
Select Next Tab / Select Previous Tab	Use these options to switch between the Console and the Debugger tabs.

Debug Tool Window. Debugger

The Debugger tab is divided into the following areas:

- [Frames](#)
- [Variables](#)
- [Watches](#)

Debug Tool Window. Frames

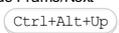
The Frames pane enables you to gain access to the list of threads of your application. To examine a thread, select it from the drop-down list on top of the pane.

The status and type of a thread is indicated by a special icon and a textual note next to the thread's name. For each thread, you can view the stack frame, examine frames, navigate between frames, and automatically jump to a frame's source code in the editor.

To examine the values stored in a frame, use the [Variables pane](#) of the Debug tool window.

Toolbar

Item Shortcut and Tooltip	Description
---------------------------------	-------------

	Previous Frame/Next Frame  / 	Use the arrow buttons to navigate through the frame stack.
	Hide Frames from Libraries	Click this button to hide frames from libraries. If this button is released, all frames are displayed.

In this topic:

- [Overview](#)
- [Toolbar](#)
- [Context menu](#)
- [Variable types](#)

Overview

The Variables pane enables you to examine the values stored in the objects of your application.

When a stack frame is selected in the [Frames pane](#), the Variables pane displays all data within its scope (method parameters, local and instance variables). In this pane, you can set labels for objects, inspect objects, evaluate expressions, add variables to watches and more.

Toolbar

This toolbar appears only when the [Watches pane](#) is hidden so the configured watches are displayed in the Variables pane. Hiding/showing the Watches pane is controlled through the  toggle button:

- When the button is pressed, which is its default status, the Watches pane is hidden and the toolbar is shown in the Variables pane. So doing, the focus is with the [Debugger tab](#).
- When the button is released, the toolbar moves to the Watches pane.

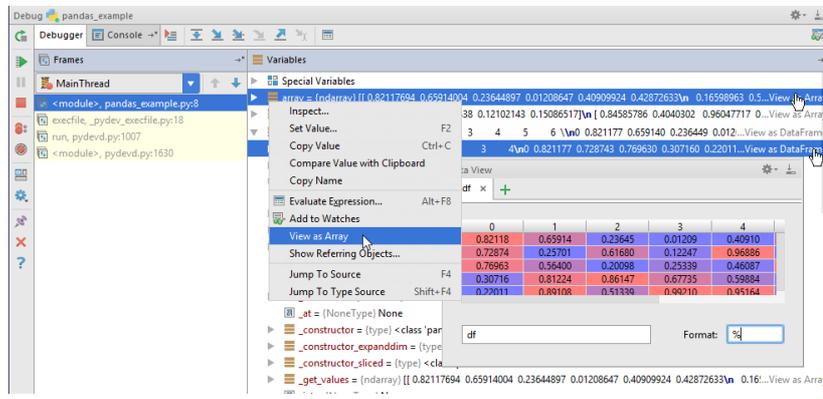
ItemShortcutDescription

		Click this button to create a new watch.
		Click this button to remove the selected watch from the list.
	 	Use these buttons to change the order of watches.
		Use this button to create a copy of the selected watch.
	Show watches in Variables tab	Use this toggle button to have the Watches pane hidden or shown. By default, the button is pressed and displayed on the toolbar of the Variables pane. Consequently, the Watches pane is hidden and the watches are shown in the Variables pane . <ul style="list-style-type: none"> - To have the Watches pane displayed separately and view the configured watches in it, release the Show watches in Variables tab toggle button. The Watches pane appears with the Show watches in Variables tab toggle button on the toolbar. - To hide the Watches pane and view the watches in the Variables pane, press the  toggle-button on the toolbar of the Watches pane. The toggle button returns to the default location on the toolbar of the Variables pane.

Context menu

ItemShortcutDescription

Inspect	N/A	This command is available for fields, local variables and reference expressions, and opens a non-modal Inspection window, where you can concentrate on a particular reference. You can open as many Inspection windows as required. The view in the Inspection window is the same as in the Watches pane , but requires less screen space.
Set Value		Use this command to change the runtime value of a field or a variable.
Copy Value		Use this command to copy the value of the selected variable to the Clipboard. If multiple items are selected, not only variables' values, but also their structure is copied, so that when you copy-paste the selection to a text file, the indentation mimics the tree output of the debugger to produce an easy-to-read output. Alternatively, hover your mouse cursor over a value and view its contents in the tooltip.
Copy JSON		This menu item is available only in the JavaScript context . Choose this command to copy the selected value in the JSON format .
Compare Value with Clipboard	N/A	Use this command to compare the selected value with the value currently in the Clipboard.
Copy Name	N/A	Use this command to copy the name of the selected variable to the Clipboard.
 Evaluate Expression		Use this command to evaluate the selected variable in the dialog that opens.
 Add to Watches	N/A	This command is available for all nodes except static ones. Use this command to create an expression that references the node and add this expression to the Watches pane .
Show Referring Objects	N/A	Use this command to display a list of objects referring to the currently selected variable.
Jump to Source		This command opens the source code of the selected variable or field in the editor and places the caret in the corresponding line.
Jump to Type Source		Use this command to navigate to the definition of the class of the selected variable or field.
View as Array		This command is available for variables that represent NumPy arrays. Note that NumPy must be downloaded and installed in your Python interpreter. With this package installed, you can get a graphical view of a NumPy array and its parts using slicing, formatting and coloring tools:



This command supports [pandas dataframes](#) as well.

Variable types

The icon on the left of each variable indicates its type:

- : static
- : global
- : field
- : array
- : primitive
- : object

In this topic:

- [Overview](#)
- [Toolbar](#)
- [Context menu](#)

Overview

By default, the Watches pane is hidden and the watches are shown in the [Variables pane](#).

- To have the Watches pane displayed separately and view the configured watches in it, release the Show watches in Variables tab toggle button  on the toolbar of the Variables pane. By default, the button is pressed.
- To hide the Watches pane and view the watches in the Variables pane, press the  toggle-button on the toolbar of the Watches pane.

In the Watches pane you can evaluate any number of variables or expressions in the context of the current stack frame. The values are updated with each step through the application, and become visible every time the application is suspended.

While the Evaluate Expression command on the context menu of the [Variables](#) pane enables you to see one expression at a time, the Watches pane shows multiple expressions that persist from one debug session to another, until you remove them.

You can create watches in this pane, in the [Variables](#) pane and even in the editor.

Watch expressions are always evaluated in the context of a stack frame that is currently inspected in the [Frames](#) pane. If an expression cannot be evaluated, it is displayed with a question mark.

Toolbar

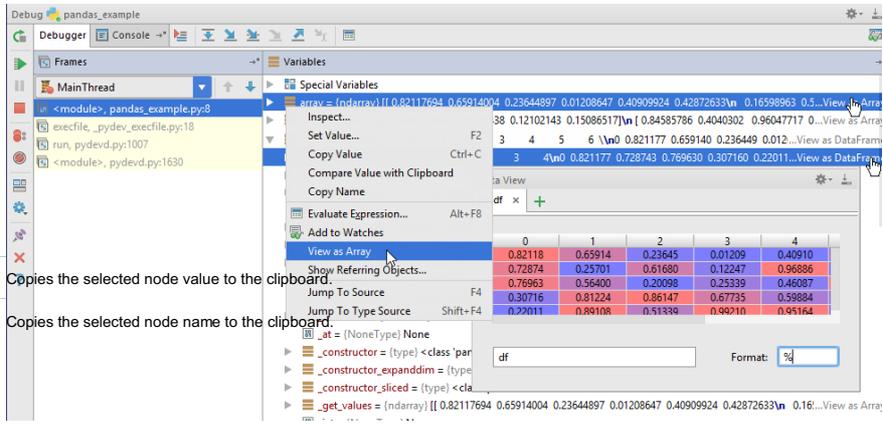
ItemShortcutDescription

		Click this button to create a new watch.
		Click this button to remove the selected watch from the list.
	 	Use these buttons to change the order of watches.
		Use this button to create a copy of the selected watch.
	Show watches in Variables tab	<p>Use this toggle button to have the Watches pane hidden or shown. By default, the button is pressed and displayed on the toolbar of the Variables pane. Consequently, the Watches pane is hidden and the watches are shown in the Variables pane.</p> <ul style="list-style-type: none"> – To have the Watches pane displayed separately and view the configured watches in it, release the Show watches in Variables tab toggle button. The Watches pane appears with the Show watches in Variables tab toggle button on the toolbar. – To hide the Watches pane and view the watches in the Variables pane, press the  toggle-button on the toolbar of the Watches pane. The toggle button returns to the default location on the toolbar of the Variables pane.

Context menu

ItemShortcutDescription

New Watch		Choose this command to create a new watch. A text field opens, where you can enter new watch expression.
Remove Watch		Choose this command to delete the currently selected watch expression from the list.
Edit		Choose this command to change the selected watch expression.
Remove All Watches		Choose this command to delete all watch expressions from the list.
Inspect		Available for fields, local variables and reference expressions. Choose this command to open the Inspect window for the node, which allows you to perform the same operations as those available in the stack frame, with the only difference that the root node is the one you have selected. You can recursively call the new Inspect windows from within each other. Each window is not modal and immediately reflects all changes in its subtree.
Show Referring Object		Choose this command to display the list of objects referring to the current watch.
Jump to Source		This command opens the source code of the selected variable or field in the editor and places the caret on a proper line.
View as Array		<p>This command is available for variables that represent NumPy arrays. Note that NumPy must be downloaded and installed in your Python interpreter.</p> <p>With this package installed, you can get a graphical view of a NumPy array and its parts using slicing, formatting and coloring tools:</p>



Copy Value Ctrl+C

Copies the selected node value to the clipboard.

Copy Name

Copies the selected node name to the clipboard.

This command supports [pandas dataframes](#) as well.

Debug Tool Window. Console

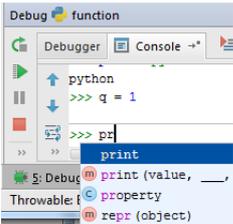
This tab is marked with  and shows the output and error stream messages.

On this page:

- [Console Toolbar](#)
- [Context Menu Options](#)
- [Keyboard Shortcuts](#)

Console Toolbar

Item
Tooltip
and
shortcut

	Up/down the Stack Trace	Click this button to navigate up or down in the stack trace and have the cursor jump to the corresponding location in the source code.  
	Use Soft Wraps	Click this button to toggle the soft wrap mode of the output.
	Scroll to the end	Click this button to navigate to the bottom of the stack trace and have the cursor jump to the corresponding location in the source code.
	Print	Click this button to send the console text to the default printer.
	Clear All	Click this button to remove all text from the console. This function is also available on the context menu of the console.
	Show Python prompt	If this button is pressed, you can enter commands in the console and view output . Note that in the debug console, code completion is available:  It's also possible to scroll through the history of commands with the up and down arrow keys.
	Browse history	Press this button to show the Debug Console History dialog box, where one can view the console entries and navigate through them, using the arrow keys.  Click OK to close the dialog box.

Context Menu Options

Item
Description

Compare with Clipboard	Opens the Clipboard vs Editor dialog box that allows you to view the differences between the selection from the editor and the current clipboard content. This dialog is a regular comparing tool that enables you to copy the line at caret to the clipboard, find text, navigate between differences and manage white spaces.
Copy URL	Choose this command to copy the current URL to the system clipboard. This command only shows on a URL, if it is included in an application's output.
Create Gist	Choose this command to open the Create Gist dialog box.
Clear All	Clears the output window.

Keyboard Shortcuts

The  key combination allows you to send EOF (end of file), i.e. to signal that no more data can be read from a data source.

Debug Tool Window: Elements Tab

In this tab, view the HTML source code that implements the active browser page and its [HTML DOM structure](#). Any changes made to the page through the browser are immediately reflected in the tab.

To Have the Tab Displayed:

1. Make sure you are using the **Chrome** browser for debugging.
2. Make sure the **JetBrains Chrome Extension** is installed in your Chrome browser, see [Installing JetBrains Chrome extension](#).
3. Make sure the **LiveEdit** repository plugin is installed and enabled. The plugin is not bundled with PyCharm, but it can be installed from the **JetBrains plugin repository** as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).

The Structure, Text, and Scripts Panes

The tab consists of three panes: Structure, Text, and Scripts.

The Structure pane shows the [HTML DOM structure](#) of the page that is currently active in the browser. The structure is updated dynamically according to the changes made on the page.

The Text pane shows HTML source code of the page that is currently active in the browser. The code is updated dynamically according to the changes made on the page.

The Scripts pane shows a tree of executed scripts.

The Structure and Text panes are mutually synchronized. When you click a node in the DOM structure, PyCharm scrolls through the contents of the Text pane.

The panes are also synchronized with the browser. PyCharm highlights the element in the browser as soon as you click the corresponding node in the DOM structure or in the Text pane.

Data View

In this section:

- [Opening the Data View tool window](#)
- [Supported formats](#)
- [Description of controls](#)

Opening the Data View tool window

This tool window appears in the following cases:

- One invokes the command View as Array/View as DataFrame in the [Variables tab](#) of the Debug tool window.
- One executes a Python code in the [Python Console](#), clicks  in the console toolbar, and then invokes the command View as Array/View as DataFrame from the context menu of an array or a DataFrame, or clicks the link View as Array/View as DataFrame .

Supported formats

The Data View tool window supports the following formats:

- [NumPy](#) arrays ([View as Array](#))
- [pandas dataframes](#) ([View as DataFrame](#)).

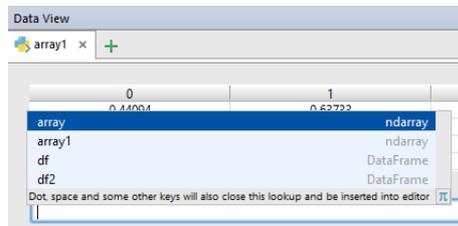
Description of controls

ItemDescription

Table The number of rows and columns correspond to the dimensions of an array or a DataFrame.

Colors of the table are regulated by the context menu of a tab (right-click a tab and select or clear the check-command Colored) or the menu from  (select or clear the check-command Colored by Default).

Text field Specify the name of an array or a DataFrame. Note that [code completion](#) is available in this field:



Format Use this field to change the presentation format. If the contents of this field changes to `%.2f`, then 2 digits will show after dot. Refer to the [Python documentation](#) for details.

Docker Tool Window

View | Tool Windows | Docker

For the tool window to be available, the Docker integration [plugin](#) must be installed and at least one Docker configuration must be defined.

The Docker tool window lets you manage your [Docker](#) images and containers.



All the available functions are accessed by means of the toolbar icons and context menu commands.

See also, [Docker](#).

Docker tool window icons and context menu commands

IconCommandDescription

Docker configurations

	Connect	Connect to Docker API. As a result, the list of Docker images and containers available locally is shown.
	Disconnect	Disconnect from Docker API. As a result, the list of Docker images and containers is hidden.
	Edit Configuration	Edit the Docker API settings for the selected configuration.
	Deploy	Deploy an image or artifact using the deployment settings in one of the Docker Deployment run configurations. The run configuration is selected from the list that is shown.
	Debug	Start one of the Docker Deployment run configurations in the debug mode. The run configuration is selected from the list that is shown.
	Pull image	Pull an image from Docker Hub or other image repository, e.g. Quay . (The Pull Image dialog will open.)
	Hide stopped containers	Hide the containers that are not running.
	Hide untagged images	Hide the images that have no tags.

Docker images

	Create container	Create a container for the selected image according to an existing or new Docker Deployment run configuration.
	Delete image	Delete the selected image or images.
	Push image	Push the selected image to Docker Hub or other image repository, e.g. Quay . (The Push Image dialog will open.)
	Hide untagged images	Hide the images that have no tags.
	Copy image ID	Copy an ID of the selected image to the clipboard.

Docker containers

	(Re)deploy	Deploy an image or artifact using the deployment settings in the corresponding Docker Deployment run configuration.
	Debug	Start an associated run configuration in the debug mode.
	Edit Configuration	Edit the settings for an associated run configuration.
	Start container	Start the selected container.
	Stop container	Stop the selected container.
	Delete container	Delete the selected container or containers.
	Hide stopped containers	Hide the containers that are not running.
	Open browser	If a web app is deployed in the container: open the default web browser.
	Copy container ID	Copy an ID of the selected container to the clipboard.
	Copy image ID	Copy an ID of the associated image to the clipboard.
	Show log	Show the container log.
	Inspect	Show low-level container information in JSON format.
	Show	Show the list of processes running in the container.

Attach Open the console for working with standard streams (stdin, stdout and stderr).

Documentation Tool Window

View | Tool Windows | Documentation

Ctrl+Q

This tool window appears in the following cases:

- [Quick Documentation lookup](#) is pinned by clicking  in the upper-right side of the lookup window.
- [Quick Definition tooltip](#) is pinned by clicking  in the upper-right corner of the tooltip.

As a result, the tool window appears in the list of available tool windows in the View menu, and gets the corresponding sidebar icon  [Documentation](#).

IconShortcutDescription

	<p>Left or Right</p>	<p>Switch to the previous or next documentation page (e.g. after using hyperlinks).</p> <p>Note On an macOS computer, you can also use the three-finger right-to-left and left-to-right swipe gestures.</p>
	<p>Shift+F1</p>	<p>View external documentation in the default browser.</p>
	<p>F4</p>	<p>Switch to the item (e.g. source) that corresponds to the documentation page currently shown.</p>
		<p>Turn the Auto-update from source option on or off. When the option is on, the information in the tool window is synchronized with your navigation in the editor and other places in the UI.</p>
		<p>Click this icon to show the font size slider. Move the slider to increase or decrease the font size as required.</p>

Duplicates Tool Window

View | Tool Windows | Duplicates

The Duplicates tool window displays results of the search for duplicates.

On this page:

- [Panels of the Duplicates tool window](#)
- [Left toolbar](#)
- [Upper toolbar](#)
- [Context menu commands](#)

Panels of the Duplicates tool window

The window consists of the following panes:

- The left pane displays the tree view of the duplicate fragments of source code. Each node shows the following information:
 - The number of duplicated code fragments found in scope.
 - The 'cost' of the duplicate (which is an arbitrary unit calculated using an additive algorithm on the base of the code block size; generally, the larger is the code fragment, the higher is its cost).
 - The containing class where the duplicates are located.
- The right pane shows the differences between the duplicated fragments of source code, selected in the left pane.

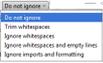
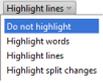
Left toolbar

Item Shortcut Description

	Rerun	Click this button to rerun the duplicates analysis in the active tab.
	Close Active Tab	Click this button to close the active tab.
		
	Autoscroll to Source	If the button is pressed, selecting an entry in the left pane opens the respective file in the editor.
		Click this button to show reference.

Upper toolbar

Item Tooltip/Image/Shortcut Description

		Move to the next/previous difference
Whitespace		<p>Use this drop-down list to define how the differences viewer should treat white spaces in the text.</p> <ul style="list-style-type: none">- Do not ignore: white spaces are important, and all differences are highlighted. This option is selected by default.- Trim whitespaces: <code>("\\t", " ")</code>, if they appear in the end and in the beginning of a line.<ul style="list-style-type: none">- If two lines differ in trailing whitespaces only, these lines are considered equal.- If two lines are different, such trailing whitespaces are not highlighted in the By word mode.- Ignore whitespaces: white spaces are not important, regardless of their location in the source code.- Ignore whitespaces and empty lines: the following entities are ignored:<ul style="list-style-type: none">- all whitespaces (as in the 'Ignore whitespaces' option)- all added or removed lines consisting of whitespaces only- all changes consisting of splitting or joining lines without changes to non-whitespace parts. <p>For example, changing <code>a b c</code> to <code>a \\n b c</code> is not highlighted in this mode.</p>
Highlighting mode		<p>Select the way differences granularity is highlighted.</p> <p>The available options are:</p> <ul style="list-style-type: none">- Highlight words: the modified words are highlighted- Highlight lines: the modified lines are highlighted- Highlight split changes: if this option is selected, big changes are split into smaller 'atomic' changes. <p>For example, <code>A \\n B</code> vs. <code>A X \\n B X</code> will be treated as two changes instead of one.</p> <ul style="list-style-type: none">- Do not highlight: if this option is selected, the differences are not highlighted at all. This option is intended for significantly modified files, where highlighting only introduces additional difficulties.
	Jump to Source	Click this button to open the file in the active pane in the editor. The caret will be placed in the same position as in the Duplicates tool window.
		
	Synchronize scrolling	Click this button to simultaneously scroll both differences panes; if this button is released, each of the panes can be scrolled independently.
	Editor settings	Click this button to invoke the list of available settings. Select or clear this options to show or hide whitespaces, line numbers and indent guides, to use or disable the use of soft wraps, and to set the highlighting level. These commands are also available from the context menu of the differences viewer gutter.

Context menu commands

Item	Keyboard Shortcut	Description
Jump to Source	F4	Open in the editor the file that contains the selected duplicate, and place the caret at the beginning of the duplicate. The fragment of code is highlighted.
Show Source	Ctrl+Enter	Open in the editor the file that contains the selected duplicate, and highlight the fragment of code.
Send to left/Send to right		Use these commands, or the arrow icons  and  , to place the selected duplicate to the left or right pane of the differences viewer.

Event Log

View | Tool Windows | Event Log

Event Log in the right-hand part of the bottom [tool window bar](#)

The Event Log tool window shows the information about "important" events that take place in PyCharm.

The information about problematic situations (e.g. errors and exceptions) is displayed in red. In such cases, clicking the more link (if present) opens a balloon with a more detailed description of the error or exception. Clicking a description link (depending on the error, the text may be different e.g.

NullPointerException) opens the IDE Fatal Errors dialog which lets you review the error and create a bug report.

- [General tab](#)
- [Database tab](#)
- [Toolbar](#)

General tab

The General tab appears when the [Database tab](#) opens. In the absence of the Database tab, the information is shown in the output pane which, in this case, is not tabbed.

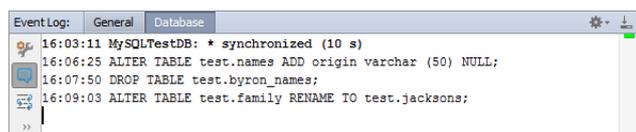
Database tab

Shown on the Database tab are the events related to working with the [Database tool window](#) and the [Table Editor](#).

For the Database tool window, the following may be shown:

- The information about data synchronizations and various manipulations with databases such as creating, modifying and deleting tables and columns, etc.
- Error messages.

For the Table Editor only error messages are shown.



Toolbar

ItemDescription

 Settings Use this icon to open the [Notifications page](#) where you can select which events you want to be notified of and also which events should be logged.

 Show balloons Use this icon to enable or disable showing notifications.
Note that this icon is a toggle which turns the corresponding feature on or off.

When showing notifications is enabled, you are notified of the events that take place in PyCharm. The corresponding notifications are shown in the balloons.

The alternative way to enable or disable this feature is by means of the Display balloon notifications check box on the [Notifications page](#).

 Use Soft Use this icon to turn on or off the soft wrap mode for the output.

Wraps

 Scroll to the end Use this icon to go to the end of the event log.

 Mark all as read Use this icon to mark all the messages as read.

 Clear all Use this icon to delete all the messages and thus clear the log.
This function can alternatively be accessed by means of the context menu.

 Help (F1) Use this icon or shortcut to open the corresponding help topic.

Favorites Tool Window

View | Tool Windows | Favorites

Alt+2

This tool window is marked with the icon .

The Favorites tool window lets you manage the following lists:

- [Favorite project items](#). Each list of Favorites in the tool window appears with the star icon .
- [Bookmarks](#). The list of bookmarks is marked with  icon.
- [Breakpoints](#). The list of breakpoints is marked with  icon.

Initially, the lists are empty. So doing, the favorites list has the same name as the project.

The lists of bookmarks and breakpoints are filled in automatically, as the new bookmarks or breakpoints are added.

To add items to favorites, do one of the following:

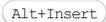
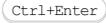
- Select one or more items in the [Project tool window](#) or the [Find tool window](#). Then choose File | Add To Favorites.
- Right-click an editor tab, and choose Add to Favorites or Add All to Favorites.

On this page:

- [Toolbar buttons](#)
- [Context menu commands](#)
- [Title bar context menu and buttons](#)
- [Context menu of the side bar button](#)
- [Using drag-and-drop](#)

Toolbar buttons

Item ShortcutDescription

		Use this button to create a new list of favorite items. In the Add New Favorites List dialog, specify the name for the new list and click OK.
		Click this button to rename the selected list.
		<p>Use this button to delete the selected list or list item.</p> <p>Depending on the selection, the following behaviors take place:</p> <ul style="list-style-type: none">- If a Favorites list , or one of the nested favorites is selected, click this button to delete an entire favorites list, or the selected item. So doing an item or a list is deleted from the Favorites tool window, but is left intact in project.- If a bookmark under the Bookmarks list  is selected, click this button to delete this bookmark from the list and from the source code. <p>Vice versa, if a bookmark is deleted from the source code, it is also removed from the Favorites tool window.</p> <ul style="list-style-type: none">- If a breakpoint under the Breakpoints list  is selected, click this button to delete the selected breakpoint from the list and from the source code. <p>Vice versa, if a breakpoint is deleted from the source code, it is also removed from the Favorites tool window.</p>

Context menu commands

When you right-click an item in the content pane, the context menu for this item is shown. This menu provides access to all the functions available for the selected item.

The commands and functionality are similar to those in the [Project tool window](#).

Title bar context menu and buttons

The title bar context menu provides the options for controlling the tool window [viewing modes](#). It also contains the commands for associating the tool window with a different [tool window bar](#), resizing and hiding the tool window.

To access the menu, right-click the window title bar.

Note that most of the menu options may alternatively be accessed by means of the title bar buttons.

Toolbar icon	Context menu command	Description
		Pinned, Docked, Floating, Windowed, Split Mode
		These options let you control general appearance and behavior of the tool window, see Viewing Modes .
		Move to
		To associate the tool window with a different tool window bar , select this command, and then select the destination tool window bar (Top,

Left, Bottom or Right).

Resize

To resize the tool window by moving one of its borders, select this command, and then select the necessary Stretch to option. Note that this command is not available in the floating mode.



Hide

Use this command to hide the tool window.

Shift+Escape

Tip When used in combination with the **Alt** key, clicking this button hides all the tool windows attached to the same **tool window bar**.



Use this button to collapse all expanded nodes.

Context menu of the side bar button

Right-click the side bar button to reveal this context menu. Refer to the section [Viewing Modes](#) for the detailed information about the viewing modes.

ItemDescription



Use this command to turn the Show Members option on or off. If this option is on, the class members (fields, methods, etc.) are shown.



Use this command to turn the Autoscroll to Source option on or off. If this option is on, PyCharm automatically navigates from a file (or a class member) selected in the Favorites tool window to the corresponding source file (or its fragment) in the editor. If the corresponding file is not currently open, it will open automatically.



Use this command to turn the Autoscroll from Source option on or off. If this option is on, PyCharm automatically navigates from a file (or a class member) selected in the editor, to the corresponding file in the Favorites tool window.

Using drag-and-drop

You can use drag-and-drop to:

- Add items to favorites: drag the item of interest from the Project tool window and drop it onto the desired favorites list in the Favorites tool window.
- Drag an external item from the Explorer/Finder and drop it onto the desired favorites list in the Favorites tool window.
- Move an item from one favorites list to another in the Favorites tool window.

Find Tool Window

View | Tool Windows | Find

Alt+3

On this page:

- [Basics](#)
- [Toolbar buttons](#)
- [Context menu commands](#)

Basics

Find tool window displays results of the following searches:

- [Find/Replace in Path](#)
- [Find Usages](#)
- [Refactoring Preview](#)
- [Find Usages of a data source](#), a table, or a column.

The results of each search are displayed in a separate tab, or replace the contents of the current tab, depending on the Open in new tab dialog setting. By default the window appears at the bottom of the screen.

It has a toolbar with a set of buttons, a pane of results, and additional buttons for [Replace in Path](#), and [Refactoring Preview](#) operations.

Warning! Unless you find something, this tool window is not visible in the View menu.

Toolbar buttons

Item	Tooltip and shortcut	Description
	Options Ctrl+Shift+Alt+F7	Click this button to open one of the Find Usages dialogs, which corresponds to the symbol in question. You can edit the search settings and click Rerun button to execute the modified search query.
	Rerun Ctrl+F5	Rerun the last search. This button is not available for viewing code coverage results .
	Close Ctrl+Shift+F4	Close the current tab or the tool window. This button is not available in Replace in Path and Refactoring Preview dialogs.
	Pin	Use to pin or unpin the tab. If a tab is pinned, the results for the next command are shown on a new tab.
	Recent find usages Ctrl+E	Show the list of recent searches. Select an item in the list to see the search results.
	Expand all	Use these buttons to have all nodes expanded or collapsed.
	Collapse all Ctrl+NumPad +	
	Previous/next occurrence Ctrl+Alt+Up Ctrl+Alt+Down	Navigate to the previous/next element in the tab of results.
	Autoscroll to source	Turns the Autoscroll to source option on or off. When the option is on and you select the search result, the corresponding source file opens in the editor and the appropriate fragment is highlighted in the file.
	Favorites	Click this button to add found usages to favorites .
	Export to Text File Alt+O	Save the contents of the current result tab. In the Export preview dialog, specify the target file or copy information to the clipboard. Before saving, you can also modify the information to be saved.
	Help F1	Use this icon or shortcut to open the corresponding help page.
	Group by usage type	This button is available for Find Usages only. If this button is pressed, the search results are grouped by the

Ctrl+T

following categories:
– comments
– unclassified usage not related to any of the categories



Group by test/production

If this button is pressed, the usages are grouped according to the Production and Test scopes.



Group by module

If this button is pressed, the found usages show under the corresponding module or library node.

Ctrl+D

Tip This type of grouping is helpful, when a package is split between several modules.



Group by package

If this button is pressed, all the usages found are displayed under their respective packages.

Ctrl+P



Group by file structure

If this toggle is on, the found usages are shown under the corresponding method nodes.

Ctrl+M



Merge usages from the same line

If this toggle is on, the duplicate usages found on the same line are merged.

Ctrl+F



Show read access

This button is available for [Find Usages](#) only.

Ctrl+R

If this button is pressed, the search results include references to the read access methods.



Show write access

This button is available for [Find Usages](#) only.

Ctrl+W

If this button is pressed, the search results include references to the write access methods.



Show import statements

This button is available for [Find Usages](#) only.

Ctrl+I

If this button is pressed, the search results include the usages in the import statements.



Preview usages

Turns showing the Preview pane on or off.



Sort members alphabetically

Click this button to have all members sorted alphabetically. Otherwise, members are sorted in the order they are declared.

Context menu commands

Item **Shortcut****Description**

Jump to Source

F4

Navigate to the selected item in the source code.

Include

Insert

For an [excluded item](#); include the item in the list of results.

Exclude

Delete

Exclude the selected item from the list of results. (Excluded items are shown strikethrough.)
When you carry out the Replace All or the Do Refactor command, the excluded items are not affected.

Remove

Alt+Delete

Remove the selected item from the list of results.

Recent Find Usages

Ctrl+E

Show the list of recent searches. Select an item in the list to see the search results.

Add to Favorites

[Add selected node to the Favorites list.](#)

Flow Tool Window

On this page:

- [Errors pane](#)
- [Project Errors pane](#)
- [Toolbar](#)
- [Context Menu](#)

Errors pane

The pane shows a list of all the discrepancies detected in the file which is opened in the active editor tab. At the top the full path to the file is displayed.

Project Errors pane

The pane shows a list of all the discrepancies detected in all the files in the current project after you run Flow against the entire project by clicking  on the toolbar. The error messages are grouped by files in which they were detected.

Toolbar

The toolbar is common for the Current Errors and the Project Errors panes.

Item	Tooltip	Description
	Show all errors	Click this button to switch to the Project Errors pane and view all the discrepancies detected in the current project.
	Help	Use this button to navigate to the help topic for the tool window.
	Close Ctrl+Shift+F4	Click this button to terminate the Flow type checker and close the tool window.
	Expand all	Use these buttons to have all nodes expanded or collapsed.
	Collapse all Ctrl+NumPad +	
	Clear All Ctrl+NumPad -	Click this button to remove all the error messages from the currently active pane.

Context Menu

The context menu is common for the Errors and the Project Errors panes.

Item	Description
Jump to source	Choose this option to open the file where the selected problem was detected and navigate to the fragment of code which caused the error.
Copy	Choose this option to copy the selected error message with the information on the file, the line, and the column where the error occurred.

Grunt Tool Window

Context menu of a Gruntfile.js - Show Grunt Tasks

View | Tool Windows | Grunt

On this page:

- [Accessing the Grunt Tool Window](#)
- [Building a Tree of Grunt Tasks](#)
- [Running Grunt Tasks and Targets](#)
- [Toolbar](#)
- [Context Menu of a Tree](#)
- [Context Menu of a Task or a Target](#)

Accessing the Grunt Tool Window

View | Tool Windows | Grunt

- the tool window can be accessed this way only after you have opened it using the Show Grunt Tasks command.

The tool window is available only when:

1. The [Node.js](#) runtime environment is installed on your computer.
2. The [NodeJS](#) repository plugin is installed and enabled. The plugin is not bundled with PyCharm, but it can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).
3. The `grunt-cli` package is installed globally and the `grunt` package is installed in the current project, see [Installing Grunt](#) for details.
4. At least one `Gruntfile.js` file is available in the current project.

The tool window opens when you invoke **Grunt** by choosing Show Grunt Tasks on the context menu of a `Gruntfile.js` in the Project tool window or of a `Gruntfile.js` opened in the editor.

As soon as you invoke **Grunt**, it starts building a tree of tasks according to the `Gruntfile.js` on which it was invoked. If a task has [targets](#), the task is displayed as a node and the targets are listed under it.

If you have several `Gruntfile.js` files in your project, you can build a separate tasks tree for each of them and run tasks without abandoning the previously built tasks trees. Each tree is shown in a separate tab.

Building a Tree of Grunt Tasks

To build a tasks tree, do one of the following:

- Select the required `Gruntfile.js` file in the Project tool window and choose Show Grunt Tasks on the context menu of the selection.
- Open the required `Gruntfile.js` file in the editor and choose Show Grunt Tasks on the context menu of the editor.
- If the Grunt tool window is already opened, click  on the toolbar and choose the required `Gruntfile.js` file from the list. PyCharm adds a new node and builds a tasks tree under it. The title of the node shows the path to the `Gruntfile.js` file according to which the tree is built.

By default, the tasks in a tree are listed in the order in which they are defined in `Gruntfile.js` (option Definition order). To have them listed in the alphabetic order, click the  toolbar button, then choose Sort by on the menu, and then choose Name.

Running Grunt Tasks and Targets

To run a task or a target from the tree, do one of the following:

- Double click the required task or target in the tree.
- Select the required task or target and choose Run <task name> on the context menu of the selection.
- Select the required task or target and press .
- To run the `default` task, select the root node in the tree and choose Run default on the context menu of the selection.
- To navigate to the definition of a task or target, select the required task or target in the tree and choose Jump to source on the context menu of the selection.

The task or target execution output will be displayed in the [Run tool window](#). The name of the target is shown in the format `<task name>:<target name>`. The tool window shows the Grunt output, reports the errors occurred, lists the packages or plugins that have not been found, etc. The name of the last executed task is displayed on the title bar of the tool window.

To run several tasks or targets, use the multiselect mode: hold  (for adjacent items) or  (for non-adjacent items) keys and select the required tasks or targets, then choose Run on the context menu of the selection.

Toolbar

ItemTooltipDescription

	Add Gruntfile	Click this button to have a tasks tree for another <code>Gruntfile.js</code> file built. Choose the required <code>Gruntfile.js</code> file from the pop-up list. PyCharm adds a new node and builds a tasks tree under it.
	Remove Gruntfile	Click this button to remove the tasks tree under the selected node.
	Reload tasks	Click this button to have the tasks tree under the selected node re-built. You may need a tree re-built after updating the tasks

	corresponding <code>Gruntfile.js</code> file because Grunt does not apply changes to trees on the fly.
	Click this button to hide all the tasks trees and have only <code>Gruntfile.js</code> nodes displayed.
	<p>Click this button to configure the current view and to change the viewing modes of the tool window, see Viewing Modes for details. Note that most of the menu items are options that you can turn on or off. An option which is on has a check mark to the left of its name. The Grunt-specific options are:</p> <ul style="list-style-type: none"> – Grunt Settings: choose this menu item to open the Grunt Settings dialog and re-configure the current settings for Grunt and for the Node interpreter, see Using Grunt Task Runner. – Sort by: choose this menu item to configure the order in which tasks are shown in trees. By default, the tasks in a tree are listed in the order in which they are defined in <code>Gruntfile.js</code> (option Definition order). To have them listed in the alphabetic order, click the  toolbar button, then choose Sort by on the menu, and then choose Name.
	<p>Hide Click this button to hide the tool window. To have it displayed again, choose View Tool Windows Grunt on the main menu. The tool window appears again showing all the previously built trees of tasks.</p>

Context Menu of a Tree

ItemDescription

Grunt Settings	Choose this menu item to open the Grunt Settings dialog box and view or edit the <code>Node.js</code> configuration
Jump to Source	Choose this menu item to open the <code>Gruntfile.js</code> file for which the current tree is built.
Reload tasks	Choose this menu item to have the tree of tasks under the selected node re-built.
Copy Path	Choose this menu item to save the path to the <code>Gruntfile.js</code> file according to which the current tree was built to the clipboard.
Remove Gruntfile.js	Choose this menu item to remove the tree of tasks under the selected node.

Context Menu of a Task or a Target

ItemDescription

Run <task/target name>	Choose this menu item to run the selected task or target.
Debug <task/target name>	Choose this menu item to debug the selected task or target.
Edit <task/target name> settings	Choose this menu item to open the Run/Debug Configuration dialog box and edit the predefined settings for the selected task or target.
Jump to Source	Choose this menu item to open the <code>Gruntfile.js</code> file for which the current tree is built and navigate to the definition of the selected task or target.

Gulp Tool Window

On this page:

- [Accessing the Gulp Tool Window](#)
- [Building a Tree of Gulp Tasks](#)
- [Running Gulp Tasks](#)
- [Toolbar](#)
- [Context Menu of a Tree](#)
- [Context Menu of a Task or a Target](#)

Accessing the Gulp Tool Window

Context menu of a Gulpfile.js - Show Gulp Tasks

View | Tool Windows | Gulp - the tool window can be accessed this way only after you have opened it using the Show Gulp Tasks command or after you have run tasks through the **Gulp.js run configuration**.

The tool window is available only when:

1. The [Node.js](#) runtime environment is installed on your computer.
2. The **NodeJS** repository plugin is installed and enabled. The plugin is not bundled with PyCharm, but it can be installed from the **JetBrains plugin repository** as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).
3. The `gulp` package is installed in the current project, see [Installing Gulp](#) for details.
4. At least one `Gulpfile.js` file is available in the current project.

The tool window opens when you invoke **Gulp.js** by choosing Show Gulp Tasks on the context menu of a `Gulpfile.js` in the Project tool window or of a `Gulpfile.js` opened in the editor. The tree is built according to the `Gulpfile.js` file on which **Gulp.js** was invoked. If you have several `Gulpfile.js` files in your project, you can build a separate tasks tree for each of them and run tasks without abandoning the previously built tasks trees. Each tree is shown under a separate node.

If you have several `Gulpfile.js` files in your project, you can build a separate tasks tree for each of them and run tasks without abandoning the previously built tasks trees. Each tree is shown under a separate node.

Building a Tree of Gulp Tasks

To build a tree of tasks, do one of the following:

- Select the required `Gulpfile.js` file in the Project tool window and choose Show Gulp tasks on the context menu of the selection.
- Open the required `Gulpfile.js` file in the editor and choose Show Gulp tasks on the context menu of the editor.
- If the Gulp tool window is already opened, click  on the toolbar and choose the required `Gulpfile.js` file from the list. PyCharm adds a new node and builds a tasks tree under it. The title of the node shows the path to the `Gulpfile.js` file according to which the tree is built.

By default, the tasks in a tree are listed in the order in which they are defined in `Gulpfile.js` (option Definition order). To have them listed in the alphabetic order, click the  toolbar button, then choose Sort by on the menu, and then choose Name.

Running Gulp Tasks

To run a task, do one of the following:

- Double click the required task in the tree.
- Select the required task and choose Run <task name> on the context menu of the selection.
- Select the required task and press .
- To run the `default` task, select the root node in the tree and choose Run default on the context menu of the selection.
- To navigate to the definition of a task, select the required task in the tree and choose Jump to source on the context menu of the selection.

The task execution output will be displayed in the [Run tool window](#).

To run several tasks, use the multiselect mode: hold  (for adjacent items) or  (for non-adjacent items) keys and select the required tasks, then choose Run on the context menu of the selection.

Toolbar

ItemTooltipDescription

	Add Gulpfile	Click this button to have a tasks tree for another <code>Gulpfile.js</code> file built. Choose the required <code>Gulpfile.js</code> file from the pop-up list. PyCharm builds a tasks tree and shows it under a separate node.
	Remove Gulpfile	Click this button to remove the tasks tree under the selected node.
	Reload tasks	Click this button to have the tasks tree under the selected node re-built. You may need a tree re-built after updating the corresponding <code>Gulpfile.js</code> file because Gulp.js does not apply changes to trees on the fly.
	Collapse all	Click this button to hide all the tasks trees and have only <code>Gulpfile.js</code> nodes displayed.
		Click this button to configure the current view and to change the viewing modes of the tool window, see Viewing Modes for details. Note that most of the menu items are options that you can turn on or off. An option which is on has a check mark to the left of its name. The Gulp -specific options are: <ul style="list-style-type: none">– Gulp Settings: choose this menu item to open the Gulp Settings dialog and re-configure the current settings for Gulp and

for the [Node interpreter](#), see [Using Gulp Task Runner](#).

- Sort by: choose this menu item to configure the order in which tasks are shown in trees. By default, the tasks in a tree are listed in the order in which they are defined in `Gulpfile.js` (option Definition order). To have them listed in the alphabetic order, click the  toolbar button, then choose Sort by on the menu, and then choose Name.



Hide

Click this button to hide the tool window. To have it displayed again, choose View | Tool Windows | Grunt on the main menu. The tool window appears again showing all the previously built trees of tasks.

Context Menu of a Tree

ItemDescription

Gulp Settings	Choose this menu item to open the Gulp Settings dialog box and view or edit the <code>Node.js</code> configuration
Jump to Source	Choose this menu item to open the <code>Gulpfile.js</code> file for which the current tree is built.
Reload tasks	Choose this menu item to have the tree of tasks under the selected node re-built.
Copy Path	Choose this menu item to save the path to the <code>Gulpfile.js</code> file according to which the current tree was built to the clipboard.
Remove Gulpfile.js	Choose this menu item to remove the tree of tasks under the selected node.

Context Menu of a Task or a Target

ItemDescription

Run <task name>	Choose this menu item to run the selected task.
Debug <task name>	Choose this menu item to debug the selected task.
Edit <task name> settings	Choose this menu item to open the Run/Debug Configuration dialog box and edit the predefined settings for the selected task.
Jump to Source	Choose this menu item to open the <code>Gulpfile.js</code> file for which the current tree is built and navigate to the definition of the selected task.

Hierarchy Tool Window

View | Tool Windows | Hierarchy

Alt+8

Use this tool window to analyze and [navigate](#) through hierarchies of classes .

Warning! The tool window is available only after you have [built a hierarchy](#) for the first time.

The contents of the tool window are not automatically updated as you navigate through the source code or switch between the editor tabs. The tool window shows the results of the latest hierarchy command and is updated when you run the next hierarchy command, unless the tab with one of the previously built hierarchies is [pinned](#) .

Toolbar buttons

Item	Description	Available In
	When this button is pressed, the hierarchical tree shows both the parent and child classes of the selected class, which is marked with an arrow  in the tree of results.	Class hierarchies
	When this button is pressed, the tool window shows all classes that extend the selected class.	Class hierarchies Call hierarchies
	When this button is pressed, the tool window shows the hierarchy of each supertype of the current class.	Class hierarchies Call hierarchies
	When this button is pressed, the elements within a tree are sorted alphabetically.	All hierarchies
Scope	Use this drop-down list to limit the scope of the current hierarchy: <ul style="list-style-type: none">- Project - PyCharm traces usages of the method across the project.- Test - PyCharm traces usages of the method across the test classes.- All - PyCharm traces usages of the method across the project and the libraries.- This class - the scope is limited to the current class. <p>In addition to the pre-configured scopes, you can define your own one: select Configure from the drop-down list, and define the required scope in the Scopes dialog box that opens.</p>	Call hierarchies

In a method hierarchy, [the tree-views](#) of the following classes are available:

-  - where this method is defined.
-  - where this method is not defined, but defined in the superclass.
-  - where this method should be defined, because the class is not abstract.

	If you made any changes of the classes or the class structure, they become visible in the Hierarchy tool window only after you press this button.	All hierarchies
	Toggle the Autoscroll to source mode. When the button is pressed, every time the node is focused, the corresponding line of source code is highlighted in the editor.	All hierarchies
	When this button is pressed, the current tab will not be overwritten; instead, the results of the next command will be displayed in a new tab.	All hierarchies
	Click this button to export a hierarchy into a text file in the specified location.	All hierarchies
	Click this button to close the selected tab of results.	All hierarchies
	Click this button to show reference page.	All hierarchies

Inspection Results Tool Window

View | Tool Windows | Inspection Results

- You can access the tool window this way only when it is already opened through Code | Inspect Code.
- After you deactivate the tool window manually by clicking the Close button , the tool window is available again only through Code | Inspect Code.

The Inspection Results tool window displays inspection results on separate tabs.

The left-hand pane of each tab shows a tree view of the inspections for which problems are found. The right-hand pane shows summary information for an item selected in the left-hand pane.

On this page:

- [Toolbar buttons](#)
- [Context menu commands](#)
- [Inspection report](#)

Toolbar buttons

ItemShortcutDescription

	Ctrl+F5	Click this button to run the inspection and show the results on the same tab.
	Ctrl+Shift+F4	Click this button to close the current tab or the tool window.
	Ctrl+NumPad Plus	Expand all nodes
	Ctrl+NumPad -	Collapse all nodes
	Ctrl+Alt+Down	Navigate to the next item.
	Ctrl+Alt+Up	Navigate to the previous item.
	F1	Use this icon or shortcut to open the corresponding help page.
		If this toggle is on, the problems are grouped into Errors and Warnings. Otherwise, the problems are grouped by inspections.
		Turns grouping by directories on or off.
		When this toggle is on, the resolved and excluded items are not shown.
		Turns the Autoscroll to source option on or off. When the option is on and you select an item, the corresponding source file opens in the editor and the appropriate fragment is highlighted.
		Click this button to export the inspection results into XML or HTML format .
		Click this button to edit the current inspection settings .
	Alt+Enter	Click this button to resolve the problem for the selected inspection item by choosing one of the available quick fixes from the list.

Context menu commands

Item ShortcutDescription

Jump to Source	F4	Open the file that contains the selected problem in the editor and place the cursor at the beginning of the corresponding code fragment.
Exclude	Delete	Exclude the selected items from further examination. Excluded nodes are shown strikethrough. If the filter toggle  is on, the excluded nodes are hidden.
Include	Insert	Include previously excluded items in the list of results. All nested subelements are included too.
	Alt+Enter	Select one of the suggested solutions.
Suppress problem /		Suppress the inspection for the selected problem or the selected class.
Suppress problem for class		
Edit Settings		Change the settings for the selected inspection or group of inspections in the Errors dialog.
Disable inspection		Disable alerts for the selected inspection in the active tab of results. If the filter toggle  is on, the nodes for disabled inspections are hidden. See also, Disabling and Enabling Inspections .
Run inspection on		Rerun the selected inspection and display the results on a new tab.

Inspection report

The inspection report is shown in the right-hand pane of the results tab when an inspection node is selected in the left-hand pane. The report may include the following:

- Problem resolution: A button for each of the available solutions. Clicking a button invokes the corresponding fix. If no buttons are present, you have to fix the problem yourself.
- Suppress: Click this button to reveal the list of inspection suppress options.

- Problem synopsis: A brief description of the problem.
- Disable inspection: Disable alerts for the selected inspection in the active tab of results. If the filter toggle  is on, the nodes for disabled inspections are hidden.
See also, [Disabling and Enabling Inspections](#).
- Run inspection on...: Rerun the selected inspection and display the results on a new tab.

JSTestDriver Server Tool Window

View | Tool Windows | JSTestDriver Server

 - Start a local server link.

In this tool window, start and stop the JSTestDriver Server to run unit tests against and capture the browser to execute unit tests in.

Prerequisites

The tool window becomes available when the following prerequisites are met:

1. The **JSTestDriver** plugin is downloaded, installed, and enabled. The plugin is not bundled with PyCharm, but it can be installed from the **JetBrains plugin repository** as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).
2. The current project contains at least one [test runner configuration files](#).

Options

ItemTooltip and shortcut	Description
--------------------------	-------------

	Run a local server Click this button to have PyCharm launch the default JSTestDriver server.
---	---

	Stop the local server Click this button to have PyCharm stop the currently running JSTestDriver server.
---	--

Capture a browser using the URL	This read-only field shows the URL address to access the Remote Console of the JSTestDriver. Copy the URL address and open it in the browser of your choice.
---------------------------------	--

	The icons indicate available browsers. The icon that corresponds to the browser you just opened, is active. Click the icon to get ready for executing tests.
--	--

NPM Tool Window

Context menu of a package.json - Show npm Scripts

View | Tool Windows | npm

- the tool window can be accessed this way only after you have opened it using the Show npm Scripts command.

On this page:

- [Accessing the npm Tool Window](#)
- [Building a Tree of npm Scripts](#)
- [Running npm Scripts](#)
- [Toolbar](#)
- [Context Menu of a Tree](#)
- [Context Menu of a Script](#)

Accessing the npm Tool Window

The tool window is available only when:

1. The [Node.js](#) runtime environment is installed on your computer.
2. The [NodeJS](#) repository plugin is installed and enabled. The plugin is not bundled with PyCharm, but it can be installed from the [JetBrains plugin repository](#) as described in [Installing, Updating and Uninstalling Repository Plugins](#) and [Enabling and Disabling Plugins](#).
3. At least one `package.json` file is available in the current project.

The tool window opens when you invoke **npm** by choosing Show npm Scripts on the context menu of a `package.json` in the Project tool window or of a `package.json` opened in the editor. Use the tool window to run **npm scripts**.

As soon as you invoke **npm**, it starts building a tree of scripts defined within the `scripts` property of the `package.json` file on which it was invoked. If you have several `package.json` files in your project, you can build a separate script tree for each of them and run scripts without abandoning the previously built trees. Each tree is shown under a separate node.

Building a Tree of npm Scripts

To build a tree of scripts, do one of the following:

- Select the required `package.json` file in the Project tool window and choose Show npm Scripts on the context menu of the selection.
- Open the required `package.json` file in the editor and choose Show npm Scripts on the context menu of the editor.
- If the npm tool window is already opened, click  on the toolbar and choose the required `package.json` file from the list. PyCharm adds a new node and builds a scripts tree under it. The title of the node shows the path to the `package.json` file according to which the tree is built.
- To re-build a tree, switch to the required node and click  on the toolbar.

By default, the scripts in a tree are listed in the order in which they are defined in `package.json` (option Definition order). To have them listed in the alphabetic order, click the  toolbar button, then choose Sort by on the menu, and then choose Name.

Running npm Scripts

To run a script from the tree, do one of the following:

- Double click the required script in the tree.
- Select the required script and choose Run <script name> on the context menu of the selection.
- Select the required script and press .

The tool window shows the npm script output, reports the errors occurred, lists the packages or plugins that have not been found, etc. The name of the last executed script is displayed on the title bar of the tool window.

To run several scripts, use the multiselect mode: hold  (for adjacent items) or  (for non-adjacent items) keys and select the required scripts, then choose Run on the context menu of the selection.

Toolbar

Item Tooltip Description

	Add package.json	Click this button to have a tree of scripts for another <code>package.json</code> file built. Choose the required <code>package.json</code> file from the pop-up list. PyCharm adds a new node and builds a tree of scripts under it it.
	Remove package.json	Click this button to remove the tree of scripts under the selected node.
	Reload scripts	Click this button to have the tree of scripts under the selected node re-built. You may need a tree re-built after updating the corresponding <code>package.json</code> file because npm does not apply changes to trees on the fly.
	Collapse all	Click this button to hide all the scripts trees and have only <code>package.json</code> nodes displayed.
		Click this button to configure the current view and to change the viewing modes of the tool window, see Viewing Modes for details. Note that most of the menu items are options that you can turn on or off. An option which is on has a check mark to the left of its name. The npm -specific options are: <ul style="list-style-type: none">- npm Settings: choose this menu item to open the npm Settings dialog and re-configure the current settings for npm and for the Node interpreter, see Running NPM Scripts.- Sort by: choose this menu item to configure the order in which scripts are shown in trees. By default, the scripts in a tree are listed in the order in which they are defined in <code>package.json</code> (option Definition order). To have them listed in the

alphabetic order, click the  toolbar button, then choose Sort by on the menu, and then choose Name.

	Hide	Click this button to hide the tool window. To have it displayed again, choose View Tool Windows Grunt on the main menu. The tool window appears again showing all the previously built trees of tasks.
---	------	--

Context Menu of a Tree

ItemDescription

npm Settings	Choose this menu item to open the npm Settings dialog box and view or edit the <code>Node.js</code> configuration
Jump to Source	Choose this menu item to open the <code>package.json</code> file for which the current tree is built.
Reload scripts	Choose this menu item to have the tree of scripts under the selected node re-built.
Copy Path	Choose this menu item to save the path to the <code>package.json</code> file according to which the current tree was built to the clipboard.
Remove package.json	Choose this menu item to remove the tree of scripts under the selected node.

Context Menu of a Script

ItemDescription

Run <script name>	Choose this menu item to run the selected script.
Edit <script name> settings	Choose this menu item to open the Run/Debug Configuration dialog box and edit the predefined settings for the selected script.
Jump to Source	Choose this menu item to open the <code>package.json</code> file for which the current tree is built and navigate to the definition of the selected script.

Project Tool Window

View | Tool Windows | Project

Alt+1

The Project tool window lets you look at your project from various viewpoints and perform the tasks such as creating new items (directories, packages, files, classes, etc.), opening files in the editor, navigating to the code fragment of interest, and more.

Most of the functions in this tool window are accessed as context menu commands in the content pane and associated shortcuts.

- [Views](#)
- [Context menu for the title bar](#)
- [Title bar icons](#)
- [Content pane](#)
- [Context menu commands for the content pane items](#)
- [File status highlights](#)

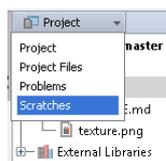
Views

The tool window provides a number of views.

Different views emphasize different project aspects and, generally, define which items are shown and how:

- Project view. In this view, all the project items along with their dependencies (SDKs and libraries) are shown. The emphasis is on the directory structure (though the packages are also shown).
- Scopes views (Project Files, Problems, etc.). What is shown in the content pane is limited to the corresponding predefined or user-defined [scope](#). In other respects, depending on the currently selected view options, a scope view may resemble the Project view.
- Scratches view. This view lets you manage your [scratch files](#) and [database consoles](#).

The necessary view is selected from the list in the left-hand part of the title bar or, if the views are represented by tabs, by clicking the corresponding tab.



To configure a view, use the corresponding options in the [title bar context menu](#). The necessary options can also be accessed by clicking  on the title bar.

Context menu for the title bar

The title bar context menu provides the options that let you control every aspect of the tool window appearance and behavior.

There are the options for configuring the project [views](#). There are also the ones for controlling the tool window [viewing modes](#).

The options that are on, have a check mark to the left of their names.

The menu also includes commands for switching between the views, resizing the tool window, and more.

To access the menu, right-click the list of views in the left-hand part of the title bar, or, if the views are represented by tabs, right-click the tab of interest.

The following table lists and briefly explains the available commands and options.

ItemShortcutDescription

Select Next View or Tab		These are the commands for switching the views.
Select Previous View or Tab		
Show List of Views or Tabs		
Edit Scopes		Use this command to open the Scopes dialog in which you can create and edit used-defined scopes . Note that this command is available only if the current view is a scope view.
Show Members		If this option is on, the files in the tree that contain classes turn into nodes. When such node is unfolded, the contained classes with their fields, methods, and other members of the selected item are shown.
Autoscroll to Source		If this option is on, PyCharm automatically navigates from a file (or a class member) selected in the Project tool window to the corresponding source file (or its fragment) in the editor. If the corresponding file is not currently open, it will open automatically.
Autoscroll from Source		If this option is on, PyCharm automatically navigates from a file in the editor to the corresponding node (file, class, field, method, etc.) in the Project tool window.
Sort by Type		If the option is off, the items (files, classes, etc.) are sorted alphabetically. If the option is on, the files are sorted by their extensions.
Folders Always on Top		If the option is on, all the folders are shown before the files. Otherwise, all the items are sorted alphabetically, and the files and folders appear intermixed.

Show Excluded Files	<p>This option is available only in the Project view. (In other views, excluded files are never shown.) Turn this option on or off to show or hide excluded folders and files.</p> <div style="display: flex; justify-content: space-around;"> <div style="border: 1px solid #ccc; padding: 5px; width: 45%;"> <p>Show Excluded Files: OFF</p> </div> <div style="border: 1px solid #ccc; padding: 5px; width: 45%;"> <p>Show Excluded Files: ON</p> </div> </div>
---------------------	--

Pinned, Docked, Floating, Windowed, Split Mode	These options let you control general appearance and behavior of the tool window. See Viewing Modes .
--	---

Remove from Sidebar	<p>This command hides the tool window, removes the associated tool window button from the tool window bar and removes the tool window from the quick access menu (☰ or ☰).</p> <p>To open the tool window again (and restore the associated features), use the main menu: View Tool Windows <Window Name>.</p>
---------------------	--

Group Tabs	<p>If this option is on, there is a list in the left-hand part of the title bar from which you can select the necessary view. If this option is off, the views are represented by tabs which appear in the left-hand part of the title bar.</p> <div style="display: flex; justify-content: space-around;"> <div style="border: 1px solid #ccc; padding: 5px; width: 45%;"> <p>Group Tabs: ON</p> </div> <div style="border: 1px solid #ccc; padding: 5px; width: 45%;"> <p>Group Tabs: OFF</p> </div> </div>
------------	---

Move to	To associate the tool window with a different tool window bar , select this command, and then select the destination tool window bar (Top, Left, Bottom or Right).
---------	--

Resize	To resize the tool window by moving one of its borders, select this command, and then select the necessary Stretch to option. Note that this command is not available for the floating mode.
--------	--

Hide	<div style="display: flex; align-items: center;"> <div style="border: 1px solid #ccc; border-radius: 10px; padding: 2px 5px; margin-right: 5px;">Shift+Escape</div> <div>Use this command to hide the tool window.</div> </div>
------	---

Title bar icons

ItemShortcutDescription

	<div style="border: 1px solid #ccc; border-radius: 10px; padding: 2px 5px; display: inline-block;">Alt+Right</div> <div style="border: 1px solid #ccc; border-radius: 10px; padding: 2px 5px; display: inline-block;">Alt+Left</div>	<p>If the views are currently shown as tabs (the Group Tabs option is off), this button appears to the right of the last visible tab. If the first or the last of the available views is currently selected, this button is shown as ▶ or ◀.</p> <p>Click this button to open the list of views, for example, to select a different view.</p>
--	---	---

	<p>Click this icon to navigate from a file in the editor to the corresponding node (file, class, field, method, etc.) in the Project tool window.</p> <p>This icon is not available if the Autoscroll from Source option is currently on.</p>
--	---

	<div style="border: 1px solid #ccc; border-radius: 10px; padding: 2px 5px; display: inline-block;">Ctrl+NumPad -</div>	Use this icon or shortcut to collapse all the nodes.
--	--	--

	<p>Click this button to open the menu for configuring the current view and changing the tool window viewing modes. Note that most of the menu items are options that you can turn on or off. An option which is on has a check mark to the left of its name.</p> <p>The available options are a subset of the title bar context menu items. Depending on the current view, the menu may include the following options:</p> <ul style="list-style-type: none"> - Edit Scopes - Show Members - Autoscroll to Source - Autoscroll from Source - Sort by Type - Folders Always on Top - Show Excluded Files - Pinned Mode, Docked Mode, Floating Mode, Windowed Mode, Split Mode - Remove from Sidebar - Group Tabs - Move to - Resize
--	--

	<div style="border: 1px solid #ccc; border-radius: 10px; padding: 2px 5px; display: inline-block;">Shift+Escape</div>	<p>Use this icon or shortcut to hide the tool window.</p> <p>When used in combination with the Alt key, clicking this icon hides all the tool windows attached to the same tool window bar.</p>
--	---	---

Content pane

The content pane shows the project items such as modules, packages, directories, files, classes, libraries, etc.

The icons for the main categories (node types) are shown and briefly explained in the following table. The icons used for the main file types are listed in [File Types Recognized by PyCharm](#); the icons for the main symbols (classes, fields, methods, etc.) are shown in [Symbols](#).

ItemDescription

	A module.
	A package.
	<p>A folder (directory). Different folder types have different colors:</p> <ul style="list-style-type: none"> - An "ordinary" folder

– A source folder 

For more information on folder types, see [Content Root](#).



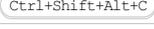
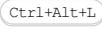
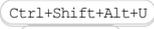
External Libraries, a category for grouping third party software associated with the project. The node is only shown when the Project view is selected.

Context menu commands for the content pane items

When you right-click an item in the content pane, the context menu for this item is shown. This menu provides access to all the functions available for the selected item.

The following table lists and briefly explains the most frequently used context menu commands.

Item ShortcutDescription

New		Use this command to create a new item (directory, file, or class), within the selected one (project or directory).
Cut		Use this command to move the selected item or items from the current location to the clipboard.
Copy		Use this command to copy the selected item or items to the clipboard.
Copy Path(s)		Use this command to copy the full path(s) of the selected item or items to the clipboard.
Copy Relative Path		Use this command to copy a relative path to the selected item to the clipboard.
Paste		Use this command to insert the contents of the clipboard into the selected location.
Jump to Source		Use this command to open the selected file in the editor. If the file is already open, the corresponding editor tab will become active.
Find Usages		Use this command to find the usages of the selected item. (The Find Usages dialog will open.)
Find in Path		Use this command to perform a text search. (Find in Path dialog will open.)
Replace in Path		Use this command to perform text search-and-replace. (Replace in Path dialog will open.)
Refactor		Use this command to perform one of the refactorings available for the selected item.
Add to Favorites		Use this command to add the selected item to an existing or new list of favorite items. See Managing Your Project Favorites .
Show Thumbnails		Use this command to view thumbnails for image files located in the selected directory. (The Thumbnails tool window will open.)
Reformat Code		Use this command to reformat the source code in the selected file or in all files in the current directory. (The Reformat Code dialog will open.) See also, Reformatting Source Code .
Delete		Use this command to delete the selected item. Use with care!
Change Dialect (<CurrentDialect>)		For SQL files and database consoles: change the SQL dialect associated with the file or console.
Run '<item_name>'		For an SQL file or database console: execute all the statements contained in the selected file or console.
Local History		Use this command to view local history for the selected file or directory, or to create a label for the current version of your project. See Local History and Using Local History .
Synchronize '<item_name>'		Use this command to synchronize the selected item with its version saved in the file system. (If you change a file or directory contents externally, PyCharm, under certain circumstances, may not be aware of the corresponding changes unless you use this command.)
Show in Explorer		Use this command to open a file browser (e.g. Windows Explorer or Finder) and show the selected item there.
File Path		Use this command to open the File Path menu. This menu shows the path from the file system root to the selected element with individual directories as the menu items. When you select an item in this menu (e.g. a directory), a file browser (e.g. Windows Explorer or Finder) opens, and the selected item is shown there.
Compare With		Use this command to compare the selected file or directory with another file or directory. Select the other file or directory the dialog that opens . See Comparing Folders and Differences Viewer for Folders .
Compare File with Editor		Use this command to compare the selected file with the file open on an active editor tab. See Comparing Files and Differences Viewer for Files .
Mark Directory As		Use this command to make the selected directory a source root or a test source root, to make the directory excluded, etc. The necessary category for the directory is selected from the submenu. For more information on the categories for directories, see Content Root .
Diagrams	 or 	Use this command to open a diagram (e.g. a UML diagram) for the selected item. For more information, see Diagram Reference .

File status highlights

PyCharm uses colors to denote VCS file status in the Project tool window. The following table presents information about the meaning of the colors.

Color	File Status	Description
Black	Up to date	File is unchanged.  test1.txt
Gray	Deleted	File is scheduled for deletion from the repository.  privatent.txt
Blue	Modified	File has changed since the last synchronization.  test1.txt
Green	Added	File is scheduled for addition to the repository.  test1.txt
Violet	Merged	File is merged by your VCS as a result of an update.  test1.txt
Brown	Unversioned	File exists locally, but is not in the repository, and is not scheduled for adding.  Test.xml
Olive	Ignored	File will be ignored in any VCS operation.  Test.html
Light brown	Hijacked	File is modified without checkout . This status is valid for the files under Perforce, ClearCase and VSS.  HelpTOC.xml
Red	Merged with conflicts	During the last update, file was merged with conflicts.  test1.txt
Lilac	Externally deleted	File is deleted locally, but was not scheduled for deletion, and still exists in the CVS repository.  test1.txt
Dark cyan	Switched	The file is taken from a different branch than the whole project. This status is valid for CVS and SVN.  test1.txt

Python Console

Tools | Python Console

In this section:

- [Important notes](#)
- [Toolbar](#)
- [Context menu commands](#)

Important notes

- Python console appears as a tool window every time you choose the corresponding command on the Tools menu.
- An interactive console is divided into two panes: the lower pane serves for user input, the upper pane displays results and messages.
- Use up and down arrow keys to browse through the history of executed commands, and repeat the desired ones.
- Python console is available for both local and remote interpreters.
- Color and font scheme of a console is configurable on the Console Colors and Console Fonts pages of the [Colors and Fonts](#) editor settings.

Toolbar

Item
Tooltip
and
shortcut

	Rerun console	Click this button to terminate the current process and launch the new one. Ctrl+F5
	Stop	Click this button to stop the current process. Clicking the button once invokes <code>soft kill</code> allowing the application to catch the <code>SIGINT</code> event and perform graceful termination (on Windows, the <code>Ctrl+C</code> event is emulated). After the button is clicked once, it is replaced with  indicating that subsequent click will lead to force termination of the application, e.g. on Unix <code>SIGKILL</code> is sent. Ctrl+F2
	Close	Click this button to close the selected tab of the Run tool window and terminate the current process. Ctrl+Shift+F4
	Execute Current Statement	Click this button to execute the command at caret, entered in the input pane of the console. Enter
	Help	Use this icon or shortcut to open the corresponding help page. F1
	Show variables	Click this button to show in a separate pane the variables declared in the console. Right-click on a variable in this pane reveals a context menu.
	Browse History	Use this icon or shortcut to open a dialog that shows all the statements that you have run for the corresponding data source. See also, Executing auto-memorized statements . Ctrl+Alt+E
	Attach Debugger	Attaches the debugger process to the console.
	New Console	Click this button to start a new console session.

Context menu commands

Command
Shortcut
Description

Compare with Clipboard		Show selection in the console and contents of the Clipboard in the Differences viewer .
 Create Gist		Choose this command to create a Git Gist .
 Clear All		Choose this item on the context menu to delete all messages from the upper part of the console.
Paste	Ctrl+V	Insert the last entry of the Clipboard next to the caret position in the lower part of the console.
Paste from History	Ctrl+Shift+V	Insert selected entry of the Clipboard next to the caret position in the lower part of the console.
Paste Simple	Ctrl+Shift+V	Insert selected entry of the Clipboard as plain text, next to the caret position in the lower part of the console.

Remote Host Tool Window

Tools | Deployment | Browse Remote Host

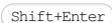
View | Tool Windows | Remote Host

Use this tool window to view the folder structure of the target FTP/FTPS/SFTP servers and the data uploaded to them.

Getting Access to the Remote Host Tool Window

- View | Tool Windows | Remote Host - the tool window can be accessed this way after you have opened it using the Tools | Deployment | Browse Remote Host command.
- The tool window is available only when the **Remote Hosts Access** bundled plugin is enabled. The plugin is active by default. If not, activate it in the [Plugins](#) page of the [Settings](#) dialog box.

Toolbar

Item	Tooltip and Shortcut	Description
Remote host		From this drop-down list, select the desired remote host configuration.
		Click the browse button to add a new server.
	Collapse All 	Click this button to have all nodes in the view collapsed.
	Refresh 	Click this button to refresh the view.
	Close 	Click this button to close the tool window.
	Help 	Click this button to show this reference page.

Context Menu

Item	Description
Upload Here	Choose this option to have the files from the currently opened project uploaded according to the selected configuration. If the action is invoked from the context menu of a file, the corresponding local file is uploaded. If the action is invoked from a folder, the entire folder is uploaded.
Download from here	Choose this option to download the selected file or folder to the currently opened. The existing local files will be updated and the missing files will be created.
New	Choose this option to create a new remote file or folder in the selected folder. The option is available only from the context menu of a folder.
Rename	Choose this option to rename the selected file or folder.
Delete	Choose this option to remove the selected file or folder.
Cut	Choose this option to copy the name of the selected file or folder to the clipboard and remove the file or folder from the tree.
Copy	Choose this option to copy the name of the selected file or folder to the clipboard.
Paste	Choose this option to insert the previously copied name of a file or a folder into the tree.
Edit Local File	When you select a file and choose this option, PyCharm opens its local copy in the editor and moves the focus to the corresponding. The option is available only when the project with the corresponding local file is currently opened in the editor.
Sync with Local	Choose this option to compare the selected remote folder with its local version. In the Differences Viewer for Folders that opens, explore the differences and synchronize the files, where applicable, as described in Comparing two folders in the Difference Viewer . See Comparing Deployed Files and Folders with Their Local Versions for details.
Compare with Local Version	Choose this option to compare the selected remote file with its local version. In the Differences Viewer for Files dialog box, that opens, explore the differences and apply them, if necessary, using the  and  buttons. For details, see Viewing Differences Between Files . See Comparing Deployed Files and Folders with Their Local Versions for details.
Edit Remote File	Choose this option to edit the selected file in the PyCharm editor without adding it to the currently opened project. See Editing Individual Files on Remote Hosts for details.
Copy Path	
Exclude Path	Choose this option to exclude the selected folder from upload/download, see Excluding Files and Folders from Upload/Download for details.
Copy Path	Choose this option to copy the absolute path to the selected file or folder on the server to the clipboard.

REST Client Tool Window

Tools | Test RESTful Web Service

View | Tool Windows | REST Client

Use the REST Client tool window for [testing a RESTful Web Service](#). The tool window is intended for composing and submitting test requests to Web service methods based on the service API, as well as for viewing and analyzing server responses.

Please note the following:

- View | Tool Windows | REST Client - the tool window can be accessed this way after you have opened it using the Tools | Test RESTful Web Service command.
- The tool window is available only when the **REST Client** bundled plugin is enabled. The plugin is active by default. If not, activate it in the [Plugins](#) page of the [Settings](#) dialog box.

On this page:

- [Common Request Settings](#)
 - [Toolbar](#)
- [Request Tab](#)
- [Cookies Tab](#)
- [Response Tab](#)
- [Response Headers Tab](#)

Common Request Settings

In this area, choose the request method and specify the data to compose the [request URI](#) from.

The server response code and the content length are shown in the upper-right corner of the REST Client tool window.

ItemDescription

HTTP method	In this drop-down list, specify the request method . The available options are: <ul style="list-style-type: none">- GET- POST- PUT- PATCH- DELETE- HEAD- OPTIONS
-------------	--

Host/port	In this text box, type the URL address of the host where the target Web service is deployed and the port it listens to. By default, the port number is <code>80</code> . If another port is used, specify it explicitly in the format <code><host URL>:<port number></code> .
-----------	---

Path	In this drop-down list, specify the relative path to the target method.
------	---

You can enter the entire URL address of a method to test in the Host/port text box. Regardless of the chosen **HTTP method**, upon pressing  PyCharm will split the URL address into the **host/port** and the path to the method. The extracted relative path will be shown in the Path text box and the extracted parameters will be added to the list in the Request Parameters pane of the Request tab.

Toolbar

ItemTooltip and shortcut

	Submit Request	Click this button to submit the generated test request to the server. <div style="background-color: yellow; padding: 5px;">Note If a server is not trusted, PyCharm shows a dialog box suggesting you to accept the server, or reject it. If you accept the server as trusted, PyCharm writes its certificate to the trust store. On the next connect to the server, this dialog box will not be shown.</div>
	Replay Recent Requests	Click this button to have a Recent Requests pop-up list displayed and select the relevant request. The fields are filled in with the settings of the selected request. Click the Submit Request button  .
	Export Request	Click this button to have the current request settings saved in an XML file so they are available in another PyCharm session. In the dialog box that opens, specify the name of the file to save the settings in and its parent folder. When necessary, you can retrieve the saved settings and run the request again.
	Import Request	Click this button to have the settings of a previously saved request retrieved from an XML file. In the dialog box that opens, select the relevant XML file.
	Generate Authorization Header	Click this button to open the Generate Authorization Header dialog box and specify your user name and password for accessing the target RESTful Web service through. Based on these credentials PyCharm will generate an authentication header which will be used in basic authentication . Learn more at http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm .
	Configure HTTP Proxy	Click this button to specify proxy server settings in the dialog box that opens.
	Close	Close the REST Client tool window.

Request Tab

Use this tab to specify the parameters to be passed to the service in the generated test request either through the query string for `GET` requests or through the request body for other request types. Also configure interaction between the client side and the Web service by specifying the format of data that the service and the client accept.

ItemDescription

Headers In this pane, specify the technical data included in the [request header](#). These data are passed through header fields and define the format of the input parameters ([accept](#) field), the response format ([content-type](#) field), the caching mechanism ([cache-control](#) field), etc. To add a field to the list, click Add **+**, then specify the field name in the Name text box and the field value in the Value drop-down list.

The set of fields and their values should comply with the Web service API. In other words, the specified input format should be exactly the one expected by the Web service as well as the expected response format should be exactly the one that the service returns.

For `accept`, `content-type`, and some other fields PyCharm provides a list of suggested values. Choose the relevant format type from the Value drop-down list.

Request Parameters In this pane, specify the parameters to be passed to the target method through a query string inside the URL. This approach is used for requests of the type `GET`. By default, the pane shows an empty list with one line.

- To add a parameter, click Add **+**, then specify the name of the parameter in the Name text box and the value of the parameter in the Value drop-down list.
- To delete a parameter from the list, select it and click Remove **-**.
- To suppress sending the specified query string parameters and disable the controls in the Request Parameters pane, press the Don't send anything toggle button .

The set of parameters and their types should comply with the Web service API, in particular, they should be exactly the same as the input parameters of the target method.

- select this check box This may be helpful, for example, if you want to test passing parameters through other request methods in the request body but still preserve the data typed in the Query string parameters pane.

Request Body The pane is disabled when the `GET`, `DELETE`, `HEAD`, or `OPTIONS` request method is selected. In this pane, specify the input parameters to be passed to the target method inside a [request message body](#).

- Empty: choose this option to send a request with an empty body.
- Text: choose this option to send a request with a string of parameters with values. Specify the parameters in the text box next to the option.
- File contents: choose this option to have the parameters inserted in the request body from a local text file. Specify the source text file in the File to send field.
- File upload(multipart/form-data): choose this option if you want to pass a binary file as a parameter, which requires that the file be converted. Specify the source binary file in the File to send field.

Cookies Tab

Use this tab to create, save, edit, and remove cookies, both received through responses and created manually. The **name** and **value** of a cookie is automatically included in each request to the URL address that matches the **domain** and **path** specified for the cookie, provided that the **expiry date** has not been reached.

The tab shows a list of all currently available cookies that you received through responses or created manually. The cookies are shown in the order they were added to the list. When you click a cookie, its details become editable and are displayed in text boxes.

ItemDescription

Name	In this fields, specify the name of the cookie to be included in the request.
Value	In this field, specify the value of the cookie to be included in requests.
Domain	In this field, specify the host and port the requests to which must be supplied with the name and value of the cookie.
Path	In this field, specify the path of the URL the requests to which must be supplied with the name and value of the cookie.
Expiry date	In this field, specify the expiry date of the cookie.
+	Click this button to add a new row to the list and define a new cookie in it.
-	Click this button to remove the selected cookie from the list.

Response Tab

Use this tab to view responses from the Web service. By default, responses are shown in the plain text form. Use the icons of the tab to have them displayed in the editor in the HTML, XML, and JSON formats.

ItemTooltip Description

	View as HTML	Click this button to open a new tab in the main editor window and display there the response as HTML.
	View as XML	Click this button to open a new tab in the main editor window and display there the server response as XML.
	View as JSON	Click this button to open a new tab in the main editor window and display there the server response as JSON.
	Open in browser	Click this button to view the response in your default Web browser.

Response Headers Tab

The tab shows the technical data provided in the [headers of Web service responses](#).

Run Tool Window

Run | Run

View | Tool Windows | Run

Alt+4

The Run tool window displays output generated by your application. If you are running multiple applications, each one is displayed in a tab named after the [run/debug configuration](#) applied.

The appearance of each tab depends on the type of the application being run and can include additional toolboxes and panes.

- Run Tool Window
 - [Run Toolbar](#)
 - [Context menu](#)
 - [Console Toolbar](#)
 - [Karma Server tab](#)
- [Test Runner Tab](#)
- [manage.py](#)
- [Jupyter Notebook](#)

The main toolbar of the Run tool window lets you rerun, stop, pause, or terminate an application. The following table contains descriptions of the buttons that are common for most applications.

Run Toolbar

Item
ItemTooltip
and
Description
and
shortcut

	Rerun	Click this button to stop the current application and run it again. When an application is stopped (■), this button toggles to ▶.
	Rerun	Click this button to rerun the current application. This button appears, when an application is stopped (■). When an application is running, this button toggles to  .
	Pause Output	Click this button to have the process output paused. Note that the button is not available for Run/Debug Configuration: Node JS , Run/Debug Configuration: Node JS Remote Debug , and Run/Debug Configuration: NodeUnit .
	Stop	Click this button to terminate the current process externally by means of the standard <code>shutdown</code> script. Clicking the button once invokes <code>soft kill</code> allowing the application to catch the <code>SIGINT</code> event and perform graceful termination (on Windows, the <code>Ctrl+C</code> event is emulated). After the button is clicked once, it is replaced with  indicating that subsequent click will lead to force termination of the application, e.g. on Unix <code>SIGKILL</code> is sent.
	Restore Layout	Click this button to to have the changes to the current layout abandoned and return to the default state.
	Pin	Use to pin or unpin the tab. If a tab is pinned, the results for the next command are shown on a new tab.
	Close	Click this button to close the selected tab of the Run tool window and terminate the current process. Ctrl+Shift+F4
	Help	Use this icon or shortcut to open the corresponding help page. F1

Context menu

Item Description

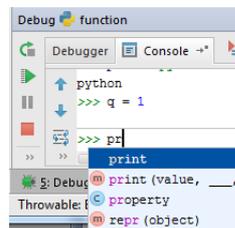
Console Toolbar

ItemTooltip
and
Description
and
shortcut

	Up/down the Stack Trace	Click this button to navigate up or down in the stack trace and have the cursor jump to the corresponding location in the source code. Ctrl+Alt+Up Ctrl+Alt+Down
	Use Soft Wraps	Click this button to toggle the soft wrap mode of the output.

	Scroll to the end	Click this button to navigate to the bottom of the stack trace and have the cursor jump to the corresponding location in the source code.
	Print	Click this button to send the console text to the default printer.
	Clear All	Click this button to remove all text from the console. This function is also available on the context menu of the console.
	Show Python prompt	If this button is pressed, you can enter commands in the console and view output .

Note that in the debug console, code completion is available:



It's also possible to scroll through the history of commands with the up and down arrow keys.

	Browse history	Press this button to show the Debug Console History dialog box, where one can view the console entries and navigate through them, using the arrow keys. Click OK to close the dialog box.
---	----------------	--

Ctrl+Alt+E

Karma Server tab

The tab is shown only when you run JavaScript unit tests using the Karma test runner. Use the tab to view and analyze information from the server. For details, see [Testing JavaScript with Karma](#).

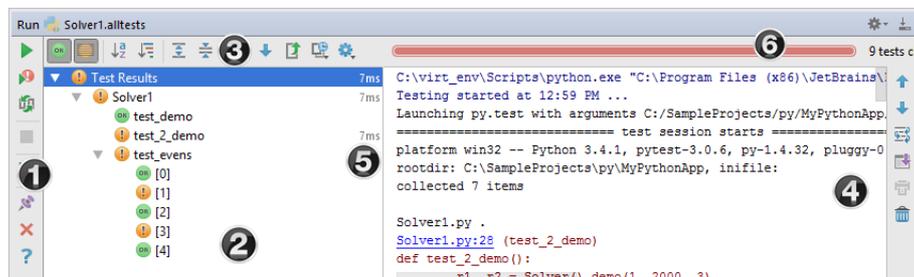
Test Runner Tab

In this section:

- [Basics](#)
- [Run toolbar](#)
- [Testing toolbar](#)
- [Test status icons](#)
- [Output pane](#)
- [Context menu commands](#)

Basics

The Test Runner tab opens in the [Run](#) tool window when a testing session begins, and features the same [toolbar buttons](#).



1. The [Run toolbar](#) is almost the same as that for the [Run](#) tool window, but features testing-specific buttons.
2. The left-hand pane shows the tree view of all tests within the current run/debug configuration.
 - The root node represents the test selected to run.
 - The nested nodes represent the hierarchy of test suites and test cases.
 - The leaf nodes represent the individual tests.

The status of each test is indicated by an [icon](#). Double-click a node to open the respective test class or test method in the editor.
3. The [testing toolbar](#) provides controls that enable you to monitor the tests and analyze results. Some of the commands are duplicated on the [context menus](#) of the test tree nodes.
4. The [Output](#) pane shows the output of the current test suit.
5. The inline statistics show the list of executed tests with the execution time of each test.
6. The color of the status bar indicates whether the tests have passed successfully. If at least one of the tests fails, the status bar turns red.

Run toolbar

Item **Tooltip** **Description**
and
Shortcut

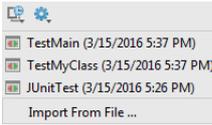
	Rerun	Click this button to rerun the current process. The process reruns always in the same console regardless of whether this console is pinned or not.
	Rerun Failed Tests	Click this button to have PyCharm execute failed tests. If you press (Shift) and click this button, you can choose whether you want to Run the tests again, or Debug, i.e. rerun the failed tests in the Debug mode.
	Toggle auto-test	If this button is pressed, the autotest-like runner is turned on. It means that the test in the current run configuration tab will automatically rerun on changing its source code. Otherwise, to rerun such test, you have to click the rerun button .
	Stop	Click this button to terminate the current process externally by means of the standard mechanisms.
	Restore Layout	Click this button to to have the changes to the current layout abandoned and return to the default state.
	Pin	When this button is pressed, the current tab will not be overwritten; instead, the results of the next command will be displayed in a new tab.
		Click this button to close the selected tab of the Run tool window and terminate the current process.
		Click this button to show reference.

Testing toolbar

Item **Tooltip** **Description**
and
Shortcut

	Show Passed	Click this button to show tests that passed successfully.
	Show Ignored	Click this button to show the ignored tests in the tree view of all tests within the current run/debug

configuration or test class.

	Sort alphabetically	Click this button to sort tests in alphabetical order.
	Sort by duration	Click this button to sort tests by duration.
	Expand All/Collapse All	Click these buttons to have all nodes in the tree view of tests expanded. These buttons are only available if the tested application contains more than test case.
	Previous/Next Failed Test	Click these buttons to navigate between the failed tests. Ctrl+NumPad Plus Ctrl+NumPad - Ctrl+Alt+Up / Ctrl+Alt+Down
	Import Test Results	Click this button to view the test results that you previously saved in an XML file or the results that PyCharm has kept in its internal history. The pop-up menu that opens shows a list of internally saved results of test sessions, each item is supplied with the name of the run configuration and a time stamp:  – To view the results of a testing session from the PyCharm history, select the item with the suitable run configuration and time stamp. – To load the previously exported results, choose Import from file and then choose the required XML file in the dialog box that opens. The loaded test results are shown in the tab and the name of the corresponding run configuration is displayed on the title bar. To re-run the tests from the loaded session, click  .

	Click this cog button to access the context menu with the following options: <ul style="list-style-type: none">– Track Running Test: turn this option on to monitor execution of the current test. If a test suite contains multiple tests, the tree view of tests expands to show sequential test methods, as they are executed.– Show Inline Statistics: turn this option on to have the statistics shown next to a test result, displaying the time used for executing each test.– Scroll to Stacktrace: turn this option on to have the console scroll to the beginning of the trace of the last failed test. If you click the root node (the test package) in the tree view with this option turned off, the console will show the very beginning of the test. This option is helpful when a test package contains multiple test classes and test methods. If some of the tests fail, you can scroll in the console to the beginning of a stack trace of an exception or assertion. <ul style="list-style-type: none">– Open Source at Exception: use this option to explore the results of a test that fails as an error, throwing an uncaught exception. If you double-click the failed test class or method in the tree view with this option turned on, the respective test class or method will open in the editor, with the caret placed at the line that caused the problem. <ul style="list-style-type: none">– Autoscroll to Source: turn this option on to have the currently selected test in the tree view synchronized with the editor automatically.– Select First Failed Test When Finished: turn this option on to have the first failed test automatically selected in the tree view upon completing the tests.
---	---

Test status icons

IconDescription

	Test error. This status is assigned to tests that caused an exception from the tested source code.
	Test failed. If at least one test receives this status, then all its parents are marked as failed.
	Test ignored.
	Test in progress.
	Test passed successfully.
	Test terminated. This status is assigned to tests that were cancelled by clicking the Stop button  . If at least one test receives this status, then all unfinished tests and their parents are marked as terminated.

Output pane

This pane shows output of each test, generated at runtime, including all the messages sent to the output stream, and the error messages. The following table shows the toolbar buttons and context menu commands available for the Output pane.

Item KeyboardDescription Shortcut

	Up the Stack Trace	Click this button to navigate up in the stack trace and have the cursor jump to the corresponding location in the source code.
	Down the Stack Trace	Click this button to navigate down in the stack trace and have the cursor jump to the corresponding location in the source code. Ctrl+Alt+Down Ctrl+Alt+Up
	Use Soft Wraps	Click this button to toggle the soft wrap mode of the output.

	Scroll to the end	Click this button to navigate to the bottom of the stack trace and have the cursor jump to the corresponding location in the source code.
	Print	Click this button to configure printing out the console output in the Print dialog box that opens.
Clear All		Choose this item on the context menu to have all messages for the selected test deleted.
Copy Content		Choose this item on the context menu to have the current contents of the Output pane placed to the Clipboard.
Compare with Clipboard		Choose this item on the context menu to invoke the Differences Viewer for Files which shows the current contents of the Clipboard in the left-hand pane and the contents of the Output pane for the selected test in the right-hand pane.

Context menu commands

Command	Keyboard shortcut	Description
---------	-------------------	-------------

View assertEquals Difference		Choose this command to show the Differences viewer for the strings being compared.
------------------------------	---	--

This command is only available when an assertion has failed.

manage.py

Tools | Run manage.py task

*.ipynb file toolbar | 

This tool window shows results of running the tasks of the `manage.py` utility.

ItemDescription



Click this button to run the previous task of the `manage.py` utility in a new tab.



Click this button to terminate the running process. After that, you can rerun it again.



Click this button to close the active tab.

Jupyter Notebook

*.ipynb file toolbar | ▶

ItemDescription



Click this button to run the notebook in a new tab.



Click this button to terminate the running process.



Click this button to close the active tab.

Trace Run Tab

The tab consists of a toolbar and three panes: Events Pane, Event Stack Pane, and Quick Evaluation Pane.

On this page:

- [Events Toolbar](#)
- [Events Pane](#)
 - [Context Menu of a Document Node](#)
 - [Context Menu of an Event or Script](#)
 - [Configuring the Range of Events to Capture](#)
 - [Defining a new event filter](#)
 - [Activating a filter](#)
 - [Adding an event to an exclusion filter on the fly](#)
- [Event Stack Pane](#)
 - [Context Menu of a Script or Function](#)
- [Synchronization between the Panes and the Editor](#)
- [Quick Evaluation Pane](#)
 - [Context Menu of Function Call Details](#)

Events Toolbar

Use the buttons on the toolbar to control the range of events to capture, configure their presentation, and navigate through the list of captured events.

Item
Tooltip
and
shortcut

	Expand all 	Click this button to have all the nodes in the list expanded.
	Collapse all 	Click this button to have all the nodes in the list collapsed.
	Up the Stack Trace	Click this button to navigate to the previous traced page in the stack trace.
	Down the Stack Trace	Click this button to navigate to the next traced page in the stack trace.
	Autoscroll to source	<p>Press this toggle button to have the list in the Events pane automatically synchronized with the Editor.</p> <ul style="list-style-type: none">- When the button is pressed: as soon as you click an event in the Events pane, the details of the event are displayed in the Event Stack pane and the script that is invoked by the event is opened in the editor automatically. <p>When you navigate through the Event Stack with the Autoscroll to Trace mode turned on, the corresponding files are also automatically opened in the editor with the calling functions highlighted.</p> <ul style="list-style-type: none">- When the button is released: the script that is invoked by the event is opened in the editor only upon double-clicking the event in the Event Stack pane.
	Capture Events	<p>Click this button to configure the range of events to be captured and shown in the Events list.</p> <p>By default, the <i>Spy-js</i> tool captures all events on all opened Web pages, excluding <i>https secure</i> web sites, unless you have specified a URL address explicitly in the run configuration. The Events pane of the <i>Spy-js</i> tool window shows all captured events. If for some reasons you do not want to have all events captured, you can suppress capturing some of them by applying user-defined event filters. When you click , the pop-up list shows all the available filters, the currently applied filter is marked with a tick. By default the Capture All predefined filter is applied.</p> <p>To stop capturing events without stopping the application, choose Mute All. The application is still running but the Events pane shows the last captured event. This is helpful if you want to analyze a script and therefore need it to be displayed in the Events pane instead of being removed as new events are captured.</p> <p>To define a custom event filter:</p> <ol style="list-style-type: none">1. Click , and then choose Edit Capture Exclusions from the list.2. In the <i>Spy-js Capture Exclusions Dialog</i> that opens, click the Add toolbar button  in the left-hand pane.3. In the right-hand pane, specify the filter name in the Exclusion name field and configure a list of exclusion rules. <p>To add a rule, click , the Add Condition to Exclusion dialog box opens. Type a pattern in the Value/pattern text box, in the Condition type drop-down list specify whether the pattern should be applied to event types or script names. Note, that glob pattern matching is used. When you click OK, PyCharm brings you to the <i>Spy-js Capture Exclusions Dialog</i>.</p> <p>To edit a rule, select it in the list, click , and update the rule in the dialog box that opens. To remove a rule, select it in the list and click .</p> <p>To activate a filter, set a tick next to the required filter in the list.</p>
		<p>Click this button to remove all or some events from tracing and have the corresponding trace files closed in the editor. Choose one of the following options on the drop-down list that is displayed:</p> <ul style="list-style-type: none">- Remove all: choose this option to cancel tracing of all captured events without closing the trace files in the editor.- Close all trace files: choose this option to have all trace files in the editor closed but keep tracing the corresponding event. To remove an event or script from tracing and close the corresponding trace files in the editor, choose Remove on the context menu of the event or script.- Remove all inactive: choose this option to remove all nodes for pages that are not active anymore (for example, because the pages

have been closed in browser).

- Save trace: choose this option to save an image of the current session. The `.json` files that store the calls and properties of the session are compressed into a `zip` archive. Upon request, when you choose Load trace, the `.json` files are extracted from the archive and loaded into `Spy-js`.
- Load trace: choose this option to have a previously saved image of a tracing session loaded into `Spy-js`. The `.json` files that store the calls and properties of that session are extracted and imported.
Note that a loaded image does not restore the session because no scripts are actually executed. All you can do is analyze the flow and properties of previously executed code.
- Close all trace files on session stop: choose this option to have all trace files in the editor closed when you click the Stop button  to stop the tracing session externally by means of the standard `shutdown` script.
Clicking the button once invokes `soft kill` allowing the application to catch the `SIGINT` event and perform graceful termination (on Windows, the `Ctrl+C` event is emulated). After the button is clicked once, it is replaced with  indicating that subsequent click will lead to force termination of the application, e.g. on Unix `SIGKILL` is sent.
- Enable `Spy-js` autocomplete and magnifier: choose this option to have the basic completion list expanded with runtime data (`Spy-js` autocomplete) and to get the possibility to evaluate expressions without actually running a debugging session (`Spy-js` magnification).
By default, the functionality is turned off.
The term `Spy-js` autocomplete denotes expanding the `basic completion list` with suggestions retrieved from the runtime data. The `Spy-js` autocomplete functionality is available from `source` files for the code that has already been executed (highlighted green in the corresponding `trace` file).

When you position the caret at a symbol in the source file and press `Ctrl+Space`, `Spy-js` retrieves data from the browser or from the running NodeJS application and merges it with the basic completion list according to the following rules:

1. If an object both is present on the basic completion list and is retrieved from the runtime, the variant that provides more information about parameters, attributes, their type, etc. remains on the list.
2. Objects retrieved by `Spy-js` are shown on top of the list and marked with the  icon. If a retrieved object is specific for a browser, the object is marked with the  icon and with the icon of this browser.

The term `Spy-js` magnification denotes `evaluating expressions` without actually running a debugging session. When you click the expression in question or position the caret at it and press `Ctrl+Alt+F8`, a tooltip is displayed below the expression showing the expression value. If `Spy-js` retrieves several values, click  icon in the tooltip to expand the list of values.

The `magnification` functionality is available from `source` files for both executed and not yet executed code.

- `Spy-js` supports `source maps`, which means that you can now jump from the Event Stack pane right to the original source code in `ECMAScript 6`, `TypeScript` or `CoffeeScript` and observe what code fragments were executed. Use the following options to configure the way source maps are treated:
 - Enable source map look-up: choose this option to enable navigation to the `ECMAScript 6`, `TypeScript` or `CoffeeScript` source code using the source maps generated during compilation.
 - Enable source map generation: choose this option to generate source maps for everything to map the instrumented code. Choose this option if you are going to debug the original code in `Chrome Dev Tools` or `Firefox FireBug` development tools.
 - Always open source mapped trace if available: choose this option to have `Spy-js` try to open the `mapped trace` file when you invoke navigation from an event to its caller.

Events Pane

The pane shows a tree of captured events. The top-level nodes represent `documents` that are Web pages involved in tracing. When you hover the mouse over a `document`, PyCharm displays a tooltip with the URL address of the `document`, the browser in which it is opened, and the operating system the browser is running on. The `document` node is also supplied with an icon that indicates the browser in which it is opened.

Under each `document` node, events detected on the page and scripts started from it are listed. Events of the same type are grouped into visual containers. The header of a container displays the name of the events grouped in it, the average execution time across all the events within the container, and the number of events inside the container. You can expand each node and inspect the individual events in it.

Script file names have different colour indicators to help distinguishing between them when working with the Event Stack pane. By hovering your mouse over a script file name, you can see the full script URL.

Once an event is clicked, its call stack is displayed in the Event Stack pane. The stack is represented by a tree of function calls.

Context Menu of a Document Node

MenuDescription item

Remove	Choose this option to cancel tracing all the scripts on the selected page and remove the selected node with all the items under it from the Events pane. All the currently opened trace files remain opened in the editor.
Remove all children	Choose this option to delete the items under the selected page but keep tracing it so that new events from the page are still received. The document node itself remains in the Event pane, and all the currently opened trace files remain opened in the editor.
Remove and close trace file(s)	Choose this option to cancel tracing all the scripts on the selected page, remove the selected node and all the items under it from the Events pane, and close the corresponding trace files in the editor.
Close trace file(s)	Choose this option to close all the currently opened trace files that correspond to the selected document node and items under it. The document node and the items under it remain in the Events pane.
Refresh the page in browser	Choose this option to reload the page that corresponds to the selected document node. Tracing of the selected node is abandoned, a new document node for tracing the same page is created, and the old node becomes <code>inactive</code> .
Try closing the page in browser	Choose this option to close the page that corresponds to the selected node. Tracing of the selected node is abandoned, and the node becomes <code>inactive</code> .
Show application dependency diagram	Choose this option to build a diagram with the dependencies within the entire application. <ul style="list-style-type: none">- The diagram is opened in a separate editor tab. The nodes in the diagram represent your project files, while the edges represent the fact that there's one or more functions in the source file that invoke functions in the target file.- To examine the details of a node or an edge, select the node or the edge in question and view its Details tree in a dedicated pane in the upper right-hand corner of the editor. The pane displays the connecting function combinations, along with event(s) the calls are made within and the number of calls made.

Context Menu of an Event or Script

MenuDescription item

Mute event	Choose this option to add an event to an exclusion filter on the fly.
Mute script	Choose this option to add a script to an exclusion filter on the fly.
Remove	Choose this option to cancel tracing the selected event or script, remove the selected item from the Events pane, but leave the corresponding trace files opened in the editor.
Add label	Choose this option to set a timestamp label. Timestamp labels help you to analyze your code execution within a specific period of time. For example, you can set two timestamp labels and view which events were captured between them. Or on the contrary, you can locate the events that were not captured within a certain period of time although you expected them to be and thus detect performance problems.
Show event dependency diagram	Choose this option to build a diagram with the dependencies in which the event selected event is involved. <ul style="list-style-type: none">– The diagram is opened in a separate editor tab. The nodes in the diagram represent your project files, while the edges represent the fact that there's one or more functions in the source file that invoke functions in the target file.– To examine the details of a node or an edge, select the node or the edge in question and view its Details tree in a dedicated pane in the upper right-hand corner of the editor. The pane displays the connecting function combinations, along with event(s) the calls are made within and the number of calls made.

Configuring the Range of Events to Capture

By default, the **Spy-js** tool captures all events on all opened Web pages, excluding **https secure** web sites, unless you have specified a URL address explicitly in the run configuration. The Events pane of the Spy-js tool window shows all captured events. If for some reasons you do not want to have all events captured, you can suppress capturing some of them by applying user-defined event filters. All the available filters are listed upon clicking the Capture Events button  on the toolbar, the currently applied filter is marked with a tick. By default the Capture All predefined filter is applied.

To stop capturing events without stopping the application, choose Mute All. The application is still running but the Events pane shows the last captured event. This is helpful if you want to analyze a script and therefore need it to be displayed in the Events pane instead of being removed as new events are captured.

You can define new custom filters or add event patterns to existing filters on the fly.

Defining a new event filter

1. Click the Capture Events button  on the toolbar, and then choose Edit Capture Exclusions from the list.
2. In the **Spy-js Capture Exclusions Dialog** that opens, click the Add toolbar button  in the left-hand pane.
3. In the right-hand pane, specify the filter name in the Exclusion name field and configure a list of exclusion rules.
 - To add a rule, click , the Add Condition to Exclusion dialog box opens. Type a pattern in the Value/pattern text box, in the Condition type drop-down list specify whether the pattern should be applied to event types or script names. Note, that [glob pattern matching](#) is used. When you click OK, PyCharm brings you to the **Spy-js Capture Exclusions Dialog**.
 - To edit a rule, select it in the list, click , and update the rule in the dialog box that opens. To remove a rule, select it in the list and click .

Activating a filter

- Click  and set a tick next to the required filter in the list. If no filters are configured or none of the available filters fits the task, create a new filter as described above.

Adding an event to an exclusion filter on the fly

While navigating through the tree of already captured events in the Events pane, you may come across some events or scripts that you definitely do not want to trace. You can create a filter as described above but in this case you will have to leave the pane. With PyCharm, you can create an exclusion rule based on any event or script, as soon as you have detected such event or script, right from the Events pane. The rule will be either added to the currently applied filter or a new filter will be created if the current setting is Capture All.

- To add an event to an exclusion filter on the fly, select the event to exclude and choose Mute <event name> event or Mute <script name> file. If a user-defined filter is currently applied, the new rule is added to it silently. If Capture All is currently active, the **Spy-js Capture Exclusions Dialog** opens, where you can create a new filter based on the selected event or script or choose an existing filter and add the new rule to it.

Event Stack Pane

Once an event in the Events pane is clicked, its call stack is displayed in the Event Stack pane. The stack is represented by a tree of function calls. Each tree node represents the invoked function. Node text contains the total execution time, the script file name and the function name. When you click a node, the Quick Evaluation pane shows additional function call details, parameter values and return value, occurred exception details if there was one during the function execution.

The pane is synchronized with the editor, so you can navigate from an item in the stack tree to the corresponding **trace file** or **source file**.

- A **trace file** is a write-protected prettified version of the script selected in the Events pane or the script whose function is double clicked in the Event Stack pane. A **trace file** is named `<file name>.js.trace`. When you double click an item in the stack tree or select it and choose Jump to Trace on the context menu of the selection, the corresponding **trace file** opens in the editor with the cursor positioned at the clicked function. Another approach is to press the Autoscroll to Trace toggle button and select various stack nodes. In this case, the trace file opens when you click an event or script in the Events pane. You can not only jump to a function but also to the place in the code where it was called from. To do that, select the required item and choose Jump to Caller

on the context menu.

The contents of the file are highlighted to display the code execution path of the selected stack node.

- When you are tracing an application with **ECMAScript6**, **CoffeeScript**, and **TypeScript** code, **Spyjs** also generates **mapped trace files**. These are **EcmaScript 6**, **TypeScript**, or **CoffeeScript** trace files with the extensions `.ts.trace`, `.coffee.trace`, or `.js.trace`. The fragments of code in these files are highlighted as if they were really executed.
 - You can also navigate to the **source file** displayed as is, without prettifying, by selecting an item in the Event Stack pane and choosing **Jump to Source** on the context menu of the selection. If the traced site is mapped with a PyCharm project, PyCharm detects the corresponding local file according to the mapping and opens this file in the editor. If you are tracing a site that is not mapped to a PyCharm project, PyCharm opens the read-only **page source**, just as if you chose **View Page Source** in the browser.
- When the traced site is mapped with a PyCharm project, PyCharm opens the **source file** on any attempt to edit the opened **trace file**.

Context Menu of a Script or Function

ItemDescription

Jump to Caller Choose this option to navigate to the fragment in the **trace file** from where the currently selected item was called. When you are tracing an application with **ECMAScript6**, **CoffeeScript**, and **TypeScript** code, PyCharm opens either the **trace JavaScript file** or the **mapped trace file** (**TypeScript**, **CoffeeScript**, or **ECMAScript6**):

- If the **Always open source mapped trace if available** option is selected, the corresponding mapped trace file opens.
- If the **Always open source mapped trace if available** option is not selected, the **JavaScript trace file** opens.

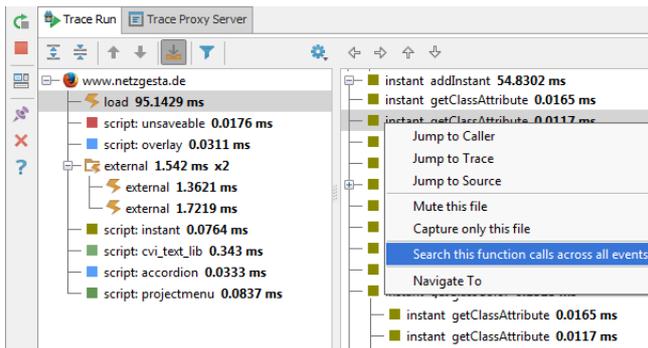
Jump to Trace Choose this option to navigate to the definition of the currently selected item in the **trace file**.

Jump to Source Choose this option to navigate to the definition of the currently selected item in the **source file**.

Mute this File Choose this option to add the selected script to an exclusion filter on the fly, see [Configuring the Range of Events to Capture](#).

Capture only this file

Search this function calls across all events Choose this option to navigate between the calls of a function within the whole trace (across all the traced events). This means that if you are tracing 5 pages in the browser and the Events pane, accordingly, shows 5 document nodes, PyCharm searches for the calls of the selected function under all these nodes and displays the number of found calls in the Status bar. The number of found calls is displayed in the Status bar, and the toolbar shows four previously hidden navigation chevron buttons.



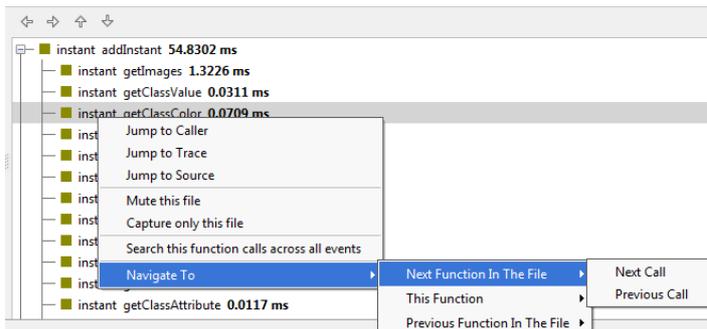
Use the chevron buttons to navigate within the found calls:

- To jump to the first detected call, click ⏪.
- To jump to the last detected call, click ⏩.
- To jump to the next detected call, click ⏴. The Status bar shows a message: `Occurrence <number> of <total number of detected calls>`
- To jump to the previous detected call, click ⏵.

The search results are reset and the search toolbar is hidden when you invoke another advanced search or navigation.

Also keep in mind that the number of call occurrences is calculated when you choose the **Search this function calls across all events** option. As you analyze the detected calls, the time passes, new events are captured, and the first detected call can happen to be already removed from the stack which means that it is no longer available for navigation.

Navigate to Use the options under this item to move through the whole stack based on calls and locate the functions that have not been called, that is, locate the fragments of code that have not been executed and analyze the reason for them to be skipped. The following six actions are available: move to the next/previous call of the next/current/previous function in a trace file. The full list of actions is available from the context menu in the Event Stack pane. Moving to the next and previous calls of the selected function, to the previous call of the previous function, and to the next call of the next function are also available from the navigation toolbar of the Event Stack pane.



When you choose one of these actions, the cursor jumps to the call in the stack. If the **Autoscroll to Trace** toggle button is pressed, the corresponding trace

file opens automatically with the cursor positioned at the call.

Synchronization between the Panes and the Editor

The Events and Event Stack panes are synchronized: when you click an event or script in the Events pane, its call stack is displayed in the Event Stack pane. To have also the corresponding trace file opened in the editor, press the Autoscroll to Trace toggle button on the toolbar.

The Event Stack pane is synchronized with the editor: when you click an item in the stack tree twice, the corresponding trace file opens in the editor with the cursor positioned at the clicked function.

To synchronize the Events pane directly with the editor, press the Autoscroll to Trace toggle button on the toolbar. In this case, as soon as you click a node in the Events pane, its call stack is displayed in the Event Stack pane and the corresponding trace file is opened in the editor. With the Autoscroll to Trace mode turned on, when you navigate through the Event Stack the corresponding files are also automatically opened in the editor with the corresponding functions highlighted.

Quick Evaluation Pane

When you click a node in the Event Stack pane, the Quick Evaluation pane shows additional function call details, parameter values and return value, occurred exception details if there was one during the function execution.

Context Menu of Function Call Details

The context menu is available from all items displayed in the pane.

ItemDescription

Inspect	Choose this option to open the Inspect dialog box.
Copy Value	Choose this option to copy the value of the selected node to the clipboard.
Compare Value with Clipboard	Choose this option to open the Differences Viewer for Files which displays the value of the selected node and the value in the clipboard so you can compare them.
Copy Name	Choose this option to copy the name of the selected node to the clipboard.

Spy-js Capture Exclusions Dialog

The dialog box opens when you click the Capture Events button  toolbar button in the Events pane of the [Spy-js Tool Window](#). In this dialog box, define custom filters to configure the range of events displayed and traced in the tool window. See [Tracing with Spy-js](#) for details.

The dialog box consists of two panes. The left-hand, Exclusions, pane shows a list of already existing user-defined event filters. The right-hand, Exclusion, pane shows the details of the filter selected in the Exclusions pane.

ItemDescription

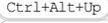
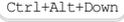
Exclusions pane	<p>The pane shows a list of all currently available user-defined filters.</p> <ul style="list-style-type: none">- To define a new event filter, click  on the toolbar, then specify the filter name and exclusion rules in the Exclusion pane.- To create a filter based on an already existing one, make a copy of the source filter by selecting it and clicking , and then rename and edit the copy as required in the Exclusion pane.- To temporarily disable a filter, clear the check box next to it.- To remove a filter from the list, select it and click .
Exclusion pane	<p>On this pane, configure custom event filters. For each filter, specify its name and create a list of exclusion rules.</p> <ul style="list-style-type: none">- To add a rule, click , the Add Condition to Exclusion dialog box opens. Type a pattern in the Value/pattern text box, in the Condition type drop-down list specify whether the pattern should be applied to event types or script names. Note, that glob pattern matching is used. When you click OK, PyCharm brings you to the Spy-js Capture Exclusions Dialog.- To edit a rule, select it in the list, click , and update the rule in the dialog box that opens.- To remove a rule, select it in the list and click .

Trace Proxy Server Tab

The tab consists of a toolbar and the console area that shows system information and error messages, information about the status of the **Spy-js** session, the availability of the proxy server, the currently active frames ("workers"), etc.

Trace Proxy Server Toolbar

Item **Tooltip** **Description**
and
shortcut

	Stop Trace Proxy Server	Click this button to terminate the currently running process externally by means of the standard <code>shutdown</code> script.
	Up/Down the Stack Trace	Use these buttons to navigate through the stack trace from one reported error to another. These buttons are available only when errors are reported, for example, if the configuration file is corrupted.  
	Use Soft Wraps	Click this button to toggle the soft wrap mode of the output.
	Scroll to the end	Click this button to navigate to the bottom of the stack trace and have the cursor jump to the corresponding location in the source code.
	Print	Click this button to send the console text to the default printer.
	Clear All	Click this button to remove all text from the console. This function is also available on the context menu of the console.

Structure Tool Window, File Structure Popup

Structure tool window

This tool window displays the structure of a file currently opened in the editor and having the focus, or selected in the Project tool window.

For diagrams, this tool window shows the diagram preview.

View | Tool Windows | Structure

Alt+7

File Structure pop-up window

This pop-up window displays the structure of a file, currently opened in the editor and having the focus.

Navigate | File Structure

Ctrl+F12

Both views help quickly navigate through the files' structure. Refer to the section [Navigating with Structure Views](#).

This section describes the buttons on the title bar of the tool window and the options on the context menu of the title bar. Turn these options on and off to have elements of certain types hidden or shown and configure the way they are presented.

The buttons on the title bar are common for all language contexts. The set of options on the context menu depends on the context.

- [Title Bar](#)
- [Python](#)
- [HTML, XML](#)
- [JavaScript, TypeScript, CoffeeScript](#)

TIP Refer to the section [Symbols](#) for the member icons in the tree view of a file in the Structure tool window.

Title Bar

The buttons on the title bar are common for all language contexts.

ItemTooltipDescription

	Collapse All Ctrl+NumPad -	Click this button to have all the nodes in the tool window collapsed.
	Expand All Ctrl+NumPad Plus	Click this button to have all the nodes in the tool window expanded.
		Click this button to open the context menu and configure the appearance of the tool window, its viewing mode , and the way it presents the structure of the current file by turning the menu items on or off.
	Hide Shift+Escape	Click this button to hide the tool window.

Python

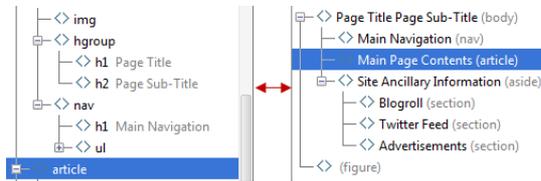
IconTooltipDescription

	Sort Alphabetically	Click this button to have the elements within a class sorted alphabetically.
	Show Inherited	Click this button to display all the methods and fields inherited by the current class and accessible from it. The inherited members are displayed gray to distinguish them from the members defined in the current class.
	Show Fields	Click this button to have all fields (properties) shown in the tree.
	Collapse All Ctrl+NumPad -	Click this button to have all the nodes in the tool window collapsed.
	Expand All Ctrl+NumPad Plus	Click this button to have all the nodes in the tool window expanded.
	Autoscroll to Source	Click this button to enable automatic navigation to the line of source code that corresponds to the selected node when the focus switches to the editor.
	Autoscroll from Source	Click this button to have PyCharm automatically move the focus in the Structure tool window to the node that corresponds to the code where the cursor is currently positioned in the editor.

HTML, XML

IconTooltipDescription

	Sort Alphabetically	Click this button to have the elements within a class sorted alphabetically.
	HTML5 Outline	Click this button on to view HTML 5 outline of a HTML file:



	Collapse All Ctrl+NumPad -	Click this button to have all the nodes in the tool window collapsed.
	Expand All Ctrl+NumPad Plus	Click this button to have all the nodes in the tool window expanded.
	Autoscroll to Source	Click this button to enable automatic navigation to the line of source code that corresponds to the selected node when the focus switches to the editor.
	Autoscroll from Source	Click this button to have PyCharm automatically move the focus in the Structure tool window to the node that corresponds to the code where the cursor is currently positioned in the editor.

JavaScript, TypeScript, CoffeeScript

IconTooltipDescription

	Sort Alphabetically	Click this button to have the elements within a class sorted alphabetically.
	Group Methods by Defining Type	Click this button to have all the methods that override/implement the methods of a particular class/interface grouped under the node that corresponds to this class/interface.
	Show Fields	Click this button to have all fields (properties) shown in the tree.
	Show Inherited	Click this button to display all the methods and fields inherited by the current class and accessible from it. The inherited members are displayed gray to tell them from the members defined in the current class.
	Collapse All Ctrl+NumPad -	Click this button to have all the nodes in the tool window collapsed.
	Expand All Ctrl+NumPad Plus	Click this button to have all the nodes in the tool window expanded.
	Autoscroll to Source	Click this button to enable automatic navigation to the line of source code that corresponds to the selected node when the focus switches to the editor.
	Autoscroll from Source	Click this button to have PyCharm automatically move the focus in the Structure tool window to the node that corresponds to the code where the cursor is currently positioned in the editor.

Thumbnails Tool Window

Project tool window | context menu | Show Thumbnails

Ctrl+Shift+T

The Thumbnails tool window provides the functions similar to those of an image browser. It shows thumbnails for folders and image files, and lets you perform related navigation and image management tasks.

To open the tool window, use the [Show Thumbnails command](#) in the [Project tool window](#) ([Ctrl+Shift+T](#)). Then, unless you close the tool window, you can show and hide it as described in the section [Manipulating the Tool Windows](#).

- [Title bar context menu](#)
- [Title bar icons](#)
- [Toolbar icons](#)
- [Content pane: context menu commands](#)

Title bar context menu

The title bar context menu provides the options for controlling the tool window viewing modes. It also contains the commands for associating the tool window with a different tool window bar, resizing and hiding the tool window.

To access the menu, right-click the window name (Thumbnails).

Note that most of the menu options may alternatively be accessed by means of the [title bar icons](#).

Item ShortcutDescription

Pinned, Docked, Floating, Split Mode	These options let you control general appearance and behavior of the tool window, see Viewing Modes .
Move to	To associate the tool window with a different tool window bar , select this command, and then select the destination tool window bar (Top, Left, Bottom or Right).
Resize	To resize the tool window by moving one of its borders, select this command, and then select the necessary Stretch to option. Note that this command is not available in the floating mode.
Hide	Shift+Escape Use this command to hide the tool window.

Title bar icons

Item ShortcutDescription

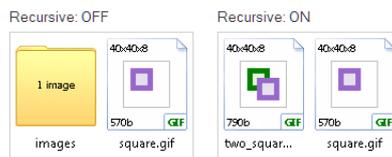
	Use this icon to open the menu for changing the tool window viewing modes .
	Shift+Escape Use this icon or shortcut to hide the tool window . When used in combination with the Alt key, clicking this icon hides all the tool windows attached to the same tool window bar .

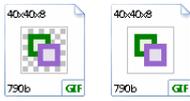
Toolbar icons

The toolbar icons provide access to the most frequently used functions available in the tool window. The same and other functions are available as [context menu commands](#) in the content pane.

Item ShortcutDescription

	Backspace Use this icon or shortcut to move one level up in the folder hierarchy. The path to the current folder is shown on the title bar to the right of the tool window name.
	Ctrl+Alt+NumPad Plus Use this icon or shortcut to turn the Recursive option on or off. If this option is off, subfolders of the current folder and image files located in the root of the current folder are shown. If this option is on, the subfolders are not shown; the image files located in the current folder and all its subfolders are shown. The following picture shows the contents of the same folder with the Recursive off and on.
	Use this icon to show or hide a "chessboard". The "chessboard" (a checkered area) is shown underneath the images so that you can see transparent image areas for .gif and .png files. The following picture shows the same thumbnail with the "chessboard" shown and hidden.





Ctrl+F4

Use this icon to close the tool window.

Content pane: context menu commands

When you right-click an item in the content pane, the context menu for this item is shown.

The following table lists and briefly explains the main context menu commands for thumbnails. Other commands, functionally, are similar to those in the [Project tool window](#).

Item	Shortcut	Description
Browse	Enter	For a folder: use this command to view the folder contents (subfolders and images). The same command may alternatively be accessed by double-clicking a folder.
Jump to Source	F4 or Enter	For an image file: use this command to view the selected file in the editor. The same command may alternatively be accessed by double-clicking an image thumbnail.
Level up	Backspace	Use this command to move one level up in the folder hierarchy.
Recursive	Ctrl+Alt+NumPad Plus	Use this command to turn the Recursive option on or off.
Show or Hide Chessboard		Use this command to show or hide the chessboard .
Close thumbnails	Ctrl+F4	Use this command to close the tool window.
Jump to External Editor	Ctrl+Alt+F4	For an image file: use this command to open the image in an external editor.

TODO Tool Window

View | Tool Windows | TODO

Alt+6

PyCharm scans your project for comments in the source code that match the TODO patterns defined in the [TODO dialog](#) and displays results in the TODO tool window.

The TODO tool window is marked with the icon  and consists of the following tabs:

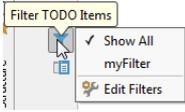
- Project tab that show the TODO items for the whole project.
- Current File tab.
- Scope Based tab that enables viewing TODO items pertaining to a certain scope, selected from the drop-down list, and ignoring the other items.
- Current Changelist, if version control is enabled.

This tool window helps you view, sort and group the TODO items in a convenient way, navigate to the source code and keep fixes under the version control.

On this page:

- [Toolbar buttons](#)
- [Context menu commands](#)
- [Title bar context menu and buttons](#)

Toolbar buttons

ItemTooltip and shortcut	Description
	Previous TODO Navigate to the previous TODO item. Ctrl+Alt+Up
	Next TODO Navigate to the next TODO item. Ctrl+Alt+Down
	Help Use this icon or shortcut to open the corresponding help page. F1
 	Expand all Use these buttons to have all nodes expanded or collapsed. Ctrl+NumPad Plus Collapse all Ctrl+NumPad -
	Autoscroll to Source Toggle the Autoscroll to source mode. When this button is pressed, every time the node is focused, the corresponding line of source code is highlighted in the editor.
	Filter TODO items Click this button to select the desired filter from the list, or invoke the TODO dialog and edit the list of TODO patterns and filters as required. 
	Preview Usages If this button is pressed, a pane to the right shows the source code of the selected file, with the corresponding TODO item highlighted.
Tip The following buttons are available in the Project and Scope Based tabs.	
	Ctrl+D If this button is pressed, the TODO items show under the corresponding module or library node.
	Ctrl+P If this button is pressed, the TODO items show under the corresponding packages.
	Ctrl+F If this button is pressed, the TODO items show as a flat list. Thus, if a package is somewhere deep within your project, you do not need to dig deep into the hierarchy.

Context menu commands

Item	Keyboard Shortcut	Description
Jump to Source	F4	Navigate to the selected usage in the source code.
Local History		Show Local History submenu for the selected search result. Refer to the Local Version Control procedures for

details.

<VCS>

Show menu of the VCS, associated with the directory. See Version Control [Procedures](#) and [Reference](#) for details.

Title bar context menu and buttons

The title bar context menu provides the options for controlling the tool window [viewing modes](#). It also contains the commands for associating the tool window with a different [tool window bar](#), resizing and hiding the tool window.

To access the menu, right-click the window title bar.

Note that most of the menu options may alternatively be accessed by means of the title bar buttons.

Toolbar icon	Context menu command	Description
	Pinned, Docked, Floating, Split Mode	These options let you control general appearance and behavior of the tool window, see Viewing Modes .
	Move to	To associate the tool window with a different tool window bar , select this command, and then select the destination tool window bar (Top, Left, Bottom or Right).
	Resize	To resize the tool window by moving one of its borders, select this command, and then select the necessary Stretch to option. Note that this command is not available in the floating mode.
	Hide	Use this command to hide the tool window.



Tip When used in combination with the  key, clicking this button hides all the tool windows attached to the same [tool window bar](#).

TypeScript Compiler Tool Window

On this page:

- [Current Errors Pane](#)
- [Project Errors Pane](#)
- [Toolbar](#)
- [Context Menu](#)
- [Console](#)

Current Errors Pane

The pane shows a list of all the discrepancies detected in the file which is opened in the active editor tab. At the top the full path to the file is displayed. For each discrepancy, PyCharm provides a brief description and information about the line number where the error occurred. In addition to this you can navigate to the code in question right from the corresponding error message by choosing Jump to Source on the context menu.

Project Errors Pane

The pane shows a list of all the discrepancies detected in all the files in the current project after you run the built-in compiler against the entire project by clicking  on the toolbar. The error messages are grouped by files in which they were detected. For each discrepancy, PyCharm provides a brief description and information about the line number where the error occurred. In addition to this you can navigate to the code in question right from the corresponding error message by choosing Jump to Source on the context menu.

Toolbar

The toolbar is common for the Current Errors and the Project Errors panes.

ItemTooltip Description

and Shortcut

	Compile Current File	Click this button to run the compiler against the entire file opened in the active editor tab. This approach is helpful if you do not want the compiler to wake up on any change as you edit your code and for this purpose the Track Changes check box on the TypeScript page of the Settings dialog box is cleared. If you click this button from the Project Errors pane, after compilation PyCharm switches to the Current Errors pane.
	Compile All	Click this button to run the compiler against all the TypeScript files in the current project. If you click this button from the Current Errors pane, after compilation PyCharm switches to the Project Errors pane.
	Close <code>Ctrl+Shift+F4</code>	Click this button to terminate the compiler and close the tool window.
	Expand all	Use these buttons to have all nodes expanded or collapsed.
	Collapse all <code>Ctrl+NumPad Plus</code>	
	Collapse all <code>Ctrl+NumPad -</code>	
	Clear All	Click this button to remove all the error messages from the currently active pane.

Context Menu

The context menu is common for the Current Errors and the Project Errors panes.

ItemDescription

Jump to source	Choose this option to open the file where the selected error occurred during compilation and navigate to the fragment of code which caused the error.
Copy	Choose this option to copy the selected error message with the information on the file, the line, and the column where the error occurred during compilation.

Console

The tab shows the log of the built-in compiler since the tool window was opened.

Version Control Tool Window

View | Tool Windows | Version Control

This tool window is available if version control is [enabled](#) for your project.

The tool window is marked with the icon  and accommodates several views, which display VCS-related information and allow you to manage changelists, perform VCS-specific actions, view changes made by other team members, etc.

The tool window consists of several tabs:

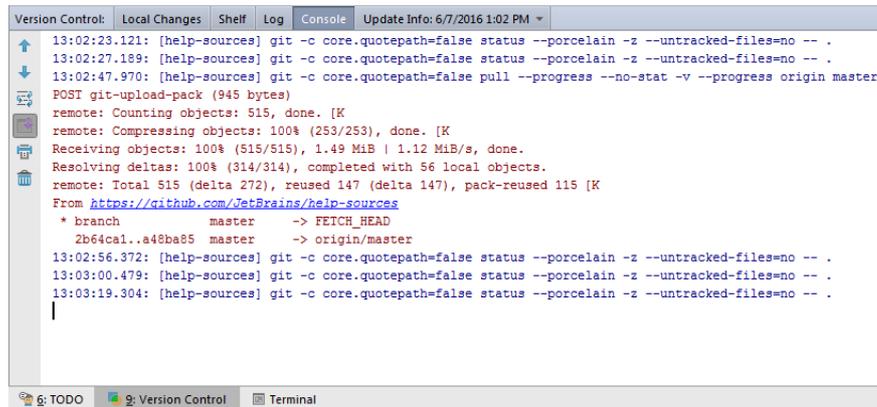
- [Console](#) tab: this tab shows the results of executing VCS-related commands.
- [Local Changes](#) tab: this tab is always present and shows the list of files that have been modified locally and have not been committed to the repository yet.
- [History](#) tab: this tab is added to the Version Control tool window when the Show History command is invoked through VCS | <specific_VCS>.
- [Integrate to Branch Info View](#) tab: this view is available after running integration with the Run status after update setting specified.
- [Log](#) tab: this tab is only available if you are using Git or Mercurial as your version control system. It shows all changes committed to all branches of the local and remote repositories, or to a specific branch or repository.
- [Repository and Incoming](#) tabs: the Repository tab shows the changes committed to the repository under the VCS roots within the current project. The Incoming tab shows the changes committed to the repository by other team members, and not checked out locally.
- [Shelf](#) tab: this tab is added to the tool window when you [shelve](#) a change or a changelist.
- [Update Info](#) tab: this tab becomes available when [local information is synchronized](#) with the server.

Console Tab

View | Tool Windows | Version Control | Console

The tab displays:

- Version control-related commands generated based on the settings you specify through the PyCharm interface.
- The results of executing version control-related commands.



The screenshot shows the PyCharm interface with the 'Console' tab selected. The console displays the output of several git commands. The first command is a status check, followed by another status check, and then a pull command. The pull command output shows the progress of downloading and unpacking objects from a remote repository. The output includes details like 'remote: Counting objects: 515, done.', 'remote: Compressing objects: 100% (253/253), done.', 'Receiving objects: 100% (515/515), 1.49 MiB | 1.12 MiB/s, done.', 'Resolving deltas: 100% (314/314), completed with 56 local objects.', and 'remote: Total 515 (delta 272), reused 147 (delta 147), pack-reused 115 [K]'. It also shows the source repository URL 'https://github.com/JetBrains/help-sources' and the local branch 'master' being updated from the origin/master.

Toolbar

Item Tooltip and Description
Shortcut

	Up the stack trace / Down the stack trace	Click these buttons to navigate up or down in the stack trace and have the cursor jump to the corresponding location.
		
		
	Use Soft Wraps	Click this button to toggle the soft wrap mode of the output.
	Scroll to the end	Click this button to navigate to the bottom of the stack trace and have the cursor jump to the corresponding location.
	Print	Click this button to send the console text to the default printer.
	Clear All	Click this button to remove all text from the console. This function is also available on the context menu of the console.

Context Menu Options

ItemDescription

Compare with Clipboard	Opens the Clipboard vs Editor dialog box that allows you to view the differences between the selection from the editor and the current clipboard content. This dialog is a regular comparing tool that enables you to copy the line at caret to the clipboard, find text, navigate between differences and manage white spaces.
Copy URL	Choose this command to copy the current URL to the system clipboard. This command only shows on a URL, if it is included in an application's output.
Create Gist	Choose this command to open the Create Gist dialog box.
Clear All	Clears the output window.

Local Changes Tab

View | Tool Windows | Version Control - Local Changes

Alt+9

The Local Changes tab lists all files that have been modified locally and have not yet been committed to the repository.

Tip You can assign a custom shortcut for the Show Local Changes action in Settings | Keymap | Version Control Systems to open the Local Changes tab.

Use this tab to [commit](#) and [revert](#) changes, [manage changelists](#), [view differences](#), [view changes in UML Class diagram](#), and [clean up](#) locked folders.

In this topic:

- [Toolbar](#)
- [Changelists pane](#)
 - [Context menu of a selection](#)
- [Preview Diff Pane](#)

Toolbar

Item **Tooltip** **Description**
and
Shortcut

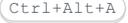
	Refresh Ctrl+F5	Click this button to refresh the status of all files in your workspace, modified both through PyCharm or through any other application.
<p>Note If you are using Perforce as your version control system, only the status of files modified through PyCharm will be updated. This approach improves performance, as it does not require connecting to the server, but it does not let you get an update on the changes made outside PyCharm, for example through the p4v client application. If you want to get an update on all changes to your workspace, use the Force Refresh option.</p>		
	Force Refresh	This button is only available if you are using Perforce as your version control system. Click this button to refresh the status of files in your workspace, both modified through PyCharm or through other applications.
	Commit Changes	Click this button to check in the selected change or changelist. You can also attach and detach Perforce jobs to/from changelists via the Commit Changes dialog.
	Revert Ctrl+Alt+Z	Click this button to roll back the selected changes.
	New Changelist Alt+Insert	Click this button to create a new changelist .
	Delete Changelist Delete	Click this button to delete the selected changelist. Note that you cannot delete the default changelist.
	Set Active Changelist	Click this button to make the selected changelist active . The active changelist is highlighted.
	Move to Another Changelist F6	Click this button to move the selected file to another changelist.
	Shelve Silently Ctrl+Shift+H	Click this button to shelve the selected file or changelist silently, without displaying the Shelve Changes dialog.
	Show Diff Ctrl+D	Click this button to view the differences between your local version of the selected file and its latest version in the repository.
	Show Changes Ctrl+Shift+Alt+D	Use this button to show classes from the selected changelist in a UML Class diagram (for details see Viewing Changes as Diagram).
	Ctrl+NumPad Plus Ctrl+NumPad -	Click these buttons to expand or collapse all nodes.
	Group by Directory Ctrl+P	Click this button to display the changed files grouped by directories. If the button is released, the changed files are grouped by changelists.
	Copy Ctrl+C	Click this button to copy the path to the selected file to the clipboard.
	Show Ignored Files	Click this button to show the Ignored files node with the list of existing files ignored by the VCS .
	Configure Ignored	Click this button to configure the list of files that will be ignored by your version control system.

Files

	Preview Diff	Click this button to have PyCharm open or close the Preview Diff pane to compare the current file with the latest committed revision.
		Click this button to show the corresponding PyCharm help page.

Changelists pane

This pane shows all your changelists, and the files that have been modified in each changelist.

If new files have been added to your project that have not yet been checked-in to a version control system, the Unversioned Files node appears under which all such files are listed. If you have a large number of unversioned files (over 50), they are not displayed in the changelists pane. Instead, the Click to browse link appears. Click this link to open the Unversioned Files dialog to review the list of unversioned files. You can quickly delete unversioned files from the Changelists pane or the Unversioned Files dialog by pressing , or add them to the VCS by pressing .

Context menu of a selection

Item	Shortcut	Description
 Commit Changes	N/A	Select this option to check in the selected file or changelist. You can also attach and detach Perforce jobs to changelists via the Commit Changes dialog.
 Revert		Select this option to roll back the selected changes.
 Move to Another Changelist		Select this option to move the selected item to another changelist.
 Show Diff		Select this option to view the differences between your local copy and the latest version in the repository.
 Jump to Source		Select this option to open the selected file(s) in the editor.
 New Changelist		This option is only available if a changelist is selected. Select this option to create a new changelist .
 Delete Changelist	N/A	This option is only available if a changelist is selected. Select this option to delete the selected changelist .
 Delete	N/A	This option is only available if single files are selected, not a changelist.
Check Out	N/A	This option is only available if a file under the Modified without Checkout node is selected. Use this option to check out the selected file from the repository.
Add to VCS	N/A	This option is only available if a file under the Unversioned Files node is selected. Use this option to add the selected files to your version control system.
Ignore	N/A	This option is only available if a file under the Unversioned Files node is selected. Use this option to ignore the selected file if you want to leave it unversioned.
Create Patch	N/A	Select this option to create a patch .
Shelve Changes	N/A	Select this option to shelve the selected changes .
Shelve in Perforce	N/A	This option is only available if you are using Perforce as a version control system. Select this option to shelve your changes in Perforce. You will be asked to select which files you want to shelve and provide a description. After you've shelved your changes, the corresponding changelist will appear. You can unshelve it any time from the changelist's context menu.
 Refresh		Select this option to refresh the status of files in your workspace.
Local History	N/A	Select this option and choose one of the following from the popup menu: – Show History: select this option to View local history of the selected file or folder. – Put Label: select this option to add a label to the current version of the selected file or folder.
<Specific version control system>	N/A	Select this option to invoke a popup menu with options specific for the version control system you are using.

Preview Diff Pane

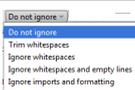
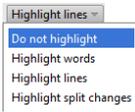
This pane opens when you click the Preview Diff button  on the toolbar. In this pane you can examine the changes made to the selected file compared to its base revision.

The pane consists of two areas:

- The affected code as it was in the base revision.
- The affected code as it is after a change has been made.

Item Tooltip and Shortcut

	Previous Difference / Next Difference  	Use these buttons to jump to the next/previous difference. When the last/first difference is hit, PyCharm suggests to click the arrow buttons  /  once more and compare other files, depending on the Go to the next file after reaching last change option in the Differences Viewer settings . This behavior is supported only when the Differences Viewer is invoked from the Version Control tool window.
--	---	--

 	<p>Compare Previous/Next File</p> <p>Alt+Left Alt+Right</p>	<p>Click these buttons to compare the local copy of the previous/next file with its update from the server.</p> <p>Note These controls are only available if more than one file has been modified locally.</p>
	<p>Jump to Source</p> <p>F4</p>	<p>Click this button to open the selected file in the active pane in the editor. The caret will be placed in the same position as in the Differences Viewer.</p>
<p>Viewer type</p>		<p>Use this drop-down list to choose the desired viewer type. The side-by-side viewer has two panels; the unified viewer has one panel only.</p> <p>Both types of viewers enable you to</p> <ul style="list-style-type: none"> - Edit code. Note that one can change text only in the right-hand part of the default viewer, or, in case of the unified viewer, in the lower ("after") line, i.e. in your local version of the file. - Perform the Apply/Append/Revert actions.
<p>Whitespace</p>		<p>Use this drop-down list to define how the differences viewer should treat white spaces in the text.</p> <ul style="list-style-type: none"> - Do not ignore: white spaces are important, and all differences are highlighted. This option is selected by default. - Trim whitespaces: (" \t", " ") , if they appear in the end and in the beginning of a line. <ul style="list-style-type: none"> - If two lines differ in trailing whitespaces only, these lines are considered equal. - If two lines are different, such trailing whitespaces are not highlighted in the By word mode. - Ignore whitespaces: white spaces are not important, regardless of their location in the source code. - Ignore whitespaces and empty lines: the following entities are ignored: <ul style="list-style-type: none"> - all whitespaces (as in the 'Ignore whitespaces' option) - all added or removed lines consisting of whitespaces only - all changes consisting of splitting or joining lines without changes to non-whitespace parts. <p>For example, changing <code>a b c</code> to <code>a \n b c</code> is not highlighted in this mode.</p>
<p>Highlighting mode</p>		<p>Select the way differences granularity is highlighted.</p> <p>The available options are:</p> <ul style="list-style-type: none"> - Highlight words: the modified words are highlighted - Highlight lines: the modified lines are highlighted - Highlight split changes: if this option is selected, big changes are split into smaller 'atomic' changes. <p>For example, <code>A \n B</code> vs. <code>A X \n B X</code> will be treated as two changes instead of one.</p> <ul style="list-style-type: none"> - Do not highlight: if this option is selected, the differences are not highlighted at all. This option is intended for significantly modified files, where highlighting only introduces additional difficulties.
	<p>Collapse unchanged fragments</p>	<p>Click this button to collapse all unchanged fragments in both files. The amount of non-collapsible unchanged lines is configurable in the Diff & Merge settings page.</p>
	<p>Synchronize scrolling</p>	<p>Click this button to simultaneously scroll both differences panes; if this button is released, each of the panes can be scrolled independently.</p>
	<p>Editor settings</p>	<p>Click this button to invoke the list of available settings. Select or clear this options to show or hide whitespaces, line numbers and indent guides, to use or disable the use of soft wraps, and to set the highlighting level. These commands are also available from the context menu of the differences viewer gutter.</p>
	<p>Show diff in external tool</p>	<p>Click this button to invoke an external differences viewer, specified in the External Diff Tools settings page. This button only appears on the toolbar when the Use external diff tool option is enabled in the External Diff Tools settings page.</p>
	<p>Help</p> <p>F1</p>	<p>Click this button to show the corresponding help page.</p>
<p>N/A</p>	<p>Annotate</p>	<p>This option is only available from the context menu of the gutter.</p> <p>Use this option to explore who introduced which changes to the repository version of the file in question, and when. The annotations view lets you see detailed information for each line of code, such as the version from which this line originated, the ID of the user who committed this line, and the commit date.</p> <p>You can configure the amount of information displayed in the annotations pane.</p> <p>For more details on annotations, refer to Viewing Changes Information</p>

The most useful shortcuts in the Diff Pane are the following:

Shortcut Description

<p>Ctrl+Shift+D</p>	<p>Use this keyboard shortcut to show the popup menu of the most commonly user diff commands.</p>
<p>Ctrl+Tab</p>	<p>Use this keyboard shortcut to switch between the left and the right panes.</p>
<p>Ctrl+Shift+Tab</p>	<p>Use this keyboard shortcut to select the position obtained by Ctrl+Tab in the opposite pane.</p>
<p>Ctrl+Z / Ctrl+Shift+Z</p>	<p>Use this keyboard shortcut to undo/redo a merge operation. Conflicts will be kept in sync with the text.</p>

Log Tab

View | Tool Windows | Version Control - Log

Alt+9

This tab is only available if you are using [Git](#) or [Mercurial](#) for version control.

This tab shows all changes committed to all branches of the local and remote repositories, or to a [specific](#) branch or repository.

The log is updated automatically in the background mode on every change (a commit, fetch, rebase, etc.), even if the Log tab is closed.

Tip You can assign a custom shortcut for the Show VCS Log action in Settings | Keymap | Version Control Systems to open the Log tab.

The tab contains the following panes:

- The [Commits](#) pane is located on the left of the tool window and shows the commits to all or [selected](#) branches from the local and remote repositories.
- The [Changed Files](#) pane is located on the right of the tool window and shows the list of files that were modified within the selected commit.
- The [Commit Details](#) pane is located on the right under the [Changed Files](#) pane and shows the details of the selected commit.

Commits Pane

The commits pane consists of the following areas:

- [Commits](#)
- [Toolbar](#) (some of the toolbar commands are duplicated in the [context menu](#)).

Commits

This area shows a list of all commits performed to the selected branch, or to all branches. For each commit, the list shows the commit message, the author, and the commit date. The latest commit in each branch is supplied with a label with the name of the branch in which it was performed.

There are the following labels:

- local (Mercurial) and regular (Git/Mercurial) for tags.
- tip (Mercurial) for the latest revision in the repository.
- HEAD (Git/Mercurial) for the current working revision.



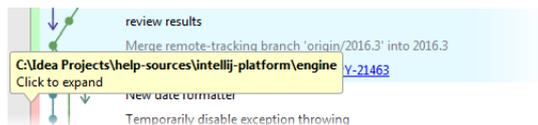
For Git, the color of the label depends on the branch type (local or remote).

For Mercurial, there are different color labels for bookmarks, open heavy branches and closed heavy branches.

Note that clicking an arrow takes you to the next commit in a long branch:



In multi-repository projects, the colored stripe on the left indicates which root the selected commit belongs to (each root is marked with its own color). Hover the mouse cursor over the colored stripe to invoke a tip that shows the root path:



You can also enable the [Show Root Names](#) option if you want to expand the Roots column with full root names.

Committed changelists often correspond to issues in tracking systems. You can jump to such issues in a browser right from the Commits pane. This functionality is available if:

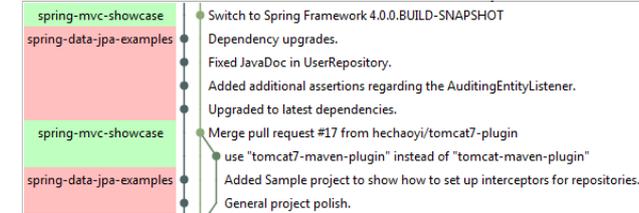
- The [pattern](#) of the bug tracking system is specified in the [Issue Navigation](#) Settings dialog box.
- The corresponding issue number is mentioned in the commit message.

After issue navigation has been configured, issue numbers in commit messages are rendered as links. Clicking such link brings you to the corresponding page of your issue tracker.

Toolbar

ItemTooltip Description
and

Shortcut

Filter	N/A	<p>Use this text box to search through the list of commits. You can enter full commit names or messages or their fragments, revision numbers, or regular expressions. To finalize the search, press <code>Enter</code> or move the focus away from the search field.</p> <p>Tip You can quickly switch the focus to the search field by pressing <code>Ctrl+L</code>.</p>
Q-	N/A	Click this button to show previous search patterns.
	N/A	Click this button to clear the search and return to the full list of commits.
	N/A	<p>Click this button to invoke the following options:</p> <ul style="list-style-type: none"> – Regex: if this option is selected, anything you type in the search field is treated as a regular expression, for example, <code>#\d+</code>. – Match Case: if this option is selected, only entries with the matching case count.
Branch	N/A	Use this drop-down list to filter commits by the branch. If you want to see commits from all local and remote branches, select All.
User	N/A	Use this drop-down list to filter commits by the author. To view all commits by a specific author, click Select and start typing the author's name. To view commits by all users, select All.
Date	N/A	Use this drop-down list to filter commits by a time-frame or a specific date. To view commits made on a specific date, click Select and specify the date. To view commits made on all dates, select All.
Paths	N/A	Use this drop-down list to filter commits by the folder (for projects that have one root), or by the root and folder (for multi-root projects). To view commits to a specific folder, click Select Folders and specify the folder name. For multi-repository projects, you can also select the check-box next to one or several roots in the Roots section.
	IntelliSort	If this option is enabled, you get a more convenient way to view merges by displaying the incoming commits first, directly below the merge commit.
	Show long edges	If this option is enabled, long branches are displayed in full, even if there are no commits in them. If this option is disabled (by default), long branches are replaced with a down arrow.
	Refresh	Click this button to refresh the list of commits.
	Go to Hash/Branch/Tag	<p>Click this button and specify a hash, tag or branch you want to jump to.</p>  <p>You can select a reference with the same name from different repositories. The name of each repository is displayed on the right along with its color indicator.</p>
	Cherry-pick (for Git) Graft (for Mercurial)	<p>Click this button to apply changes from the selected commit to the current branch.</p> <p>Note This button is disabled if the selected commit is already contained in the current branch.</p>
	Highlight non-picked commits	<p>This command is only available if you are using Git as your version control system.</p> <p>Click this button to highlight the commits from the selected branch that have not yet been applied to the current branch.</p>
	Quick settings	<p>Click this button to invoke one of the following commands:</p> <ul style="list-style-type: none"> – Show Root Names: enable this option if you want to expand the Roots column on the left showing full root names in a multi-repository project.  <ul style="list-style-type: none"> – Compact References View: if this option is enabled, branch references for a single commit are displayed in a collapsed view: <code>origin/171.3780</code> if you want to expand each branch reference on a line, deselect this option: <code>origin/171.3780</code> <code>idea/171.3780.29</code> – Show Tag Names: enable this option if you want tag names to be displayed in addition to the tag icon: <code>idea/171.3780.29</code> if this option is disabled, you can still view a tag name by hovering the mouse over the tag icon. – Collapse Linear Branches: enable this option to collapse all branches on the graph so that a dotted line is shown instead of successive commits. It is also possible to collapse an individual expanded branch by clicking it. – Expand Linear Branches: enable this option to expand all collapsed branches to show successive commits on the graph.

It is also possible to expand an individual collapsed branch by clicking it.

- Highlight: select which types of commits you want to highlight:
 - My Commits: if enabled, your commits are highlighted in bold font.
 - Merge Commits: if enabled, merge commits are grayed out.
 - Current Branch: if enabled, commits to the current branch are highlighted with a blue background.

Context menu commands

Item	Description	Available in
Copy Revision Number 	Use this command to copy the revision number of the selected commit to the clipboard.	Git Mercurial
Create Patch 	Use this command to create a patch based on the selected commit.	Git Mercurial
Cherry-pick (Git)	Use this command to apply the changes from the selected commit to the current branch .	Git
Graft (Mercurial) 		Mercurial
Checkout Revision	Use this command to check out the state of files recorded in the selected commit.	Git
Update to Revision	Use this command to change your working copy parent revision to the selected commit. New commits will carry on from the revision (commit) you update to.	Mercurial
New Branch	Use this command to create a new branch based on the selected commit.	Git Mercurial
New Tag	Use this command to add a new tag to the selected commit.	Git Mercurial
Branch <branch_name> / Branches	This command appears for all branches that point to the selected commit (Branch <branch_name> if there is one branch, or Branches if there are multiple branches) and provides the same options as the ones available from the Branches popup and submenu : <ul style="list-style-type: none"> - Checkout as new local branch - Compare - Rebase onto - Merge - Delete <p>If the Control repositories synchronously option is enabled, and the selected branch exists in multiple repositories, an additional menu option named In All Repositories appears that allows you to perform the same operations in all repositories simultaneously.</p>	Git Mercurial
Reset Current Branch to Here	Use the command to reset the current branch head to the selected commit. In the Git Reset dialog that opens, select the mode in which the working tree will be updated.	Git
Undo Commit	This command is only available for the commits made by you, and allows you to revert the changes and undo the selected commit.	Git
Open in GitHub 	Use this command to open the page that corresponds to the selected commit on GitHub .	Git
MQ	Use this submenu to manage Mercurial Queues : <ul style="list-style-type: none"> - Import: use this command to turn the selected changeset into a patch. - Goto patch: use this command to open the MQ: <project_name> tab that shows a queue of patches that have not been applied yet. - Rename Patch: use this command to rename the selected patch. - Finish Patches: use this command to turn the selected patch into a permanent changeset. 	Mercurial

Changed files pane

This pane shows a list of files that were modified within the currently selected commit.

Toolbar

Many of the options available in the toolbar are duplicated in the context menu.

Item/Tooltip and Shortcut	Description
---------------------------	-------------

	<p>Show Diff</p> <p>Ctrl+D</p>	Click this button to open the Differences Viewer for Files where you can compare the current and the previous revision of the selected file.
	Show Diff with Local	Click this button to show the differences between the selected revision of the selected file and its current local copy.
	<p>Edit Source</p> <p>F4</p>	Click this button to open the local copy of the selected file for editing.
	<p>Open Repository Version</p>	Click this button to open the repository version of the selected file for editing.
	<p>Revert Selected Changes</p>	Click this button to roll back the changes in the selected file.
	<p>Show History for Revision</p>	Click this button to open the History tab for the selected file that lets you explore the history of all file revisions.
	Show details	Click this button to show the Commit details pane .
	<p>Group by Directory</p> <p>Ctrl+P</p>	Click this button to transform a flat list of files into a tree of packages with files.
	<p>Expand All/Collapse All</p> <p>Ctrl+NumPad Plus</p>	<p>Click this button to expand/collapse all nodes.</p> <p>Note that these buttons are only available only when tree-view is enabled.</p>

Commit Details

This area is displayed when the [Show Details](#)  option is enabled.

This area shows the details on the commit selected in the [Commits](#) list, such as the commit message, hash, author, the link to the author's email, date, time, root and branches.

If the selected commit is contained in more than six branches, only the first six are displayed and the Show All link appears that you can click to expand a complete list of branches.

History Tab

The History tab is added to the [Version Control tool window](#) on invoking the Show History command for a file or directory through the menu of a particular VCS. A new tab is created for each file or directory with the following name: History: <file_name>. The set of toolbar buttons differs slightly depending on your version control system.

All commands available from the toolbar are also available from the context menu of a selection.

Item Tooltip and Shortcut

Item	Tooltip	Description
	Compare	Click this button to compare the selected revision of a file with its previous revision in the Differences Viewer for Files .
	Ctrl+D	
	Show Diff with Local	Click this button to compare the selected revision of a file with its local copy in the Differences Viewer for Files .
	Create Patch	Click this button to create a patch from the selected revision.
	Get	Click this button to retrieve the selected revision. If the local copy has already been modified, PyCharm prompts to overwrite the local version, or cancel the operation.
	Annotate	Click this button to open the selected revision of a file in the editor with annotations.
	Show All Affected Files	Click this button to open the Paths Affected in Revision dialog where you can view all files that were modified in the selected revision.
	Shift+Alt+A	
	Copy Revision Number	Click this button to copy the revision number of the commit that the selected file belongs to to the clipboard.
	Compare all classes from revision on UML	Click this button to view all classes of the selected revision as a UML Class diagram. See section Viewing Changes as Diagram .
	Ctrl+Shift+D	
	Open in GitHub	Click this button to open the page that corresponds to the selected commit on GitHub .
	Show All Branches	Click this button to display changes from branches other than the current one.
	Show Branches	Note This option is only available if you are using Perforce for version control. Click this button to show branches.
	Show All Revisions Submitted In Selected Changelist	Note This option is only available if you are using Perforce for version control. Click this button to display the list of all revisions committed in the same changelist as the selected revision of a file.
	Refresh	Click this button to refresh the current information.
	Show Details	Click this button to show the commit message for the selected revision.
	Close	Click this button to close the current history tab.
	Ctrl+Shift+F4	

Integrate to Branch Info View

View | Tool Windows | Version Control

This view is available after running integration with the Run status after update setting specified. The view displays a list of files or packages affected by the latest integration. The items are displayed under the following nodes:

- Modified
- Merged
- Not in repository
- Locally added

ItemTooltip and shortcut	Description
	Click this button, to group information within nodes by packages. If the button is released, files are presented in plain lists.
	 Click this button to expand all nodes.
	 Click this button to collapse all nodes.
	Click this button to close the view.
	 Click this button to show the corresponding reference page.

Repository and Incoming Tabs

VCS | Show Changes View - Repository/Incoming

View | Tool Windows | Version Control - Repository/Incoming

Alt+9

The Repository and Incoming tabs are only available for non-distributed version control systems (i.e. all VCSs supported by PyCharm except for Git and Mercurial).

The Repository tab shows the changes committed to the repository under the VCS roots within the current project. The Incoming tab shows the changes committed to the repository by other team members, and not yet checked out locally. Both tabs display the information stored in the history cache. The number of changelists displayed depends on the [cache scope](#).

Each tab contains the following panes:

- The [Changelists](#) pane shows changelists.
- The [Changed Files](#) pane shows the list of files that were modified and committed within the selected changelist.

Note that if you are using SVN 1.5 or higher both on the server and in the local working copies, the Repository tab also features a [Merge Info](#) pane that configures the view in the other two panes and provides control over integration between branches.

Changelists Pane

The pane shows the changelists committed and stored in the history cache. When you click a changelist, the files affected by the selected commit are displayed in the [Changed Files](#) pane.

Committed changelists often correspond to issues in tracking systems. You can have such issues opened in the browser right from the Changelists pane. This functionality has the following prerequisites:

- The [pattern](#) of the bug tracking system is specified in the [Issue Navigation](#) Settings dialog box.
- The corresponding issue number is mentioned in the commit message.

After issue navigation has been configured, issue numbers in commit messages are rendered as links. Clicking such link brings you to the corresponding page of your issue tracker.

Item Tooltip and Shortcut	Description	Available In	
	Refresh	Click this button to refresh the information in the view.	Both tabs
			
	Show Details	Click this button to show the following information on the selected changelist: <ul style="list-style-type: none">– Changelist number– User and client name– Date and time of commit	Both tabs
			
	Create Patch	Click this button to create a patch based on the selected changelist.	Repository tab
	Revert Changes	Click this button to create a <i>reverse patch</i> for the selected changelist and roll back the changes made previously. You can use this action to revert changes committed by any user. The Select Target Changelist dialog box opens. Note that if the reverse patch applies to a version committed earlier, this rollback attempt may fail because of the conflicts with the later changes.	Repository tab
	Clear	Click this button to clear the history cache. The commits list will be emptied. To restore it, click Refresh .	Repository
	Edit Revision Comment	Click this button to edit the message for the selected commit.	Repository
	Update Project	Click this button to update the project to the latest available version.	Incoming tab
			
	Expand All	Click this button to expand all nodes.	Both tabs
			
	Collapse All	Click this button to collapse all nodes.	Both tabs
			
	Copy	Click this button to copy the commit message of the selected changelist to the Clipboard.	Both tabs
			

?

	 Help	Click this button to show the corresponding help topic.	Both tabs
	 Highlight Integrated	Click this button to have the Merge Info pane displayed. The button is enabled only when both the server side and the client side use Subversion 1.5.	Repository tab
Filter by		Use this drop-down list to hide the changelists that are of no interest to you, and only view only the changelists that satisfy a certain criterion. The following options are available: <ul style="list-style-type: none"> – User: select this option to filter the commits by the user. – Structure: select this option to filter the commits by the target module or folder. – Client: select this option to filter the commits by the computer from which they were made. – None: select this option to turn off filtering and return to the default view. 	Both tabs
Group by		Use this drop-down list to group changelists following a certain criterion. The following options are available: <ul style="list-style-type: none"> – Date: select this option to group the commits by date. – User: select this option to group the commits by users. – Client: select this option to group the commits by the computer from which they were made. 	Both tabs
Search		Use this text box to enter a search pattern and locate the commits whose commit messages matches the specified string. As you type, the list dynamically reduces to show the changelists with the commit messages that match the specified pattern. To save the search pattern, press Enter. <p>To view the list of recent search patterns, click the  button.</p> <p>To clear the list of search patterns, click the  button.</p>	Repository tab

Changed files pane

ItemTooltip and Shortcut	Description	
	Show Diff	Click this button to show the differences between the current and the previous revision of the selected file. 
	Show Diff with Local	Click this button to show the differences between the selected revision of the selected file and its current local copy.
	Edit Source	Click this button to open the local copy of the selected file for editing. 
	Open Repository Version	Click this button to open the repository version of the selected file.
	Revert Selected Changes	Click this button to revert the changes to the selected file and roll back to its previous revision.
	Integrate to Branch	Click this button to integrate the changes from the selected file to the target branch.
	Compare Subversion Properties	This option is only available if you are using Subversion as your version control system. Click this button to view the differences in file properties between the current version and the previous revision.
	Show History	Click this button to open the History view of the selected file in the Version Control tool window.
	Group by Directory	Click this button to transform a flat list of files into a tree of packages with files. 
	Expand All	Click this button to expand all nodes. The button is available only when the files in the pane are displayed grouped by directories. 
	Collapse All	Click this button to collapse all nodes. The button is available only when the files in the pane are displayed grouped by directories. 

Merge Info pane

The pane is available only if you are using SVN 1.5 or higher both on the server and in the local working copies.

In this pane, specify a pair of branches whose integration with each other you want to monitor. The Changelists pane will show the changelists related to the specified branches and provide the information on the [integration status](#) of each changelist.

You can specify several pairs of branches if several projects or roots are involved.

Item Tooltip and Shortcut	Description
---------------------------	-------------

From	Specify the URL address of the source branch. PyCharm suggests the URL address selected in the Checkout from Subversion dialog box.
To	Do the following: <ul style="list-style-type: none"> Specify the path to the target branch. Click  or press <code>Shift+Enter</code> to open the Select Branch dialog box. Specify the path to the local working copy to which you will apply patches created based on the selected changelists. Click  to open the Configure Working Copy Paths dialog box and select a working copy.
	Highlight Integrated Click this button to have each changelist in the Changelists pane supplied with an indication of whether it is integrated or not.
	Integrate To Branch Click this button to integrate the selected changelist into the working copy. The Integrate To Branch dialog box opens.
	Undo Integrate To Branch Click this button to revert the last integration of the selected changelist into the working copy.
	Mark As Merged Click this button to indicate that the selected changelist is integrated into the working copy without actually integrating the changelist. The action affects the administrative information in the <code>.svn</code> folder. The icon next to the selected changelist changes from  to  .
	Mark As Not Merged Click this button to indicate that the selected changelist is not integrated into the working copy without actually reverting integration. Update the administrative information in the <code>.svn</code> folder. The icon next to the selected changelist changes from  to  .
	Filter Out Integrated Click this button to display only changelists that have not been integrated into the working copy.
	Filter Out Not Integrated Click this button to display only changelists that have been integrated into the working copy.
	Filter Out Others Click this button to hide extraneous changelists in the Changelists pane. Extraneous changelists are changelists that are managed in another VCS or are located under another root.
	Show Working Copies Click this button to open the Subversion Working Copies Information dialog box.
	Refresh Click this button to refresh the information in the Changelists pane.

Update Info Tab

View | Tool Windows | Version Control

This tab is available when [local information is synchronized](#) to the server.

Item Tooltip and shortcut

	Group by Packages	When this button is pressed, the update information within nodes is grouped by packages.
	Group by Changelists	When this button is pressed, the update information within nodes is grouped by changelists, and by the day the changelist has been committed. The Update Info tab is divided into two panes: the left pane shows the changelists, grouped by the check-in date, and the right pane shows the list of changed files . Grouping by changelists is not available if the project is under Git or Mercurial control.
	Expand all	Use these buttons to have all nodes expanded or collapsed.
	Collapse all	
	Show Diff	Click this button to open the Differences Viewer for Files , where you can compare the local copies of all the project files one after another with their updates from the server. Use the buttons Compare Next File  and Compare Previous File  to scroll through the list of updated files.
	Close	Click this button to close the tab.
	Help	Click this button to show reference page.

The controls of the Changed files pane appear when the button  is pressed.

	Show Diff	Click this button to show differences between the selected and the previous revision for the selected file in the Changes Files pane.
	Show Diff with Local	Click this button to show differences between the selected revision of a file, and its current local copy.
	Edit Source	Click this button to open local copy of the selected file for editing.
	Open Repository Version	Click this button to open repository version of the selected file.
	Revert Selected Changes	Click this button to revert selected changes.
	Show History	Click this button to open the History view of the selected file in the History Tab of the Version Control tool window.
	Group by Directory	When the button is not pressed, the pane shows a flat list of files. When the button is pressed, the pane shows files in their respective packages. In the latter case, expand and collapse buttons appear in the toolbar.
	Expand all	Use these buttons to have all nodes expanded or collapsed.
	Collapse all	
	Select All	Click this button to select all files in the Changed Files pane.

Shelf Tab

View | Tool Windows | Version Control - Shelf

The tab is added to the [Version Control tool window](#) when you [shelve](#) a change or a changelist, and is displayed until you permanently remove all shelved changes, including the already [unshelved](#) ones, and imported external patches.

By default, this tab shows all shelved changes that have not been unshelved yet. Changes are grouped into shelves. A shelf is a changelist created when you [shelve](#) changes. A shelf is identified by the commit message. You can have PyCharm show the unshelved changes. They can be restored and re-applied as many times as necessary, until they are removed permanently.

For details, see [Shelving and Unshelving Changes](#).

Note You can assign a custom shortcut for the Show Shelf action in [Settings | Keymap | Version Control Systems](#) to open the Shelf tab.

Toolbar

Item
Tooltip
and
shortcut

	Show Diff Ctrl+D	Choose this option to open the Differences Viewer for Files and compare the shelved version of a file with its current local version.
	Create Patch	Choose this option to create a patch file based on a shelved change. In the Create Patch dialog box that opens, specify the file to save the patch in, and the changes to create a patch from. By default, all changes from the selected shelf are involved in the patch creation. To view which changes are included, click the Selected link. For details, see Creating Patches .
	Show/Hide already unshelved	Click this button to have PyCharm show or hide all available shelved changes, both already applied and not. Note that this button is duplicated on the context menu.
		Click this button to reveal the menu of actions described below .

Context menu

The context menu and the toolbar button  provide the same set of actions. The context menu is available by right-clicking a change, a change list, or anywhere in the tab.

Item
Shortcut
Description

Unshelve Changes		Choose this option to apply all changes from the selected shelf. In the Unshelve Changes to Changelist dialog box that opens, specify the changelist you want to add the changes to. For details, see Shelving and Unshelving Changes .
Unshelve	Ctrl+Shift+U	Choose this option to select specific changes you want to unshelve instead of applying an entire shelf. In the Unshelve Changes to Changelist dialog box that opens, specify the changes you want to apply, and the changelist you want to add these changes to. For details, see Shelving and Unshelving Changes .
Restore		Choose this option to re-activate the unshelved changes. By default, unshelved changes are no longer shown in the list of available shelved changes, therefore you first need to have PyCharm display it by choosing Show Already Unshelved. You can restore any change as many times as you need until the change is permanently removed by choosing Delete. For details, see Shelving and Unshelving Changes .
Import Patches		Choose this option to apply patches created externally or through PyCharm. In the dialog box that opens, choose the files to import patches from. The imported patches are treated as shelved changes and are shown in the Shelf tab of the Version Control tool window where you can unshelve them at any time.
Rename	Shift+F6	Choose this option to modify the name of the selected shelved changelist.
Delete	Delete	Choose this option to permanently delete the selected shelved change. It can remove any change no matter whether it has been already unshelved or not. For details, see Shelving and Unshelving Changes .
Show Already Unshelved		Choose this option to have PyCharm show all available shelved changes, both already applied and not. By default, unshelved changes are hidden. For details, see Shelving and Unshelving Changes .
Delete All Already Unshelved		Choose this option to have PyCharm permanently remove all shelved changes that have been already applied. For details, see Shelving and Unshelving Changes .

MQ: <project_name> Tab

View | Tool Windows | Version Control - Log - Context menu of a commit - Mercurial - Goto patch

This tab is only available if your project is under [Mercurial](#) version control.

It displays all patches and allows you to manage the [Mercurial Queue](#).

You can drag-and-drop patches in the queue to change the order in which they will be applied.

Toolbar and Context Menu

Item	Shortcut	Description	Available from
 Reload from file	Ctrl+F5	Use this command to refresh the list of patches from the <code>series</code> file. Note that when you drag-and-drop patches in the queue, the corresponding changes in the <code>series</code> file will only be saved when you switch to a different tab, or perform some other external action. This allows you to revert such changes simply by reloading from file.	Toolbar
 Goto	Shift+Alt+G	Use this command to apply the selected patch and all patches above it in the queue. The applied patches will become visible in the Log tab and the changes will be registered in the working copy of the repository.	Toolbar Context menu
 Move and Push	Shift+Alt+P	Use this command to apply the selected patch without applying whatever other patches may be before it in the patch stack.	Toolbar Context menu
 Fold	Shift+Alt+D	Use this command to merge the selected non-applied patch with the topmost applied patch, and remove it from the list.	Toolbar Context menu
 Delete	Delete	Use this command to delete the selected patch from the series file	Toolbar
Apply Patch	N/A	Use this command to apply the selected patch. The Apply Patch dialog will be displayed where you can select which changes you want to restore.	Context menu

V8 Heap Tool Window

The tool window opens when you take a snapshot and choose to open it. The tool window shows the collected profiling data. If the window is already opened and shows the profiling data for another session, a new tab is added. Tabs that were opened automatically are named after the run configurations that control execution of the applications and collecting the profiling data.

If you want to open and analyze some previously saved memory profiling data, choose V8 Profiling - Analyze V8 Heap Snapshot on the main menu and select the relevant `.snapshot` file. PyCharm creates a separate tab with the name of the selected file.

The tool window has three tabs that present the collected information from different points of view.

On this page:

- [Containment](#)
- [Biggest Objects](#)
- [Summary](#)
- [Details Pane](#)
- [Toolbar](#)
- [Context Menu of an Object](#)

Containment

The tab shows the objects in your application grouped under several top-level entries: **DOMWindow objects**, **Native browser objects**, and **GC Roots**, which are roots the **Garbage Collector** actually uses. See [Containment View](#) for details.

For each object, the tab shows its **distance from the GC root**, that is the shortest simple path of nodes between the object and the GC root, the [shallow size](#) of the object, and the [retained size](#) of the object. Besides the absolute values of the object's size, PyCharm shows the percentage of memory the object occupies.

Biggest Objects

The tab shows the most memory-consuming objects sorted by their [retained sizes](#). In this tab, you can spot memory leaks provoked by accumulating data in some global object.

Summary

The tab shows the objects in your application grouped by their types. The tab shows the number of objects of each type, their size, and the percentage of memory that they occupy. This information may be a clue to the memory state.

Details Pane

Each tab has a Details pane, which shows the path to the currently selected object from GC roots and the list of object's **retainers**, that is, the objects that keep links to the selected object. Every heap snapshot has many "back" references and loops, so there are always many retainers for each object.

Navigating through a Snapshot

- To help differentiate objects and move from one to another without losing the context, mark objects with text labels. To set a label to an object, select the object of interest and click  on the toolbar or choose Mark on the context menu of the selection. Then type the label to mark the object with in the dialog box that opens.
- To navigate to the function or variable that corresponds to an object, select the object of interest and click  on the toolbar or choose Edit Source on the context menu of the selection. If the button and the menu option are disabled, this means that PyCharm has not found a function or a variable that corresponds to the selected object.
If several functions or variables are found, they are shown in a pop-up suggestion list.
- To jump from an object in the Biggest Objects or Summary tab or Occurrences view to the same object in the Containment tab, select the object in question in the Biggest Objects or Summary tab and click  on the toolbar or choose Navigate in Main Tree on the context menu of the selection. This helps you investigate the object from the containment point of view and concentrate on the links between objects.
- To search through a snapshot:
 1. In the Containment tab, click  on the toolbar.
 2. In the [V8 Heap Search Dialog](#) that opens, specify the search pattern and the scope to search in. The available scopes are:
 - Everywhere: select this check box to search in all the scopes. When this check box is selected, all the other search types are disabled.
 - Link Names: select this check box to search among the object names that **V8** creates when calling the **C++ runtime**, see <http://stackoverflow.com/questions/11202824/what-is-in-javascript>.
In the [V8 Heap Tool Window](#), link names are marked with the `%` character (`%<link name>`).
 - Class Names: select this check box to search among functions-constructors.
 - Text Strings: select this check box to perform a textual search in the contents of the objects.
 - Snapshot Object IDs: select this check box to search among the unique identifiers of objects. **V8** assigns such a unique identifier in the format to each object when the object is created and preserves it until the object is destroyed. This means that you can find and compare the same objects in several snapshots taken within the same session.
In the [V8 Heap Tool Window](#), object IDs are marked with the `@` character (`@<object id>`).
 - Marks: select this check box to search among the labels you set to objects manually by clicking  on the toolbar of the Containment tab.

The search results are displayed in the Details pane, in a separate Occurrences of '`<search pattern>`' view. To have the search results shown grouped by the search scopes you specified, press the Group by Type toggle button on the toolbar.

When you open the dialog box next time, it will show the settings from the previous search.

Toolbar

The toolbar is common for all tabs and most of the toolbar buttons are available in all tabs.

ItemDescriptionAvailable in

	Click this button to set a label to the selected object. This helps you differentiate objects and move from one to another without losing the context.	Containment tab, Biggest Objects tab, Summary tab, Details pane, Occurrences view
	Click this button to find a function in a snapshot: in the V8 Heap Search Dialog that opens, specify the search pattern and the scope to search in. The available scopes are: <ul style="list-style-type: none">– Everywhere: select this check box to search in all the scopes. When this check box is selected, all the other search types are disabled.– Link Names: select this check box to search among the object names that V8 creates when calling the C++ runtime, see http://stackoverflow.com/questions/11202824/what-is-in-javascript. In the V8 Heap Tool Window, link names are marked with the % character (%<link name>).– Class Names: select this check box to search among functions-constructors.– Text Strings: select this check box to perform a textual search in the contents of the objects.– Snapshot Object IDs: select this check box to search among the unique identifiers of objects. V8 assigns such a unique identifier in the format to each object when the object is created and preserves it until the object is destroyed. This means that you can find and compare the same objects in several snapshots taken within the same session. In the V8 Heap Tool Window, object IDs are marked with the @ character (@<object id>).– Marks: select this check box to search among the labels you set to objects manually by clicking  on the toolbar of the Containment tab.	Containment tab
	Click this button to jump from an object in the Biggest Objects or Summary tab or Occurrences view to the same object in the Containment tab. This helps you investigate the object from the containment point of view and concentrate on the links between objects.	Biggest Objects tab, Summary tab, Occurrences view
	Click this button to navigate to the function or variable that corresponds to the selected object.	Containment tab, Occurrences view
	Press this toggle button to have the search results shown grouped by the search scopes you specified.	Occurrences view
	Click this button to open the reference page for the tool window.	All
	Click this button to close the tool window.	All

Context Menu of an Object

ItemDescription

Mark	Choose this option to set a label to the selected object. This helps you differentiate objects and move from one to another without losing the context.
Navigate in Main Tree	Choose this option to jump from an object in the Biggest Objects or Summary tab or Occurrences view to the same object in the Containment tab. This helps you investigate the object from the containment point of view and concentrate on the links between objects.
Jump to Source	Choose this option to navigate to the function or variable that corresponds to the selected object.

V8 Heap Search Dialog

The dialog box opens when you click  on the toolbar in the [V8 Heap Tool Window](#). Use the dialog box to find constructors, object IDs, character strings, etc. in a snapshot. The search results are displayed in the Details pane, in a separate Occurrences of '<search pattern>' view.

ItemDescription

Search	In this text box, type the search pattern to look for. Select the Case Sensitive check box if necessary.
Scope	<p>In this area, specify the type of objects to limit the search to. When the Everywhere check box is selected, all the other search types are not available.</p> <ul style="list-style-type: none">– Everywhere: select this check box to search in all the scopes. When this check box is selected, all the other search types are disabled.– Link Names: select this check box to search among the object names that V8 creates when calling the C++ runtime, see http://stackoverflow.com/questions/11202824/what-is-in-javascript. In the V8 Heap Tool Window, link names are marked with the % character (%<link name>).– Class Names: select this check box to search among functions-constructors.– Text Strings: select this check box to perform a textual search in the contents of the objects.– Snapshot Object IDs: select this check box to search among the unique identifiers of objects. V8 assigns such a unique identifier in the format to each object when the object is created and preserves it until the object is destroyed. This means that you can find and compare the same objects in several snapshots taken within the same session. In the V8 Heap Tool Window, object IDs are marked with the @ character (@<object id>).– Marks: select this check box to search among the labels you set to objects manually by clicking  on the toolbar of the Containment tab.

V8 Profiling Tool Window

The tool window is opened automatically when you stop the **Node.js** application you are profiling. If the window is already opened and shows the profiling data for another session, a new tab is added. Tabs that were opened automatically are named after the run configurations that control execution of the applications and collecting the profiling data.

If you want to open and analyze some previously saved profiling data, choose **V8 Profiling - Analyze V8 Profiling Log** on the main menu and select the relevant V8 log file `isolate-<session number>`. PyCharm creates a separate tab with the name of the log file.

Based on the collected profiling data, PyCharm builds three call trees and displays each of them in a separate pane. Having several call trees provides the possibility to analyze the application execution from two different points of view: on the one hand, which calls were time consuming ("heavy"), and on the other hand, "who called whom".

On this page:

- [Toolbar](#)
- [Context Menu](#)
- [Top Calls Pane](#)
- [Bottom-up Pane](#)
- [Top-down Pane](#)
- [Flame Chart](#)
- [Selecting a Fragment in the Timeline](#)
- [Synchronization in the Flame Chart](#)

Toolbar

Item	Tooltip	Description	Available in
	Jump to source	Click this button to navigate to the function definition.	Top Calls Bottom-up Top-down
	Filter	Click this button to filter out light calls and have PyCharm display only the calls that indeed cause performance problems. Using the slider, specify the minimum Total% or Parent% value for a call to be displayed and click Done.	Top Calls Bottom-up Top-down
	Expand Heavy Traces	When a tab for a profiling session is opened, by default the nodes with heaviest calls are expanded. While exploring the trees, you may like to fold some nodes or expand other ones. Click this button to restore the original tree presentation.	Top Calls Bottom-up Top-down
 	Expand All/ Collapse All	Click these buttons to expand or collapse all the nodes in the current pane.	Top Calls Bottom-up Top-down
 	Export to text file/ Export Timeline Chart	Click this button to save the call tree in the current pane to a text file or the current timeline chart to a <code>.png</code> file. Then specify the target file in the dialog box that opens.	All
	Help	Click this button to navigate to the Help topic for the tool window.	All
	Close	Click this button to close the V8 Profiling tool window.	All
	Zoom	Click this button to open the selected fragment of the flame chart in a separate tab and have the selected fragment enlarged to fit the tab width so you can examine the fragment with more details.	Flame Chart

Context Menu

The context menu is available only from items in the Top Calls, Bottom-up, and Top-down panes.

Item Description

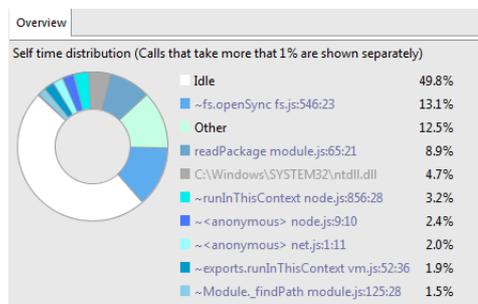
- Copy Call** Choose this option to copy the name of the selected function and the name of the file where it is defined to the Clipboard.
- Copy** Choose this option to copy the name of the selected function, the name of the file where it is defined, and the measurements data. This may be helpful if you want to compare the measurements for a function from two sessions, for example, after you make some improvements to the code.
- Compare with Clipboard** Choose this option to compare the selected with the contents of the Clipboard in the **Difference Viewer** that opens.
- Expand Node/** Choose these options to expand or collapse the selected tree node.
- Collapse Node**

Top Calls Pane

The Top Calls pane shows a list of performed activities sorted in the descending order by the **Self** metrics. For each activity PyCharm displays its **Total**, **Total%**, and **Self%** metrics. For each function call, PyCharm displays the name of the file, the line, and the column where the function is defined.

Calls	Total	Total %	Self %
Unknown	1153	98%	0%
JavaScript			
Function: ~<anonymous> jscs:1:11	730	62%	62%
Function: ~_rng uuid.js:20:29	83	7%	7%
Function: ~ScriptBreakPoint.set native debug.js:266:40	59	5%	5%
Function: ~exports.runInThisContext vm.js:68:36	45	3%	3%
Function: ~runInThisContext node.js:733:28	40	3%	3%
Function: ~FrameMirror.evaluate native mirror.js:895:40	16	1%	1%
Function: ~fs.statSync fs.js:707:23	14	1%	1%
LazyCompile: ~test native regexp.js:129:20	6	0%	0%
LazyCompile: ~captureStackTrace native messages.js:808:27	6	0%	0%
Function: ~fs.lstatSync fs.js:702:24	6	0%	0%
Function: ~fs.openSync fs.js:440:23	5	0%	0%
LazyCompile: Join native array.js:68:14	4	0%	0%

The diagram in the Overview pane shows distribution of self time for calls with the **Self%** metrics above 1%.



Bottom-up Pane

The Bottom-up pane also shows the performed activities sorted in the descending order by the **Self** metrics. Unlike the Top Calls pane, the Bottom-up pane shows only the activities with the **Total%** metrics above 2 and the functions that called them. This is helpful if you encounter a **heavy** function and want to find out where it was called from.

For each activity PyCharm displays its execution time in **ticks** and the **Of Parent** metrics. For each function call, PyCharm displays the name of the file, the line, and the column where the function is defined.

Top-down Pane

The Top-down pane shows the entire call hierarchy with the functions that are execution entry points at the top. For each activity PyCharm displays its **Total**, **Total%**, **Self**, and **Self%** metrics. For each function call, PyCharm displays the name of the file, the line, and the column where the function is defined. Some of the functions may have been optimized by V8, see [Optimizing for V8](#) for details.

- The functions that have been optimized are marked with an asterisk (*) before the function name.
- The functions that possibly require optimization but still have not been optimized are marked with a tilde (~) character before the function name. Though optimization may be delayed by the engine or skipped if the code is short-running, a tilde (~) points at a place where the code can be rewritten to achieve better performance.

Calls	Total	Total %	Self	Self %
Function: listOnTimeout timers.js:94:23	1084	92%	0	0%
Function: ~Module.runMain module.js:488:26	1084	92%	0	0%
Function: Module_load module.js:268:24	1084	92%	0	0%
Function: ~Module.load module.js:339:33	1070	91%	0	0%
Function: ~Module_extensions.js module.js:4	1069	90%	0	0%
Function: ~Module_compile module.js:36	1069	90%	0	0%
Function: ~<anonymous> jscs:1:11	1065	90%	730	62%
Function: ~require module.js:372:1	296	25%	0	0%
Function: ~Module.require mo	296	25%	0	0%
Function: Module_load moi	296	25%	0	0%
Function: ~Module.load	292	24%	0	0%
Function: ~Module.	291	24%	0	0%
Function: ~Modi	291	24%	0	0%
Function: ~<	273	23%	0	0%
Function:	273	23%	0	0%

Flame Chart

Use the multicolor chart in the Flame Chart tab to find where the application paused and explore the calls that provoked these pauses. The chart consists of four areas:

- The upper area shows a timeline with two sliders to limit the beginning and the end of a fragment to investigate.
- The bottom area shows a stack of calls in the form of a multicolor chart. When called for the first time, each function is assigned a random color, whereupon

every call of this function within the current session is shown in this color.

- The middle area shows a summary of calls from the **Garbage Collector**, the engine, the external calls, and the execution itself. The colors reserved for the **Garbage Collector**, the engine, the external calls, and the execution are listed on top of the area:



- The right-hand pane lists the calls within a selected fragment, for each call the list shows its duration, the name of the called function, and file where the function is defined.

Selecting a Fragment in the Timeline

To explore the processes within a certain period of time, you need to select the fragment in question. You can do it in two ways:

- Use the sliders:



- Click the **window** between two sliders and drag it to the required fragment:



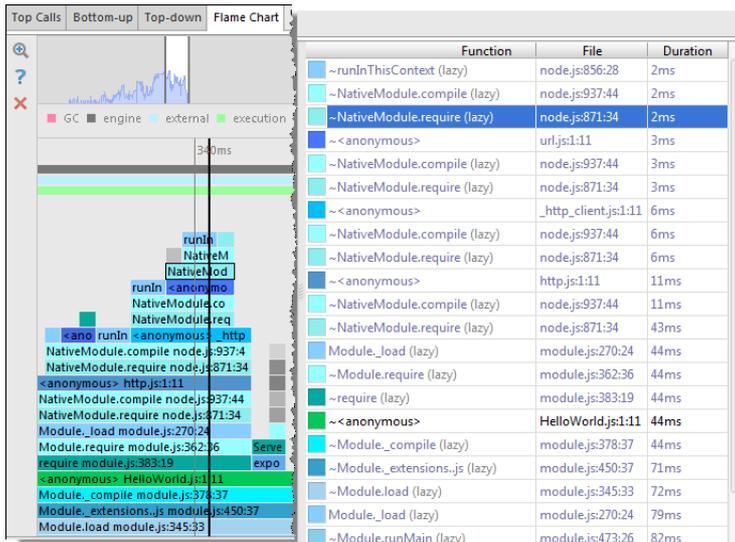
In either case, the multicolor chart below shows the stack of calls within the selected fragment.

To enlarge the chart, click the selected fragment and then click the Zoom button on the toolbar. PyCharm opens a new tab and shows the selected fragment enlarged to fit the tab width so you can examine the fragment with more details.

Synchronization in the Flame Chart

The bottom and the right-hand areas are synchronized: as you drag the slider in the bottom area through the timeline the focus in the right-hand pane moves to the call that was performed at each moment.

Moreover, if you click a call in the bottom area, the slider moves to it automatically and the focus in the right-hand pane switches to the corresponding function, if necessary the list scrolls automatically. And vice versa, if you click an item in the list, PyCharm selects the corresponding call in the bottom area and drags the slider to it automatically.



PyCharm supports navigation from the right-hand area to the source code of called functions, to the other panes of the tool window, and to areas in the flame chart with specific metrics.

- To jump to the source code of a called function, select the call in question and choose Jump to Source on the context menu of the selection.
- To switch to another pane, select the call in question, choose Navigate To on the context menu of the selection, and then choose the destination:
 - Navigate in Top Calls
 - Navigate in Bottom-up
 - Navigate in Top-down

PyCharm switches to the selected pane and moves the focus to the call in question.

- To have the flame chart zoomed at the fragments with specific metrics of a call, select the call in question, choose Navigate To on the context menu of the selection, and then choose the metrics:
 - Navigate to Longest Time
 - Navigate to Typical Time
 - Navigate to Longest Self Time
 - Navigate to Typical Self Time

You can also navigate to the stacktrace of a call to view and analyze exceptions. To do that, select the call in question and choose Show As Stacktrace.

PyCharm opens the stacktrace in a separate tab, to return to the Flame Chart pane, click V8 CPU Profiling tool window button in the bottom tool window.

Version Control Reference

This part provides miscellaneous information, related to common version control operations, and to VCS integrations:

- [CVS Reference](#)
- [Git Reference](#)
- [Mercurial Reference](#)
- [Perforce Reference](#)
- [Subversion Reference](#)
- [Apply Patch Dialog](#)
- [Commit Changes Dialog](#)
- [Configure Ignored Files Dialog](#)
- [Create Patch Dialog](#)
- [Enable Version Control Integration Dialog](#)
- [File Status Highlights](#)
- [New Changelist Dialog](#)
- [Patch File Settings Dialog](#)
- [Push Dialog \(Mercurial, Git\)](#)
- [Select Target Changelist Dialog](#)
- [Shelve Changes Dialog](#)
- [Show History for File / Selection Dialog](#)
- [Show History for Folder Dialog](#)
- [Revert Changes Dialog](#)
- [Unshelve Changes Dialog](#)

CVS Reference

In this part:

- [CVS Global Settings Dialog](#)
- [CVS Options Dialog](#)
- [CVS Root Dialog](#)
- [CVS Tool Window](#)

Check Out From CVS Dialog

VCS | Checkout From Version Control | CVS

The dialog consists of the following pages:

- [Select CS Configuration](#)
- [Select CVS Element to Check Out](#)
- [Select Checkout Location](#)
- [Check out to](#)

Select CVS Configuration

ItemDescription

List of available CVS configurations	Use this list to select the desired CVS configuration.
Configure	Click this button to define a new CVS configuration, or modify an existing one, in the CVS Roots dialog box.

Select CVS Element to Check Out

Use this page to select elements of the repository to check out. Next button is only available when an element is selected.

Select Checkout Location

Use this page to specify the target location for the artifacts to check out. All actions can be performed using the toolbar buttons, or context menu.

ItemShortcut Description

		Jump to the user's home directory.
		Jump to the project root directory.
		Create a new directory where the files will be checked out to.
		Delete the selected directory.
		Synchronize with external changes.
	N/A	Show Hidden Files and Directories

Check out to

Use this page to define CVS-specific checkout options.

ItemDescription

List of local paths	Select the local path to which the module name should be added.
Make new files read-only	Check this option to set read-only attribute for the files that did not exist locally but were checked out from the repository.
Prune empty directories	Check this option to delete empty directories from the repository.
Change keyword substitution to	Check this option to enable keyword substitution, and select the desired substitution mode from the drop-down list.

Configure CVS Root Field by Field Dialog

VCS | CVS | Configure CVS Roots

This dialog box opens when you click the Edit by Field button in the [CVS Roots](#) dialog box. Use this dialog box to specify the parameters for connecting to the CVS server and have PyCharm assemble them into a correct repository string according to the CVS root string syntax.

ItemDescription

Method	From this drop-down list, select the desired connection method. The available options are: <ul style="list-style-type: none">- pserver- ext- ssh (internal implementation)- local
User	In this text box, type your login to the CVS server.
Port	In this text box, specify the port to listen to on the CVS server host.
Host	In this text box, type the name of the host where the desired CVS server is located.
Repository	In this text box, type the path to the CVS repository relative to the host name.

CVS Roots Dialog

VCS | CVS | Configure CVS Roots

Use this dialog box to set up CVS roots. The dialog box is available for files and directories that are under CVS version control.

Common Options

ItemDescription

	Click this button to configure a new CVS root.
	Click this button to remove the selected CVS root configuration from the list.
	Click this button to create a copy of the selected CVS root.
Global Settings	Click this button to open the Global CVS Settings dialog box where you can set up CVS options at the global level.

CVS root In this text box, specify the CVS repository string according to the following syntax:

```
[ :method: ] [ [user] [ :password@ ] hostname [ :port ] ] /path/to/repository .
```

Tip Obtain the valid string from your system administrator or click the Edit by Field button to open the [Configure CVS Root Field by Field](#) dialog box where you can specify the mandatory connection parameters and have PyCharm **assemble** them into a correct repository string.

Edit by Field Click this button to open the [Configure CVS Root Field by Field](#) dialog box where you can specify the mandatory connection parameters and have them assembled into a CVS root string automatically.

Use version Use this section to specify the revision you want to synchronize your local working copy with. The available options are:

- HEAD revision: this option is suggested by default.
- By tag: select this option to access the revision with a specific tag. Type the desired tag in the text box or click the Browse button  and select the desired tag from the list. The list shows all the tags available on the CVS server according to the specified CVS root.
- By date: select this option to access the revision with a specific date and time stamp. Type the end date and time in the format `dd:mm:yy hh:mm:ss` or click the Browse button  and select the desired date from the calendar. This date and time are passed to the server with the GMT parameter.

Tip The controls in the area are available only after the CVS root text box is filled in with valid data.

Warning! If you perform update or checkout from the CVS repository with the By tag or By date option selected, the resulting working copy will be permanently restricted to the specified tag or date, until you force the update operation to reset this [sticky data](#).

Test connection Click this button to check that the specified settings ensure successful connection to the CVS server.

Additional Connection Settings

In this area, specify additional settings to flexibly configure connection to the CVS server. The contents of the area depends on the [connection method](#) set in the CVS root text box.

- [pserver](#)
- [ext](#)
- [ssh](#)
- [local](#)

Pserver

Warning! The settings specified in this area affect all CVS roots that use the pserver connection method.

ItemDescription

Password	In this text box, type the fully qualified path to the <code>.cvspass</code> file. Click the Browse button  to select the file in the corresponding dialog .
Connection timeout	In this text box, type the connection timeout in seconds.
Proxy Settings	See the Proxy Settings section below.

Ext

ItemDescription

Use internal `ssh` implementation Select this check box to access the [ssh area](#), where you can specify the SSH version to use, the port to listen to, and configure your private key and password.

Clear this check box to access the Ext Protocol Settings area with the following controls available:

- Path to external rsh: in this text box, specify the location of the external `rsh`. If necessary, click the Browse button  to select the necessary location in the [corresponding dialog](#).
- Path to private key file: in this text box, specify the location of the file with your private `ssh` key. If necessary, click the Browse button  to select the file in the [corresponding dialog](#).
- Additional parameters: in this text box, specify additional connection parameters.

Ssh

This area is also available when you have specified the [ext](#) connection method and selected the Use internal ssh implementation check box.

ItemDescription

SSH version	In this area, specify the SSH version to use. The available options are: <ul style="list-style-type: none">- Allow both- Force SSH1- Force SSH2
Port	In this text box, specify the <code>ssh</code> port to listen to.
Use Private key file	Select this check box, if you want to pass server authentication using a private <code>ssh</code> key. In the text box, specify the location of the file with your private <code>ssh</code> key. If necessary, click the Browse button  to select the file in the corresponding dialog .
Change password	Click this button to open the SSHPassword dialog box, where you can specify the password for the current CVS root.
Proxy Settings	See Proxy Settings section below.

Local

Tip PyCharm does not provide the server functionality. If you want to use a local CVS client, you need to install CVS on your local host computer and configure it to work as a server.

ItemDescription

Path to CVS client	In this text box, type the path to CVS client installed on the host computer and configured to work as a server. If necessary, click the Browse button  to select the necessary location in the corresponding dialog .
Server command	In this text box, specify the server command.

Proxy Settings

ItemDescription

Use proxy	Select this check box to enable using the Proxy server and access the Login, Password, Proxy host, and Proxy port text boxes below. This check box is available only in two cases: <ul style="list-style-type: none">- The connection method is <code>pserver</code> or <code>ssh</code>.- The connection method is <code>ext</code> and the Use internal ssh implementation check box is selected.
Protocol	Select the protocol to use. The available options are: <ul style="list-style-type: none">- HTTP- Socks4- Socks5
Login	In this text box, specify your user name.
Password	In this text box, specify the user password.
Proxy host	In this text box, specify the Proxy host name.
Proxy port	In this text box, specify the Proxy port number.

CVS Tool Window

VCS | Browse CVS Repository

View | Tool Windows | CVS

This tool window opens, when you browse a CVS repository, and enables you to view contents of the repository, check out files, browse changes, view annotations and navigate to source code.

ItemDescription

	Click this button to close the current tab.
	Click this button to open the selected file in the editor.
	Click this button to obtain a local copy of the selected file or directory.
	Click this button to open selected file in the editor with the annotations turned on. See Viewing Annotations .
	Click this button to see the changes that affect the selected file or directory, and that have been committed to the repository by a certain user, or during the specified period. The filtering information is entered in the Search Criteria dialog. Search results show in a dedicated tab of the Version Control tool window .
	Click this button to show reference page .

Import into CVS

VCS | Import into CVS

Use this dialog to import a directory into the specified CVS repository.

Select CVS Configuration

Use this page to select the target CVS root and [change its configuration](#), if necessary.

Select Directory to Import to

Use this page to select the target directory.

Select Import Directory

Use this page to select the directory to be imported. If you are importing an PyCharm project, make sure that the project file is located under that directory. Multiple selection is not available.

Customize Keyboard Substitution

Use this page to specify the [keyword substitution rule](#) for the files imported into the repository.

Import Settings

ItemDescription

Name in repository	<p>In this field, specify the name that corresponds to the <code>module</code> argument.</p> <p>Tip For import, <code>module</code> refers to the absolute location in the repository, not to a module name defined in the modules file.</p>
Vendor	<p>In this field, specify the name that corresponds to the <code>vendor-tag</code> argument. This tag is used as a branch tag. No checkouts will ever be done explicitly on it. Type a name that is relevant to the project, or just VENDOR.</p>
Release tag	<p>In this text box, specify the string that corresponds to the <code>release-tag</code> argument. The tag should refer to a version or a release number.</p> <p>Tip A tag name cannot contain punctuation marks.</p> <p>For example: <code>-release-2.2</code> is wrong</p> <p><code>release-2.2</code> is correct.</p>
Log message	<p>In this field, specify the string that corresponds to the <code>-m</code> command-line argument. By default, the field shows the previous log message; you can accept default, or type a new comment.</p>
Checkout after import	<p>Select this option to have CVS checkout run after completing the import operation.</p>
Make checked out files read-only	<p>Select this option to mark the checked out files as read-only after import. This option is disabled if the Checkout after import option is cleared.</p>

Rollback Actions With Regards to File Status

In this section:

- [Menu commands according to file status.](#)
- [Effect of rolling back local changes.](#)

Menu Commands According to File Status

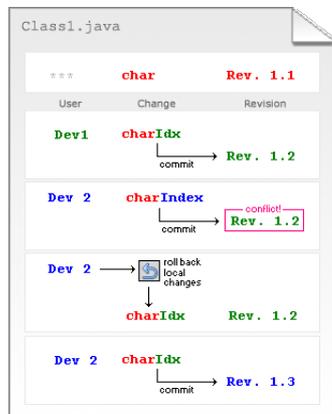
Depending on the file status, the Rollback Changes command will be *aliased* as shown in the following table:

File status	Rollback Command	Result
modified	Rollback Local Changes	all changes made in the file will be reverted, and the file will acquire the <i>up to date</i> status
deleted	Rollback Deletion	file will be restored both on the disk and in CVS with the <i>up to date</i> status
externally deleted	Rollback Deletion	file will be restored on the disk and assigned the <i>up to date</i> status
added	Rollback Creation	file will be deleted from disk
merged	Rollback Local Changes	all local changes will be dropped, changes from the repository will be accepted, and the file will be assigned the <i>up to date</i> status
merged with conflicts	Rollback Local Changes	all local changes will be dropped, changes from the repository will be accepted, and the file will be assigned the <i>up to date</i> status
unknown	Rollback Creation	file will be deleted from the disk

Effect of rolling back local changes

The effect of Rollback Local Changes may not be what you intuitively expect in terms of the revision you have locally after running the command.

The image below represents a file in CVS and a sequence of actions by two developers. It shows a simple example of what happens in terms of the local copy's CVS revision after rolling back conflicting local changes.



Here's what happens:

- The developer *Dev1* takes Revision 1.1 from the repository, modifies it, and commits changes to CVS.
- The developer *Dev2* doesn't know about *Dev1*'s changes and modifies the same code in the local copy of Revision 1.1. When *Dev2* commits these changes, he gets a message from CVS that the repository has changed. So he runs Update to synchronize, and his local copy is then updated to Revision 1.2, and CVS sets the *Merged with Conflicts* status on the file.
- *Dev2* decides to roll back local changes. He is left with a local copy of Revision 1.2.
- When *Dev2* commits the file to CVS, it becomes Revision 1.3 even though the content is identical to Revision 1.2.

Update Directory / Update File Dialog (CVS)

VCS | CVS | Update File

VCS | CVS | Update Directory Editor | context menu of a file | CVS | Update File

Project Tool Window | context menu of a file | CVS | Update File

Project Tool Window | context menu of a selection | CVS | Update Files

Project Tool Window | context menu of a folder | CVS | Update Directory

Project Tool Window | context menu of a selection | CVS | Update Directories

In these dialog boxes, configure synchronization of local files or folders with the repository.

ItemDescription

Branch Merging	<p>In this area, specify the repository branch to synchronize with.</p> <ul style="list-style-type: none">– Don't merge - select this option to synchronize with the counterpart of the current branch in the repository. This option is selected by default.– Merge with branch - select this option to have the local copy synchronized with a repository branch different from the counterpart of the current branch. In the first text box below, specify the branch to synchronize with. Type the branch name manually or click the Browse button  and select the desired branch from the list in the Select Tag dialog box that opens. This option is equivalent to the <code>-j</code> command-line option of the <code>update</code> command.– Merge two branches - select this option to have the local copy synchronized with the result of merging two repository branches different from the counterpart of the current branch. In the text boxes below, specify the branches to synchronize with. This option is equivalent to the <code>-j -j</code> command-line option of the <code>update</code> command.
Use Version	<p>In this area, specify the version of repository file(s) or folder(s) to synchronize with.</p> <ul style="list-style-type: none">– Default - select this option to have the local copy synchronized with the latest repository version.– By tag - select this option to have the local copy synchronized with a particular repository version. In the text box, specify the desired version number or tag. Type the version number or tag manually or click the Browse button  and select the desired branch from the list in the Select Revision or Tag dialog box that opens.<ul style="list-style-type: none">– When updating a single file, you can specify its repository counterpart either through a version number or a tag.– When updating an entire folder, its repository counterpart can be specified only through a tag.– By date - select this option to have the local copy synchronized with a repository version that was submitted on a particular date. Type the date manually or click the Calendar button  and select the desired date from the calendar pop-up window that opens with the current date selected by default. This option is equivalent to the <code>-D</code> command-line option of the <code>update</code> command.
Reset sticky data	<p>Select this check box to remove the date or tag restriction from the local file(s) or folder(s) to be updated. Such restrictions are set on files and folders checked out or previously updated with the By tag or By date options. This option is equivalent to the <code>-A</code> command-line option of the <code>update</code> command.</p>
Prune empty directories	<p>Select this option while updating a directory to have PyCharm remove the subfolders whose repository counterparts are empty. This option is equivalent to the <code>-P</code> command-line option of the <code>update</code> command.</p>
Change keyword substitution to	<p>Select this check box to have the default keyword expansion mode changed. From the drop-down list, choose the relevant substitution. This option is equivalent to the <code>-k</code> command-line option of the <code>update</code> command.</p>
Create new directories	<p>Select this option while updating a directory to have PyCharm create new local subfolders when any new subfolders have been created in the repository counterpart of the directory to be updated. This option is equivalent to the <code>-k</code> command-line option of the <code>update</code> command.</p> <p>By default, all the new subfolders will be picked including empty ones. To avoid creating empty subfolders, it is recommended that you select the Prune empty directories check box as well.</p>
Clean copy	<p>Select this check box to have PyCharm backup the local changes and replace the changed file(s) with their counterparts from the repository. This option is equivalent to the <code>-c</code> command-line option of the <code>update</code> command.</p>
Do not show this dialog in the future	<p>Select this option to have the <code>update</code> operation performed silently in the future. To have PyCharm show this dialog box before update again:</p> <ol style="list-style-type: none">1. Open the Version Control - Confirmation page of the Settings dialog box.2. In the Display Option dialogs when these commands are invoked area, select the Update check box.

Git Reference

In this part:

- [Checkout Dialog](#)
- [Clone Repository Dialog](#)
- [Line Separators Warning Dialog](#)
- [Merge Branches Dialog](#)
- [Pull Changes Dialog](#)
- [Push Rejected Dialog \(Git\)](#)
- [Rebase Branches Dialog](#)
- [Reset Head Dialog](#)
- [Git Reset Dialog](#)
- [Stash Dialog](#)
- [Tag Dialog](#)
- [Unstash Changes Dialog](#)
- [Update Project Dialog \(Git\)](#)
- [GitHub Integration Reference](#)

Checkout Dialog

VCS | Git | Checkout Branch

Use this dialog box to switch between existing branches and create new branches in a local repository.

ItemDescription

Git Root	From this drop-down list, select the path to the local repository in which you want to switch between existing branches or create a new branch.
Current Branch	This read-only field shows the name of the working branch in the selected local repository.
Checkout	<p>From this drop-down list, select the branch or tag to which you want to switch or which you want to copy to a new branch. To specify a particular commit, type its commit hash or use an expression, for example, of the following structure:</p> <pre><branch>~<number of commits backwards between the latest commit (HEAD) and the required commit> .</pre> <p>Refer to the Git commit naming conventions for details.</p>
Validate	<p>Click this button to check that the commit specified in the Checkout field exists and view which files were affected in it.</p> <p>The Validate button next to an editable field indicates that regular expressions are allowed in the field.</p>
Include Tags	Select this check box to have tags also shown in the Checkout drop-down list.
As New Branch	<p>In this text box, type the name of the branch to be created.</p> <p>You can also type the name of an existing branch and select the Override check box.</p>
Create Ref Log	Select this check box to have a reference log for the new branch created where all changes made to the branch references will be recorded.
Track Branch	Select this check box to track the parent of the new branch.
Override	<p>Select this check box if you want to checkout a branch to an existing branch.</p> <p>Use this option with care because the history of the target branch will be lost.</p>
Checkout	Click this button to switch to the specified existing branch or to create a new branch in the local repository.

Clone Repository Dialog

Checkout from Version Control | Git

Use the dialog box to set up a local repository by downloading the data from a remote repository.

ItemDescription

Git Repository URL In this text box, type the URL of the remote repository which you want to clone.

Test Click this button to check that connection to the remote repository has been established successfully.

Parent Directory In this text box, specify the directory where you want PyCharm to create a folder for your local Git repository. Type the path manually or click the Browse button  and choose the desired directory in the [dialog that opens](#).

Directory Name In this text box, type the name of the new folder into which the repository will be cloned.

Warning! The parent directory must not contain a folder with the specified name.

Clone Click this button to start cloning the specified repository.

Line Separators Warning Dialog

The dialog box opens when you attempt to commit a file with `CRLF` line endings, provided that you have enabled this behaviour by selecting the Warn if CRLF line separators are about to be committed checkbox in the [Git](#) page of the Settings dialog box.

ItemDescription

Cancel	Click this button to stop the commit operation whereupon you can fix the problem manually.
Commit As Is	Click this button to have the file with <code>CRLF</code> line separators committed.
Fix and Commit	Click this button to have PyCharm set the <code>core.autocrlf</code> attribute to <code>input</code> . As a result, all the <code>CRLF</code> separators will be replaced with <code>LF</code> separators and committed into the repository. Note that the reverse operation will not be performed when you download the files into your working directory, that is, no <code>CRLF</code> will appear in place of <code>LF</code> .

Merge Branches Dialog

VCS | Git | Merge Changes

Use this dialog box to specify arguments for merging branches in a local Git repository.

ItemDescription

Git Root	From this drop-down list, select the path to the local repository in which you want to merge branches.
Current Branch	This read-only field shows the name of the branch which is currently checked out in the selected local repository. This is the target branch, the changes from the selected source branches will be applied to it. The contents of the field depend on the selection in the Git Root drop-down list.
Branches to Merge	Use this list box to specify the source branches from which changes will be applied to the target branch. The list shows only those branches that contain applicable commits. Applicable commits are commits made after a branch separated from the target branch.
Strategy	From this drop-down list, select the merge strategy . The available options are: <ul style="list-style-type: none">– Default– Resolve - select this option if you need to resolve two HEADs, one of which is the current branch and the other HEAD is the branch which you selected in the Branches to Merge list. When this option is selected, the 3-way merge algorithm is applied.– Recursive - the default merge strategy for merging the current branch with one branch. Select this option if you need to resolve two HEADs by applying the 3-way merge algorithm and there are more than one common ancestor that can be used for 3-way merge.– Octopus - the default merge strategy for merging the current branch with more than one branch. Merges that require resolving conflicts manually are not performed.– Ours - select this option if you need to resolve several HEADs. The result of the merge is always the HEAD of the current branch.– Subtree - a modified recursive strategy. When two or more source branches are selected in the Branches to Merge list box, only the Octopus and Ours options are available.
No Commit	Select this check box if you need to inspect and, if necessary, adjust the result of merging before committing the result. The merge is performed but is not committed automatically, as if it failed.
No Fast Forward	Select this check box to generate a merge commit even if the merge resolved as a fast-forward .
Squash Commit	Select this check box to create a single commit on top of the current branch instead of merging one or more other branches. The working tree and index state are produced as if a real merge happened, but commit is not performed and the HEAD is not moved.
Add Log Information	Select this check box to have PyCharm populate, in addition to branch names, a log message with one-line descriptions from the actual commits that are being merged.
Commit Message	In this text box, provide a description for the commit. The text box is available only if the No Commit check box is not selected.
Merge	Click this button to initiate merging the specified branches in the local repository according to the defined settings.

Pull Changes Dialog

VCS | Git | Pull Changes

Use this dialog box to specify parameters for fetching changes from a remote repository and applying them to a local repository.

ItemDescription

Git Root	From this drop-down list, select the path to the local repository which you want to refresh.
Current Branch	This read-only field shows the name of the branch which is currently checked out in the selected local repository. The changes retrieved from the source remote repository will be applied to the displayed branch. The contents of the field depend on the selection in the Git Root drop-down list.
Remote	From this drop-down list, select the alias of the source remote repository.  To check that the selected alias corresponds to the correct URL address, expand the list - the URL addresses will be displayed explicitly.
Get Branches	Click this button to have the Branches to Merge list box display the actual list of branches in the source remote repository. 
Branches to Merge	Use this list box to specify the branches to which you want to apply the fetched changes.
Strategy	From this drop-down list, select the merge strategy . The available options are: <ul style="list-style-type: none">- Default- Resolve - select this option if you need to resolve two HEADs, one of which is the current branch and the other HEAD is the branch from which you pulled changes. When this option is selected, the 3-way merge algorithm is applied.- Recursive - the default merge strategy for pulling one branch. Select this option if you need to resolve two HEADs by applying the 3-way merge algorithm and there are more than one common ancestor that can be used for 3-way merge.- Octopus - the default merge strategy for pulling more than one branch. Merges that require resolving conflicts manually are not performed.- Ours - select this option if you need to supersede old development history of side branches. By applying this strategy any number of HEADs can be resolved but the result of the merge is always the HEAD of the current branch.- Subtree - a modified recursive strategy.
No Commit	Select this check box if you need to inspect and, if necessary, adjust the result of merging the fetched data before committing the result. The merge is performed but is not committed automatically, as if it failed.
No Fast Forward	Select this check box to generate a merge commit even if the merge resolved as a fast-forward .
Squash Commit	Select this check box to create a single commit on top of the current branch instead of merging one or more other branches. The working tree and index state are produced as if a real merge happened, but commit is not performed and the HEAD is not moved.
Add Log Information	Select this check box to have PyCharm populate, in addition to branch names, a log message with one-line descriptions from the actual commits that are being merged.
Pull	Click this button to initiate fetching changes from the specified remote repository and applying them locally according to the defined settings.

Push Rejected Dialog (Git)

VCS | Git | Push

The dialog box opens when [pushing a branch](#) is rejected due to lack of synchronization between your local repository and the remote storage. Use this dialog box to have PyCharm update the local branch using [rebase](#) or [merge](#) and to configure automatic update of conflicting branches in the future.

ItemDescription

Update not rejected repositories as well – Select this check box to have all the local repositories updated, no matter whether `push` has been rejected for all of them or not.
– When this check box is cleared, PyCharm will update only the repositories for which `push` was rejected.

The check box is available only if your project uses several repositories.

Remember the update method choice and silently update in future When this check box is selected, PyCharm saves the update method `rebase` or `merge` invoked by pressing the corresponding button and will apply it to update the conflicting local branch silently.
After you leave the dialog box, the Auto-update if push ... rejected check box in the [Git](#) page of the Settings dialog box is selected and the applied update method becomes default. To change this setting, clear the Auto-update if push ... rejected check box.

Merge Click this button to have the conflicting branch updated using `merge` .

Rebase Click this button to have the conflicting branch updated using `rebase` .

Rebase Branches Dialog

VCS | Git | Rebase

Use this dialog box to specify the branch to rebase, the new base, the rebasing mode, and configure the rebasing procedure.

ItemDescription

Git Root	From this drop-down list, select the path to the local repository in which you want to rebase a branch.
Branch	From this drop-down list, select the branch to rebase. By default, the current working branch is selected. If you specify another branch, it will be automatically checked out first.
Interactive	Select this check box to view and possibly edit a list of commits to be rebased.
Preserve Merges	Select this check box to have the possibility to recreate merges instead of ignoring them. The check box is available only when the Interactive check box is selected. Warning! When this check box is selected, Git does not support squashing commits.
Onto	Use this drop-down list to specify the new base for the selected branch. To specify the required commit, type its commit hash or use an expression, for example, of the following structure: <code><branch>~<number of commits backwards between the latest commit (HEAD) and the required commit> .</code> Refer to the Git commit naming conventions for details. If no commit is specified, the HEAD of the selected branch is used as the new base.
Validate	Click this button to check that the commit specified in the Onto field exists and view which files were affected in it.
From	Use this drop-down list to specify the commit starting from which you want to apply the branch to the new base. Type the required commit hash or use an expression, for example, of the following structure: <code><branch>~<number of commits backwards between the latest commit (HEAD) and the required commit> .</code> Refer to the Git commit naming conventions for details. To apply the entire branch, leave the field empty.
Validate	Click this button to check that the commit specified in the From field exists and view which files were affected in it.
Show Tags	Select this check box to have tagged commits included in the Onto and From drop-down lists.
Show Remote Branches	Select this check box to have branches in the remote repository included in the Onto drop-down list.
Merge Strategy	From this drop-down list, select the merge strategy . The available options are: <ul style="list-style-type: none">- Default- Resolve - select this option if you need to resolve two HEADs, one of which is the current branch and the other HEAD is the branch from which you pulled changes. When this option is selected, the 3-way merge algorithm is applied.- Recursive - the default merge strategy for pulling one branch. Select this option if you need to resolve two HEADs by applying the 3-way merge algorithm and there are more than one common ancestor that can be used for 3-way merge.- Octopus - the default merge strategy for pulling more than one branch. Merges that require resolving conflicts manually are not performed.- Ours - select this option if you need to supersede old development history of side branches. By applying this strategy any number of HEADs can be resolved but the result of the merge is always the HEAD of the current branch.- Subtree - a modified recursive strategy.
Do not use merge strategies	When this check box is selected, no merge strategy is applied during rebase.
Rebase	Click this button to initiate rebasing according to the defined settings.

The dialog box opens when you click Rebase in the [Rebase branches](#) dialog box, with the Interactive check box selected.

Use this dialog box to define the order in which commits should be applied, squash or edit commits before applying, skip commits that contain extraneous changes, etc.

ItemDescription

Action	Use this drop-down list to define the action that must be applied to the selected commit. The available options are: <ul style="list-style-type: none">– Pick: select this option to apply the selected commit as is.– Edit: select this option to edit the files and/or the commit message before applying the selected commit.– Skip: select this option to ignore the selected commit.– Squash: select this option to combine the selected commit with the previous commit.– Reword: select this option if you want to edit the commit message for the selected commit before applying it.– Fixup: select this option to combine the selected commit with the previous one, and construct a commit message from the previous commit message with the "fixup!" prefix.
Commit	This read-only field displays the hash of the selected commit.
Comment	This read-only field shows the comment supplied for the selected commit.
View	Click this button to view the files affected in the selected commit.
Move Up/Move Down	Use these buttons to change the order in which commits should be applied.

Reset Head Dialog

VCS | Git | Reset Head

Use this dialog box to specify the commit you want to reset the current HEAD to and define the reset method.

ItemDescription

Git Root	In this drop-down list, select the path to the local repository in which you want to reset the HEAD of a branch.
Current Branch	This read-only field shows the name of the branch which is currently checked out in the selected local repository. The contents of the field depend on the selection in the Git Root drop-down list.
Reset Type	Use this drop-down list to define the reset method to use. The available options are: <ul style="list-style-type: none">– Mixed - the default strategy. When this option is selected, the index is reset while the working tree is not, which means that changed files are preserved but not marked for commit. You are presented with a report of what has not been updated.– Soft - when this option is selected, the index and the working tree are not affected, only the HEAD pointer is moved to the specified commit. Your current state with any changes remains different from the commit you are switching to. All the changes are "staged" for committing.– Hard - when this option is selected, both the working directory and the index are changed to the specified commit.
To Commit	In this text box, specify the commit you want to reset the current HEAD to. Type the desired commit hash or use an expression, for example, of the following structure: <code><branch>-<number of commits backwards between the current HEAD and the required commit> .</code> Refer to the Git commit naming conventions for details.
Validate	Click this button to check that the commit specified in the To Commit field exists and view which files were affected in it.
Reset	Click this button to initiate resetting the HEAD to the specified commit according to the defined method.

Git Reset Dialog

View | Tool Windows | Log - Context menu of a selected commit - Reset Current Branch to Here

Use this dialog box to reset the current branch head to the selected commit, and update the working tree and the index according to the selected mode. The following options are available:

- Soft: The modified files will not be changed, and the differences will be staged for commit.
- Mixed: The modified files will not be changed, and the differences will be unstaged. This mode is used by default and is identical to the `git reset` command.
- Hard: The modified files will be reverted to the selected commit, and all uncommitted changes will be lost.
- Keep: The modified files will be reverted to the selected commit, but local changes will be preserved.

Stash Dialog

VCS | Git | Stash Changes

Use this dialog box to specify parameters for saving local modifications in a new stash.

Item	Description
Git Root	From this drop-down list, select the path to the local repository in which you want to stash modifications.
Current Branch	This read-only field shows the name of the branch which is currently checked out in the selected local repository.
Message	In this text box, provide a description of the new stash. This description will be displayed in the Unstash Changes dialog box to help you select the required stash.
Keep Index	Select this check box to have PyCharm bring the changes staged in the index to your working tree for examination and testing.
Create Stash	Click this button to have the specified local modification saved in a new stash.

Tag Dialog

VCS | Git | Tag Files

Use this dialog box to assign tags to commits and commit objects.

ItemDescription

Git Root	In this drop-down list, select the path to the local repository in which you want to tag a commit or a commit object.
Current Branch	This read-only field shows the name of the branch which is currently checked out in the selected local repository.
Tag Name	In this text box, type the name of the new tag.
Force	Select this check box to assign an existing tag to another commit or object. This check box is enabled only if you type the name of an existing tag in the Tag Name text box whereupon PyCharm displays an error message.
Commit	In this text box, specify the commit or object which you want to tag. Type the desired commit hash or use an expression, for example, of the following structure: <code><branch>~<number of commits backwards between the latest commit (HEAD) and the required commit> .</code> Refer to the Git commit naming conventions for details.
Validate	Click this button to check that the commit specified in the Commit field exists and view which files were affected in it.
Message	In the text box, provide a description of the commit to be tagged. Tip Fill in this text box to create an annotated tag.
Create Tag	Click this button to have the specified commit or commit object tagged.

Unstash Changes Dialog

VCS | Git | Unstash Changes

Use this dialog box to specify parameters for applying stashed modifications to your working copy.

ItemDescription

Git Root	From this drop-down list, select the path to the local repository in which you want to apply the stashed modifications.
Current Branch	This read-only field shows the name of the branch which is currently checked out in the selected local repository.
Stashes	From this list, select the stash that you want to apply. Each item is supplied with a number, an indication of the branch in which it was created, and the description provided during the stash creation.
View	Click this button to open the Paths affected in commit dialog box where you can learn which files are affected in the selected stash.
Drop	Click this button to remove the selected stash from the Stashes list.
Clear	Click this button to remove all the stashes from the Stashes list.
Pop Stash	Select this check box to have the selected stash removed from the list after it is applied.
Reinstate Index	Select this check box to have the stashed index modifications applied as well.
As New Branch	If you want to create a new branch on the basis of the selected stash, type the name of the new branch in this list box. When the list box is filled in, the Pop Stash and Reinstate Index check boxes are selected and disabled.
Apply Stash	Click this button to have the specified modifications applied to the working copy.
Pop Stash	Click this button to have the specified modifications applied to the working copy and then removed from the list. The button is available only if the Pop Stash check box is selected.

Update Project Dialog (Git)

VCS | Update Project

In this dialog box, choose the strategy to synchronize your local repository with the remote storage and to choose the way to clean your working tree before update.

ItemDescription

Update Type	<p>In this area, choose the method to use for synchronization. The strategy will be applied to all Git version control roots. The available options are:</p> <ul style="list-style-type: none">– Merge: choose this option to have the merge strategy applied. The result is identical with that of running <code>git fetch ; git merge</code> or <code>git pull --no-rebase</code>.– Rebase: choose this option to have the rebase strategy applied. The result is identical with that of running <code>git fetch ; git rebase</code> or <code>git pull --rebase</code>.– Branch Default: choose this option to have the default command for the branch applied. The default command is specified in the <code>branch.<name></code> section of the <code>.git/config</code> configuration file.
Clean working tree before update	<p>In this area, specify the method to save your changes while cleaning your working tree before update. The changes will be restored after the update is completed. The available options are:</p> <ul style="list-style-type: none">– Using Stash: choose this option to have the changes saved in a Git stash, so you can apply patches with stashed changed even outside PyCharm, because they are generated by Git itself.– Using Shelve: choose this option to have the changes saved on a shelf. Shelving is a PyCharm internal operation, patches generated from shelved changes are normally applied (unshelved) inside PyCharm. Applying shelved changes outside PyCharm is also possible but requires additional steps.
Do not show this dialog in the future	<p>Select this check box to have PyCharm update your project silently in the future using the specified Update Type and Clean working copy method. To have PyCharm show this dialog box before update again:</p> <ol style="list-style-type: none">1. Open the Version Control - Confirmation page of the Settings dialog box.2. In the Display Option dialogs when these commands are invoked area, select the Update check box.

GitHub Integration Reference

In this part:

- [Select Repository to Clone Dialog](#)
- [Share Project on GitHub Dialog](#)
- [Login to GitHub Dialog](#)
- [Create Gist Dialog](#)
- [Create Pull Request Dialog](#)

Use this dialog box to choose the source remote repository, the folder to clone the source to, and the name of the project to be created around the cloned data.

ItemDescription

Repository	From this drop-down list, select the source repository to clone the data from.
Parent Directory	In this text box, specify the directory where the local repository for cloned sources will be created. Type the path to the directory manually or click the Browse button  and choose the desired directory in the Select project destination folder dialog box that opens.
Directory Name	In this text box, specify the name of the folder where the repository will be created based on the cloned sources.
Clone	Click this button to start cloning the sources from the specified remote repository.

Use this dialog box to create a repository on the GitHub remote storage and publish your project sources there.

ItemDescription

New repository name In this text box, type the name of the repository to be created. By default, PyCharm suggests the name of the current project.

Description In this text box, describe the main functionality implemented in the project.

Private Select this check box to suppress access to the repository for other users.

Warning! The check box is disabled for free accounts.

Share Click this button to have PyCharm initiate repository creation and afterwards upload the project sources to the new repository.

Use this dialog box to log on the [GitHub](#) remote storage using your account credentials or to create a GitHub account if you do not have it yet.

ItemDescription

Host	Specify the URL of your GitHub repository.
Auth Type	Use this drop-down list to select how you want to be authenticated on GitHub. <ul style="list-style-type: none">– Password. If this option is selected and you have two-factor authentication enabled in your GitHub account settings, you will be asked to enter an authentication code each time PyCharm requires you to log in to your GitHub account.– Token (recommended by GitHub for authentication from third-party applications, as it does not require PyCharm to remember your password).
Save password	This option is only available if Password is selected as authentication method. Select this option if you want PyCharm to remember your password.
Save token	This option is only available if Token is selected as authentication method. Select this option if you want PyCharm to remember your GitHub authentication token.
Login	In this text box, type your GitHub logon name. This field is only available if Password is selected as authentication method above.
Password	In this text box, type your GitHub account password. This field is only available if Password is selected as authentication method above.
Token	Specify your personal access token. This field is only available if Token is selected as authentication method above.
Sign up	Click this link to open the Sign up for GitHub page where you can create a GitHub account.

Create Gist Dialog

<Context menu of an open file in the editor> | Create gist

<Context menu of a file in the Project view> | Create gist

<Context menu of a folder in the Project view> | Create gist

<Context menu of a group of files or folders selected in the Project view> | Create gist

<Context menu of a console> | Create gist

Use this dialog box to save source code and console output in [gists](#).

Warning! The menu item and the dialog box are only available when you are logged on GitHub and a file is opened in the editor.

ItemDescription

Description In this text box, provide a brief description of the code snippet to be published.

By default, the new gist will be **public**, that is, visible for all registered GitHub users and your login will be shown as the gist's owner. You can change this default behaviour by selecting the check boxes below.

Private Select this check box to make the new gist available for you only and invisible for other GitHub users.

Anonymous Select this check box to make the new gist visible for all registered GitHub users but without displaying your login.

Open in browser

- Select this check box to have the new gist opened in the [default PyCharm browser](#).
- To have PyCharm display the URL address of the new gist so you can access it later, clear this check box.

Creating [pull requests](#) is a nice way to share your code with the community and to follow the collaborative development. With it you can tell others about the changes you've made and ask for comments, review, or just share the knowledge. With all this going on, a pull request appears in the original repository only after approval. A pull request can be prepared right from PyCharm without switching to the browser.

ItemDescription

Base Fork	In this field, specify the repository to apply the changes to. PyCharm attempts to retrieve all the relevant repositories and shows them in the drop-down list. Choose the repository from the drop-down list or click Select Other Fork and from the Form Owner drop-down list choose the name of the user who is the owner of the target repository.
Base Branch	In this field, specify the branch to apply the changes to. Click Show Diff to view the list of commits to be included in the pull request. To view the details of a commit, select it and switch to the Log tab which shows a list of files included in the selected commit list. See also Merging, Deleting, and Comparing Branches .
Title	In this field, specify the name of your pull request.
Description	In this field, optionally provide a brief description of the changes to be applied through the request.

Mercurial Reference

In this part:

- [Clone Mercurial Repository Dialog](#)
- [Create Mercurial Repository Dialog](#)
- [Merge Dialog \(Mercurial\)](#)
- [New Bookmark Dialog](#)
- [Pull Dialog](#)
- [Switch Working Directory Dialog](#)
- [Tag Dialog \(Mercurial\)](#)

Clone Mercurial Repository Dialog

Checkout from Version Control | Mercurial

Use the dialog box to set up a local repository by downloading the data from a remote repository.

ItemDescription

Mercurial Repository URL In this text box, type the URL of the remote repository which you want to clone.

Test Click this button to check that connection to the remote repository has been established successfully.

Parent Directory In this text box, specify the directory where you want PyCharm to create a folder for your local Mercurial repository. Type the path manually or click the Browse button  and choose the desired directory in the [dialog that opens](#).

Directory Name In this text box, type the name of the new folder into which the repository will be cloned.

Warning! The parent directory must not contain a folder with the specified name.

Clone Click this button to start cloning the specified repository.

Create Mercurial Repository Dialog

VCS | Create Mercurial Repository

Use this dialog box to create a local Mercurial repository in the folder of your choice.

ItemDescription

Create repository for the whole project

Select this option to have a repository initialized in the project root directory. This option is helpful if you want to put the entire project under Mercurial control.

Select where to create repository

Select this option to have a repository initialized in one of the folders below the project root. With this option, you can have folders of your project under control of different version control systems. In the text box below, specify the folder to create the repository in. Type the path manually or click the Browse button  and choose the desired folder in the [dialog that opens](#).

Merge Dialog (Mercurial)

VCS | Mercurial | Merge

context menu of the Editor - Mercurial | Merge

VCS | Mercurial | Branches - <branch name> - Merge

context menu of the Editor - Mercurial | Branches - <branch name> - Merge

Use this dialog box to merge the [current working directory](#) to a [named branch](#), [light-weight branch \(bookmark\)](#), or a specific [changeset](#) identified by a [tag](#), [hash](#), or [revision number](#).

By default, **Mercurial** requires that before merge the current working directory should be **clean**, that is, it should not contain any uncommitted changes. Otherwise the merge operation fails and PyCharm shows the corresponding error message. The message also recommends that you clean the current working directory by running the `hg merge <target branch, bookmark, or changeset> -C` to discard the uncommitted changes.

ItemDescription

Repository From this drop-down list, choose the repository to run the merge in. The contents of the Branch, Tag, and Bookmark drop-down lists are updated to show the branches, tags, and bookmarks that are available in the selected repository.

Merge with In this area, choose the branch, bookmark, or changeset to merge with.

- Branch: choose this option to switch to another line of development identified by a [branch name](#) and merge to the [branch head](#). Choose the desired branch from the drop-down list which shows all the named branches available in the current repository.
- Tag: choose this option to merge to a [changeset](#) to which you have previously assigned a [tag identifier](#). Choose the relevant tag from the drop-down list. The list shows both local tags (from `.hg/localtags`) and global tags (from `.hgtags`).
- Bookmark: choose this option to switch to another line of development which is identified by a [bookmark](#) and merge to its [head](#). Choose the relevant bookmark from the drop-down list which shows all the available light-weight branches in the current repository.
- Revision: choose this option to merge to a specific changeset identified by its [hash](#) or [revision number](#). In the text box, type the relevant revision number or paste the hash. To copy a hash, open the Log tab of the Version Control tool window, select the relevant branch and revision, and then choose Copy Hash on the context menu of the selection.

New Bookmark Dialog

VCS | Mercurial | Branches | New Bookmark

context menu of the Editor - Mercurial | Branches | New Bookmark

Use this dialog box to establish a new light-weight Mercurial branch (bookmark). The bookmark will immediately appear in the Branches pop-up. You can create both **active** and **inactive** bookmarks. By default, PyCharm creates an **active** bookmark, so you are immediately switched to the new bookmark and it is marked with a tick in the Branches pop-up. If you do not want to want to move your development to the new bookmark right now, you can create an **inactive** bookmark and switch to it later. Note, that tracking and updating is available only for the bookmark that is currently active. You can have only one active bookmark. For details, see <http://mercurial.selenic.com/wiki/Bookmarks> and [Managing Mercurial Branches and Bookmarks](#).

ItemDescription

Bookmark Name	In this text box, type the name of the bookmark. This identifier will always point at the head of the new light-weight branch as you commit changes. You can use this name to identify the head of the relevant light-weight branch when during update or merge .
Inactive	<ul style="list-style-type: none">– Clear this check box to activate the new bookmark and thus enable tracking and updating the light-weight branch the bookmarks identifies. The check box is cleared by default.– Select this check box to have an inactive bookmark created, that is, to remain in the current light-weight branch (bookmark) or named branch and switch to the new bookmark later.

Pull Dialog

VCS | Mercurial | Pull Changesets

Use this dialog box to specify parameters for fetching changes from a remote repository and applying them to a local repository.

ItemDescription

Pull From

In this text box, specify the URL address of the remote repository to fetch the changes from.

Switch Working Directory Dialog

VCS | Mercurial | Update to

Use this dialog box to update the [current working directory](#) to a [named branch](#), [light-weight branch \(bookmark\)](#), or a specific [changeset](#) identified by a [tag](#), [hash](#), or [revision number](#).

By default, **Mercurial** requires that before update the current working directory should be **clean**, that is, it should not contain any uncommitted changes. Otherwise the update operation fails and PyCharm shows the corresponding error message. The message also recommends that you clean the current working directory by running the `hg update <target branch, bookmark, or changeset> -C` to discard the uncommitted changes.

ItemDescription

Repository	From this drop-down list, choose the repository to run the update in. The contents of the Branch, Tag, and Bookmark drop-down lists are updated to show the branches, tags, and bookmarks that are available in the selected repository.
Switch to	<p>In this area, choose the branch, bookmark, or changeset to switch to.</p> <ul style="list-style-type: none">– Branch: choose this option to switch to another line of development identified by a branch name and update to the branch head. Choose the desired branch from the drop-down list which shows all the named branches available in the current repository.– Tag: choose this option to update to a changeset to which you have previously assigned a tag identifier. Choose the relevant tag from the drop-down list. The list shows both local tags (from <code>.hg/localtags</code>) and global tags (from <code>.hgtags</code>).– Bookmark: choose this option to switch to another line of development which is identified by a bookmark and update to its head. Choose the relevant bookmark from the drop-down list which shows all the available light-weight branches in the current repository.– Revision: choose this option to update to a specific changeset identified by its hash or revision number. In the text box, type the relevant revision number or paste the hash. To copy a hash, open the Log tab of the Version Control tool window, select the relevant branch and revision, and then choose Copy Hash on the context menu of the selection.
Overwrite locally modified files (no backup)	<p>If you are going to update to another branch, bookmark, or changeset and you have any uncommitted changes in the current line of development, technically there can be two ways to treat them. The uncommitted changes can be either committed before update or abandoned (cleaned).</p> <p>By default, Mercurial requires that before update the current working directory should be clean, that is, it should not contain any uncommitted changes. Otherwise the update operation fails and PyCharm shows the corresponding error message. The message also recommends that you clean the current working directory by running the <code>hg update <target branch, bookmark, or changeset> -C</code> to discard the uncommitted changes. Use the Overwrite locally modified files (no backup) check box to prevent failures during update when the current working copy is not clean.</p> <ul style="list-style-type: none">– Select the check box to abandon any uncommitted local changes.– Clear the check box if you are sure that the current working directory is clean.

Tag Dialog (Mercurial)

VCS | Mercurial | Tag Repository

context menu of the Editor - Mercurial | Tag Repository

Use this dialog box to create a global tag that identifies the **tip** of a repository, which is the most recently changed **head** in this repository. The created tag will be stored in the file `.hgtags` and tracked by Mercurial.

ItemDescription

Select repository to tag	From this drop-down list, choose the repository whose tip you want to tag. The list shows the location of the repositories under the project root.
--------------------------	---

Tag name	In this text box, type the name of the tag. By this name, you will be able to find the tag in the Log Tab of the Version Control tool window.
----------	---

Perforce Reference

This feature is supported in the Professional edition only.

In this part:

- [Edit Jobs Linked to Changelist Dialog](#)
- [Integrate File Dialog \(Perforce\)](#)
- [Link Job to Changelist Dialog](#)
- [Perforce Options Dialog](#)
- [Update Project Dialog \(Perforce\)](#)

Edit Jobs Linked to Changelist Dialog

This feature is supported in the Professional edition only.

VCS | Show Changes View - Local

View | Tool Windows | Changes - Local

VCS | Commit Changes

The dialog box opens in the following cases:

- When you select a changelist and then select Edit Associated Jobs from the context menu.
- When you click the  button in the Perforce area of the [Commit Changes](#) dialog box.

Use the dialog box to search for Perforce jobs, link jobs to the selected changelist, and detach currently linked jobs.

ItemTooltip and Shortcut	Description
--------------------------------	-------------

	Unlink selected jobs	Click this button to detach the selected job from the changelist.
	Search	Click this button to open the Link Job to Changelist dialog box, where you can search for available jobs, view their details, and link the desired job to the changelist.
	Find and link job matching the pattern	Click this button to start quick search for the job that matches the pattern specified in the text box and attach the job to the changelist. In the text box, specify the exact name of the desired job or a search pattern according to the Perforce jobs syntax rules .

Tip If only one job matching the pattern is found, it is attached to the changelist automatically. Otherwise, to select a job among several available jobs, click  and find the desired job using the [Link Job to Changelist](#) dialog box.

Close	Click this button to save the specified settings and leave the dialog box.
-------	--

Integrate File Dialog (Perforce)

VCS | Integrate Project

Context menu of a file or directory | Perforce | Integrate File/Directory

Use this dialog box to integrate changelists from one branch spec to another.

ItemDescription

Branch Spec	Select the branch spec that will be used for change integration. Consider the following: <ul style="list-style-type: none">– If the Reverse option is enabled, changes are integrated from the selected branch to the local copy.– If the Reverse option is disabled, changes are integrated from the local copy to the selected branch.
Integrate changelist	Use this option to invoke the Changes Browser, where you can select the changelist that will be integrated into the current branch/local copy.
Store Changes To Changelist	Specify the changelist where the integrated changes should be stored.
Revert unchanged files before sync (p4 revert -a)	Select this option to revert unchanged files.
Run resolve automatically after the sync (p4 resolve -am)	Select this option to automatically resolve the files that can be resolved without conflicts.

Link Job to Changelist Dialog

This feature is supported in the Professional edition only.

The dialog box opens when you click the  button in the [Edit Jobs Linked to Changelist](#) dialog box.

Use this dialog box to search for available jobs, view their details, and link jobs to the changelist.

Tip When you need to attach only one job to a changelist, you can use the [quick search](#) functionality. This requires that you know the exact name of the job or at least can specify a search pattern for it.

Specify search parameters

Use the controls in this area for specifying various criteria to limit the search output. Follow the Perforce jobs [syntax rules](#). The specified values are joined in the generated command line query via the `AND` operation.

Tip At least one of the fields should be filled in.

ItemDescription

Job name pattern	In this text box, type the desired job name search pattern.
Status	Use this drop-down list to specify the status of the job you are looking for. The available options are: – * - when this option is selected, all the jobs that match the remaining search criteria are displayed, regardless of their job statuses. Tip If you specify Status as the only criterion and select this option, all the jobs that are currently present on the Perforce server will be retrieved. – Open – Closed – Suspended
User name pattern	In this text box, specify the search pattern or the exact name of the user who created the desired job.
Date before/Data after	Use these text boxes to specify the time period the desired job is created in. The appropriate formats are <code>yyyy/mm/dd</code> or <code>yyyy/mm/dd:hh:mm:ss</code> .
Description pattern	In this text box, type the desired job description search pattern.
Search	Click this button to start searching for jobs that match the specified criteria.

Search results

Use this area to view the details of found jobs, select the desired job, and attach it to the changelist.

ItemDescription

Search results	The list contains the jobs found according to the specified search criteria. When you select a job, the read-only area shows its details.
OK	Click this button to link the selected job to the changelist.

Perforce Options Dialog

This feature is supported in the Professional edition only.

File | Settings | Version Control | Configure | Perforce for Windows and Linux

PyCharm | Preferences | Version Control | Configure | Perforce for macOS

Ctrl+Alt+S



In this dialog box, configure connection to the specified Perforce server.

ItemDescription

Perforce is online	Select this check box to enable establishing connection to the Perforce server. If there is a connection but the server does not respond (for example, because the server is backing up currently), you can disable such attempts by clearing this check box, or PyCharm will suggest to do that after a timeout. After that the version control-specific operations will be disabled.
Use P4CONFIG or default connection	Use server information and user credentials from the P4CONFIG environment variable or default Perforce client connection. Otherwise, specify port, user, client, and password.
Charset	Select the charset corresponding to the one set on the server.
Dump Perforce commands to <i>IDEA_Home\bin\p4.output</i>	Log Perforce commands in the specified file.
Use login authentication	Toggle authentication.
Try to log in silently	Skip prompt dialog. This option works, when login authentication is required.
Use native Perforce API	Speed up connection to the server using a special library.
Path to P4 executable	Specify the path to the Perforce client executable file.
Test connection	Make sure that connection to the Perforce server is established.
Show branching history	See branches for files in the dialogs showing File History. When you work with several branches, it is recommended to enable this option so that the file branches are correctly displayed.

Update Project Dialog (Perforce)

This feature is supported in the Professional edition only.

VCS | Update Project

Ctrl+T

Context menu of a file or directory | Perforce | Update File/Directory

VCS | Perforce | Update File/Directory

Use this dialog box to update the local working copy of a file, directory, or project with a revision from the repository.

ItemDescription

Revert unchanged files before sync (p4 revert -a)	Select this check box to discard changes made to open files. This option corresponds to the Perforce <code>revert</code> command. See p4 revert reference for details.
Force sync (-f)	Select this check box to forcibly copy files from the depot into the workspace. This option corresponds to the Perforce <code>sync</code> command. See p4 sync reference for details.
Run resolve automatically after the sync (p4 resolve -am)	Select this check box to resolve conflicts between file revisions. This option corresponds to the Perforce <code>resolve</code> command. See p4 resolve reference for details.
Do not show this dialog in the future	If this check box is selected, the specified actions will be performed silently in future.

Subversion Reference

In this section:

- [Authentication Required](#)
- [Changes Browser](#)
- [Check Out From Subversion Dialog](#)
- [Configure Subversion Branches](#)
- [Create Branch or Tag Dialog \(Subversion\)](#)
- [Import into Subversion](#)
- [Integrate Project Dialog \(Subversion\)](#)
- [Integrate to Branch](#)
- [Lock File Dialog \(Subversion\)](#)
- [Mark Resolved Dialog \(Subversion\)](#)
- [Select Branch](#)
- [Select Repository Location Dialog \(Subversion\)](#)
- [Set Property Dialog \(Subversion\)](#)
- [Subversion Options Dialog](#)
- [Subversion Working Copies Information Tab](#)
- [SVN Repositories](#)
- [Update Project Dialog \(Subversion\)](#)

Authentication Required

VCS | [Browse VCS Repository](#) | [Browse Subversion Repository](#)

Use this dialog to specify your credentials and gain access to the Subversion repository. The dialog is opened when you add a new repository location, or attempt to browse a repository.

ItemDescription

Authentication realm	This read-only area displays the repository name and URL.
User name	By default, this field shows the current user name. You can change the name as required.
Password	Type the password for your Subversion account.
Save credentials	Select this check box to preserve the specified user name and password.

Changes Browser

VCS | Integrate Project

This dialog opens when you select the Specified option in the [Integrate Project dialog](#), and click the  button.

Use this dialog to select which revision to use in integration.

The Changes Browser dialog consists of the following areas:

- [Changes list](#)
- [Commit message](#)
- [Commit details](#)

Changes list

This pane contains the list of all changes to your project. For each change, there is the revision number, the user who made the change, the date and the description. You can sort the list by this information by clicking the corresponding column header.

You can click the Older and Newer buttons to display the previous/next list of change.

Commit message

This read-only area shows the commit message for the selected revision.

Commit details

This pane shows a list of files that were modified in the selected revision.

Toolbar

ItemTooltip and Shortcut	Description
--------------------------------	-------------

	Show Differences	Click this button to open the Differences dialog that points at the differences between the selected revision and the previous revision of the selected file. 
	Show Diff with Local	Click this button to open the Differences dialog that points at the differences between the selected file in the current revision and in your local working copy.
	Edit Source	Click this button to open the source code of the selected file in the editor. 
	Open Repository Version	Click this button to open the repository version of the selected file for editing.
	Revert Selected Changes	Click this button to roll back the changes in the selected file.
	Compare Subversion Properties	Click this button to view the differences in properties between the selected revision of the selected file and your local working copy.
	Group by Directory	Click this button to transform a flat list of files into a tree of packages with files. 
	Expand All/Collapse All	Click this button to expand/collapse all nodes. Note that these buttons are only available only when tree-view is enabled. 

Check Out From Subversion Dialog

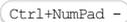
VCS | Checkout From Version Control | Subversion

Use this dialog box to create local working copies.

In this topic:

- [Toolbar](#)
- [Main Controls](#)
- [Context Menu](#)

Toolbar

ItemTooltip and Shortcut	Description
	Add Repository Location Click this button to configure a new repository location .
	Edit Location URL Click this button to edit the URL address of the selected repository.
	Discard Location Click this button to discard selected repository location.
	Show/Hide Details Click this button to display the details for each node below the repository location (changelist number, user name, date and time of the last change).
	Refresh Click this button to refresh the view. 
	Collapse All Click this button to have all the nodes below all the repository locations collapsed. 

Repositories

In this area, manage the available repositories and select the locations to check out contents from.

ItemDescription
Repositories Use this tree view to explore and manage the available repository locations. If necessary, right-click a node and choose the relevant item from its context menu.
Checkout Click this button to check out the contents of the selected node to the specified location.

Context Menu

ItemDescription
New Select this menu item to configure a new repository location , or a new remote folder in the selected repository location.
Checkout Select this menu item to check out the contents of the selected node.
Compare With Select this menu item to compare the selected node with the specified branch.
Browse Changes Select this menu item to view changes that match the specified criteria (author, time range, and revision).
Import Select this menu item to import a directory into the repository. You can select the directory you want to import from the Import Directory dialog that opens.
Export Select this menu item to export the contents of the selected repository or folder to the specified destination. The exported contents are not under version control.
Branch or Tag Select this menu item to create a branch or tag of the selected folder.
Move or Rename Select this menu item to change name of the selected folder.
Delete Select this menu item to delete the selected folder from the repository location. 
Copy URL Select this menu item to put the URL string to the clipboard. 
Refresh Select this menu item to synchronize to the repository.
Edit Location URL Select this menu item to edit the selected repository location.
Discard Location Select this menu item to discard the selected repository location.

This dialog appears when you have selected the repository you want to check out and the destination folder.

ItemDescription

Checkout	This read-only field shows the selected source repository.
Destination	From this list, select the directory to create the working copy in. Choose one of the available folders or click the Browse button  and select the relevant folder in the dialog box that opens.
Update / Switch to revision	In this area, specify the revision to check out. The available options are: <ul style="list-style-type: none">- HEAD - select this option to have the latest revision checked out.- Specified - select this option to have PyCharm check out a specific earlier revision. Type the revision number in the text box or click the Browse button  and select the relevant revision from the Changes Browse that opens.
Depth	Use this drop-down list to specify the range of recursion into subdirectories. The available options are: <ul style="list-style-type: none">- Empty: select this option to involve only the current file.- Files: select this option to involve the files in the folder.- Immediates: select this option to involve direct children of the current file.- Infinity: select this option to enable full recursion.
Include external locations	Select this check box to have externals included in the working copy.

Configure Subversion Branches

[Select Branch pop-up dialog box](#) - Configure Branches

Use this dialog box to compose a list of branches you work with.

ItemDescription

Trunk location	In this text box, specify the URL address of the trunk in the repository. If necessary, click the Browse button  to open the Select Repository Location dialog box and select the required trunk in the repository structure tree.
Branch locations	In this list, select the URL address of the folders in which the required branches are stored.
Add	Click this button to add a branch to the list. The Select Repository Location dialog box opens where you can select the required branch in the repository structure tree.
Remove	Click this button to remove the selected branch from the list.

Create Branch or Tag Dialog (Subversion)

VCS | Subversion | Create Branch or Tag

In this dialog box, set the arguments for creating a branch or a tag on the basis of a local working copy or a repository version.

ItemDescription

Copy from	In this section, specify the source folder to create a branch or tag from. The source of the copy can be taken from the local working copy or from the repository.
Working Copy	Click this option to create a branch or tag on the basis of your local working copy. Type the path in the text box or click the Browse button  and select the desired directory in the dialog that opens .
Repository Location	Click this option to create a branch or tag on the basis of the repository. Do one of the following: <ul style="list-style-type: none">– Type the URL of the repository location in the text box.– Click the browse button and select the source repository location.– Click the  button to use the project home directory.
Revision	In this section, specify the source revision to create a branch or tag from. The available options are: <ul style="list-style-type: none">– HEAD - select this option to have a branch or tag created on the basis of the HEAD revision.– Specified - select this option to have a branch or tag created on the basis of a specific revision. Type the revision number in the text box manually or click the Browse button  and select the desired revision in the Changes Browser dialog box, that opens.
Copy to	Use this section to define the target folder for a branch or tag. The available options are: <ul style="list-style-type: none">– Branch or Tag - select this option to have the selected revision copied to a specific branch or tag. <p>In the Base URL text box, specify the base URL of the branch or tag. In the Name text box, specify the name of the new branch.</p> <ul style="list-style-type: none">– Any location - select this option to have to have the selected revision copied to a location of your choice. In the text box below, specify the URL of any valid location. Type the URL manually or click the Browse button  and specify the desired location in the Select Repository Location dialog box, that opens.
Comment	Type some meaningful description in the text area.

Import into Subversion

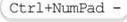
VCS | Import into Subversion

Use this dialog box to specify the options for importing data into Subversion.

In this topic:

- [Toolbar](#)
- [Main Controls](#)
- [Context Menu](#)

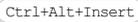
Toolbar buttons

ItemTooltip and Shortcut	Description
	Add Repository Location Click this button to configure a new repository location .
	Edit Location URL Click this button to edit the URL address of the selected repository.
	Discard Location URL Click this button to discard the selected repository location and remove it from the list.
	Show/Hide Details Click this button to display the details for each node below the repository location (changelist number, user name, date and time of the last change).
	Refresh Click this button to refresh the view. 
	Collapse All Click this button to have all the nodes below all the repository locations collapsed. 

Main Controls

ItemDescription	Description
Repositories	Use this tree view to explore and manage the available repository locations. Right-click nodes and examine context menus.
Import	Click this button and in the dialog that opens , select the directory whose contents should be imported to the selected repository location.

Context Menu

ItemDescription	Description
New	Select this menu item to configure a new repository location , or a new remote folder in the selected repository location.
Checkout	Select this menu item to check out the contents of the selected node.
Compare With	Select this menu item to compare the selected node with the specified branch.
Browse Changes	Select this menu item to view changes that match the specified criteria (author, time range, and revision).
Export	Select this menu item to export the contents of the selected repository or folder to the specified destination. The exported contents are not under version control.
Branch or Tag	Select this menu item to create a branch or tag of the selected folder.
Move or Rename	Select this menu item to change name of the selected folder.
Delete	Select this menu item to delete the selected folder from the repository location.
Copy URL	Select this menu item to put the URL string to the clipboard. 
Refresh	Select this menu item to synchronize to the repository.
Edit Location URL	Select this menu item to edit the selected repository location.
Discard Location	Select this menu item to discard the selected repository location.

Integrate Project Dialog (Subversion)

VCS | Integrate Project

Use this dialog box to integrate differences between two branches in the Subversion repository into a local working copy.

ItemDescription

Source 1	In this text box, specify the URL address of the first branch to compare. If necessary, click the Browse button  and select the desired URL from the Select Repository Location dialog box.
Source 2	In this text box, specify the URL address of the second branch to compare. If necessary, click the Browse button  and select the desired URL from the Select Repository Location dialog box.
Revision	For each source, specify the revision to use. The possible options are: <ul style="list-style-type: none">- HEAD - select this option to use the Head revision of the source. The Head revision is suggested by default- Specified - select this option to use a revision different from the Head revision. Specify the required revision in the text field. If necessary, click the  button and select the revision from the Changes Browser dialog box.

Tip Based on the sources and revisions you specify, the difference between source 2 and source 1 is calculated and applied to the local working copy.

Use ancestry	Select this check box to take the ancestry of the Source1 and Source2 URLs into consideration when comparing revisions. If the check box is not selected, only the contents of the files are compared.
Try merge, but make no changes	Select this check box to enable the <code>--dry-run</code> switch of the <code>svn</code> command. If this check box is not selected, the sources are merged silently.
Depth	Use this drop-down list to specify the range of recursion into subdirectories. The available options are: <ul style="list-style-type: none">- Empty - select this option to involve only the current file.- Files - select this option to involve the files in the folder.- Immediates - select this option to involve direct children of the current file.- Infinity - select this option to enable full recursion.

Integrate to Branch

View | Tool Windows | Version Control | Repository Tab | Context menu of a changelist or file |

Subversion | Integrate to Branch

Use this dialog to specify the options for integrating changes into a branch.

Item	Tooltip	Description
Source branch URL		This read-only field shows the URL address of the source branch.
Target branch URL		This read-only field shows the URL address of the target branch.
Integrate into		From this list, select the path to the local working copy into which the changes will be integrated.
working copy		
Ignore whitespaces		Select this option if whitespaces are not important.
Try merge, but make no changes		Select this option to preview merge results by enabling the <code>--dry-run</code> switch of the <code>svn</code> command. If this option is unchecked, sources are merged silently.
	Add	Click this button to add a working copy to the list.
	Remove	Click this button to remove the selected working copy from the list.

Lock File Dialog (Subversion)

VCS | Subversion | Lock

Context menu of a file | Subversion | Lock

Use this dialog box to lock file when it is necessary to avoid overwriting changes.

ItemDescription

Lock Comment	In this text box, describe the reason for locking the file and some additional comments, if necessary.
Steal existing lock	Select this check box to override the lock previously set on the desired file by someone else.
Do not show this dialog in the future	Select this check box to suppress displaying this dialog box and have files and folders locked silently. To have PyCharm show this dialog box before locking files or folders again: <ol style="list-style-type: none">1. Open the Version Control - Confirmation page of the Settings dialog box.2. In the Display Option dialogs when these commands are invoked area, select the Checkout check box.

Mark Resolved Dialog (Subversion)

VCS | Subversion | Mark Resolved

Project tool window | context menu of a file | Subversion | Mark Resolved

Local Changes tab of the Version Control tool window | context menu of a file | Subversion | Mark Resolved

Project tool window | context menu of a file | Subversion | Mark Resolved

Version Control tool window - Merged with conflicts list | context menu of a file | Subversion | Mark Resolved

Use this dialog box to have PyCharm consider conflicts in a file or directory resolved. This operation is most often required after merging text or property conflicts manually.

ItemDescription

Files and directories	The list shows all the files and directories where merge or updated resulted in conflicts that PyCharm cannot resolve automatically. When you have examined the conflicts resolved them manually or considered irrelevant, you need to tell PyCharm that these files are no longer conflicting. To appoint a file for marking as free from conflicts, select the check box next to it.
Select All	Click this button to have all the items in the list appointed for marking as resolved.
Deselect All	Click this button to clear the list of candidates for marking as resolved.
Mark Resolved	Click this button to have PyCharm treat all the selected items as conflict free. The dialog box closes, whereupon the Local Changes tab of the Version Control tool window shows the affected files as updated and available for submitting to the server.

Select Branch

[Version Control tool window](#) | Repository Tab | Merge Info Pane - Browse

[Version Control tool window](#) | Subversion Working Copies Information Tab | Merge from

[Update Project/Directory dialog box](#) - Browse

The pop-up dialog box opens when you click the Browse button  or press `Shift+Enter` to select the path to the target branch.

Use this dialog box to select the relevant branch or working copy.

ItemDescription

Trunk	Choose this option to set the current trunk as the target branch or working copy.
Branches	Choose this option to select the relevant branch in the Branches list.
Tags	Choose this option to select the relevant tag in the Tags list.
Configure Branches	Choose this option to open the Configure Subversion Branches dialog box and compose a list of branches you work with.

Select Repository Location Dialog (Subversion)

VCS | Subversion | Branch or Tag | Copy From | Repository location

VCS | Subversion | Branch or Tag | Copy To | Any location

Item	Description
Repositories	Use this tree view to explore and manage the available repository locations. Right-click nodes and examine context menus.
Copy as	Specify the name under which the file or folder will be stored in branch.

Set Property Dialog (Subversion)

VCS | Subversion | Set Property

Use this dialog to define SVN-specific properties for the files and folders under SVN version control (ignore list, externals etc.)

ItemDescription

Property name	Enter custom property name in the text field, or use the drop-down list to select one of the pre-defined properties.
Set property value	Click this radio-button to set value for the specified property name in the text area. Properties that accept multiple values, such as an ignore list, can be entered on multiple lines.
Delete property	Click this radio-button to remove selected property from the list.
Update properties recursively	Check this option, if you want to apply the property to every file and directory under the selected directory.

Subversion Options Dialog

File | Settings | Version Control | Subversion

ItemDescription

Use system default Subversion configuration directory

Store Subversion configuration files in the system default directory:
`user_home\Application Data\Subversion`

Subversion configuration directory

Remove the content of the corresponding directory in the Subversion configuration directory.
You may need to clear the authorization information from the configuration file, for example, when your credentials have changed.

Clear authentication cache

Remove the content of the corresponding directory in the Subversion configuration directory.
You may need to clear the authorization information from the configuration file, for example, when your credentials have changed.

Subversion Working Copies Information Tab

View | Tool Windows | Version Control - Subversion Working Copies Information

Use this tab to configure the format of your working copies.

Tip The tab is only available, when the current project sources are entirely or partially under Subversion control.

The tab displays a list of all detected directories under Subversion control supplied with information on the formats used.

ItemDescription

Refresh	Click this button to get the information on all the detected Subversion working copies up-to-date.
Root Path	This read-only field shows the full path to the directory.
URL	This read-only field shows the URL address of the remote directory the selected local copy is mapped to.
Format	This read-only field shows the actual Subversion format used in the selected directory.
Change	Click this link to open the Convert Working Copy Format dialog box, where you can select the desired format option.
Depth	This read-only field shows the range of recursion into subdirectories specified in the Update dialog box.
Working Copy Root	This read-only field is displayed only if the directory in question is the root of a working copy.
Configure Branches	Click this link to open the Configure Subversion Branches dialog box, where you can view and update the list of branches to work with.
Merge from	Click this link to open the Select Branch pop-up dialog box and appoint the source of changes to merge to the current directory.

SVN Repositories

VCS | Browse Subversion Repository

Use this tool window to view, add, and edit the location of SVN repositories.

Toolbar

Item
Tooltip **Description**
and
shortcut

	Add Repository Location	Click this button to configure a new repository location . The New Repository Location dialog box opens, where you can select a repository URL in the drop-down list that contains previously added URL addresses.
	Edit Location Url	Click this button to edit the URL of the selected repository.
	Discard Location	Click this button to remove the selected repository from the list.
	Show/Hide Details	Click this button to display the details for each node under the repository location (changelist number, user name, date and time of the last change).
	Refresh	Click this button to refresh the view.
		Ctrl+F5
	Expand All/Collapse All	Click this button to expand/collapse all nodes.
		Ctrl+NumPad Plus
		F1
	Close	Click this button to close the tool window.

Context Menu

Item
Description

New	Select this menu item to configure a new repository location , or a new remote folder in the selected repository location. The New Repository Location dialog box opens where you can select a repository URL in the drop-down list that contains previously added URL addresses.
Show History	Select this item to open the Version Control tool window with the history of the selected repository location.
Checkout	Select this menu item to check out the contents of the selected node.
Compare with	Select this menu item to compare the selected node with the specified branch.
Browse changes	Select this menu item to view changes that match the specified criteria (author, time range, and revision).
Export	Select this menu item to export the contents of the selected repository or folder to the specified destination. The exported contents are not under version control.
Branch or Tag	Select this menu item to create a branch or tag of the selected folder.
Move or Rename	Select this menu item to change name of the selected folder.
Delete	Select this menu item to delete the selected folder from the repository location.
Copy URL	Select this menu item to put the URL string to the clipboard.
Refresh	Select this menu item to synchronize to the repository.
Edit location Url	Select this menu item to edit the selected repository location.
Discard location	Select this menu item to discard the selected repository location.

Update Project Dialog (Subversion)

VCS | Update Project

Ctrl+T

Context menu of a file or directory | Subversion | Update File/Directory

VCS | Subversion | Update File/Directory

Use this dialog box to update the local working copy of a file, directory, or project with a revision from the repository.

ItemDescription

Update/Switch to specific Url

- Select this check box to synchronize your local working copy with a specific repository. Specify the source repository either in the URL text box through its full Url address or in the Use Branch text box through the branch name.
- Clear this check box to bring the changes from the repository that corresponds to the current working copy.

Use Branch

In this text box, specify the location of the required repository through its branch name. Click the Browse button  to choose the required branch in the [Select Branch](#) dialog box that opens.

Note

1. To use this method of specifying the required repository, you need to [configure a list of branches](#) you work with. If you have not done it yet, click [Configure Branches](#) in the [Select Branch](#) dialog box.
2. The text box is enabled only when the Update/Switch to specific Url check box is selected.

Url

In this text box, specify the location of the required repository through its full URL address. Click the Browse button  to open the [Select Repository Location](#) dialog box.
The text box is enabled only when the Update/Switch to specific Url check box is selected.

Update/Switch to specific revision

Select this check box to synchronize your local working copy with a specific revision different from the HEAD revision. The Update/Switch to specific revision text box becomes enabled.

In this text box, specify the number of the revision to be used. Click the Browse button  to open the [Changes Browser](#) dialog box. By default, PyCharm suggests to update your local working copy the HEAD revision. This option corresponds to the `--non-recursive (-N)` switch of the Subversion `update` command.

Depth

Use this drop-down list to specify the range of recursion into subdirectories. The available options are:

- Working copy - select this option to get files/directories from repository subtrees that have not been checked out yet.
- Empty: select this option to involve only the current file.
- Files: select this option to involve the files in the folder.
- Immediates: select this option to involve direct children of the current file.
- Infinity: select this option to enable full recursion.

Force Update

Select this check box to have local files replaced with the files from the repository even if the local files have modifications and thus abandon the local modifications.

Update administrative information only in changed subtrees

This option only applies to working copies older than SVN 1.7 managed by SVNKit.

During synchronization with the server (update), SVN locks your working copy one subtree after another by creating empty `lock` files in the corresponding administrative `.svn` directories. After that, SVN starts comparing file hashes to detect which local files need to be synchronized.

When this option is selected, SVN first checks if any files from a subtree have been modified on the server, and locks this subtree (i.e. creates a `.svn/lock` file) only if such files are detected. This approach improves performance but may cause concurrency issues, for example, with antiviral software.

Ignore Externals

Select this check box if you do not want PyCharm take into account [externals definitions](#) during update.

Do not show this dialog in the future

Select this check box to have PyCharm perform future updates silently.
To have PyCharm show this dialog box before update again:

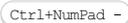
1. Open the [Version Control - Confirmation](#) page of the [Settings](#) dialog box.
2. In the Display Option dialogs when these commands are invoked area, select the Update check box.

Apply Patch Dialog

VCS | Apply Patch

Use the dialog box to restore changes that were preserved in a patch file in the specified directory.

Item Tooltip Description and Shortcut

Item	Tooltip	Description
	Patch file name	In this text box, specify the name of the *.patch file to be applied. Type the fully qualified name or click the Browse button  and locate the desired patch file using the Select Patch File dialog box, that opens.
	Group by Directory	Use this button to toggle between the flat view and the directory tree view. Select the check boxes next to the changes that you want to be applied. 
	Expand All/Collapse All	Use these buttons to expand/collapse all nodes  
	Map base directory	In the dialog that opens , select the directory relative to which file names in the patch file will be interpreted. You can map a base directory to a single file, directory, or to a selection.
	Show Differences	Click this button to open the Differences Viewer for Files that shows the differences between your local working copy, the repository version, and the patch. Use the buttons Compare Previous File  and Compare Next File  to have the files in patch compared in a chain. <div style="background-color: #ffffcc; padding: 5px;">Tip if the patch cannot be applied without conflicts, the lines with conflicts are highlighted with red.</div>
	Strip Directory	Use this button to apply the changes to files located in different directories from the ones specified in the patch. Clicking this button removes one slash in the path to the target file. Click the button as many times as many leading directories you need to strip. The number of removed slashes is indicated in square brackets.
	Restore Directory	Use this button to revert the last strip directory action. Click the button as many times as many previously stripped leading directories you need to restore.
	Reset Directories	Use this button to revert all strip directory actions in the selection.
	Remove Directories	Click this button to have all the leading directories stripped and have the changes applied to the file with the specified name in the base directory.
	Refresh	Click this button to synchronize the tree with the current state of the file system.
Summary		This section displays summary information for the currently selected changelist (the number of modified, new, and deleted files).
Existing Changelist		Select this option to add the patched files to an existing changelist and select the desired changelist from the drop-down list.
New Changelist		Select this option to create a new changelist and add the patched files to it. <ul style="list-style-type: none">- Name: in this text box, type the name of the new changelist. By default, it is the name of the current patch.- Comment: in this text box, type the comment to the new changelist.- Make this changelist active: select this check box to have PyCharm automatically give the active status to the new change list immediately after the changes are restored in it. When this check box is cleared, the current active changelist remains active. See Changelist for details.- Track Context: select this check box to have PyCharm preserve the context of the task associated with the new changelist on its deactivation and restore the context when the changelist becomes active. See Managing Tasks and Context for details.

Commit Changes Dialog

VCS | Commit Changes

View | Tool Windows | Version Control - Local Changes - Context menu of a file or a changelist - Commit Changes

Use this dialog box to commit (check in) changes from the selected changelist to the repository and, optionally, to create a patch file.

This dialog box consists of several areas:

- [Modified files pane](#)
 - [Toolbar](#)
- [Commit Message pane](#)
 - [Toolbar](#)
- [VCS-specific controls](#)
- [Before Submit / Before Commit section](#)
- [After Submit / After Commit section](#)
- [Diff pane](#)
 - [Toolbar](#)
- [Submit / Commit button](#)

The options available in this dialog depend on the version control system you are using.

Modified files pane

This section contains a list of files that have been modified since the last commit. Deselect the check-boxes next to the files that you want to exclude from current commit.

Toolbar

Item	Tooltip and Shortcut	Description	Available in
	Show Differences Ctrl+D	Click this button to open the Differences dialog box that highlights the differences between your local working copy of the selected file and its repository version.	All VCSs
	Refresh Changes Ctrl+F5	Click this button to reload the Changed files tree view so it is up-to-date.	All VCSs
	Show Unversioned Files	Click this button if you want to see newly added files that have not been added to version control yet under the Unversioned Files node.	All VCSs
	Add to VCS Ctrl+Alt+A	Click this button to move the files selected under the Unversioned Files node to the active changelist, so that they are added to your version control system during the commit.	All VCSs
	Move to Another Changelist F6	Click this button to add the selected file(s) to another changelist. The Move to Another Changelist dialog box opens where you can select an existing changelist or create a new one.	All VCSs
	Delete	Click this button to delete the selected file.	
	Ignore	Click this button to leave the selected files unversioned.	All VCSs
	Revert	Click this button to revert all changes made to the local working copy of the selected files.	All VCSs
	Jump to source F4	Click this button to open the source code of the selected file in the editor.	All VCSs
	Revert Unchanged Files	Click this button to revert the files that have not been modified locally.	Subversion Perforce
	Group by Directory Ctrl+P	Click this button to toggle between the flat view and the directory tree view.	All VCSs
 	Expand or collapse all nodes Ctrl+NumPad Plus Ctrl+NumPad -	Click these buttons to expand or collapse all nodes in the directory tree. These buttons are not available in flat view.	All VCSs
Change list	N/A	From this drop-down list, select the changelist that contains the modified files to be checked in or included in the patch. The active changelist is selected by default.	All VCSs

The summary under the modified files pane shows statistics on the currently selected changelist, such as the number of modified, new, deleted and

unversioned files. This area also shows how many files of each type are shown, and how many of them will be committed.

Commit Message pane

In this area, enter a comment to the current commit. You cannot commit your changes until you enter some description in the Commit Message field.

This comment will also be used as the name of the patch file, if you decide to create a patch.

Toolbar

Icon and Tooltip	Shortcut	Description
------------------	----------	-------------

 Commit Message History	Ctrl+M	Click this icon to invoke the Commit Message History dialog that contains a list of your twenty-five last commits and the corresponding commit messages.
--	--------	--

VCS-specific controls

The controls in this section are located in the top-right part of the dialog, and contain the options that are specific for the version control system you are using.

Item	Description	Available for
Author	Use this drop-down list to select the author of the changes that you are going to commit. This may be useful when you are committing changes made by another person.	Git
Amend commit	Select this check box to replace the previous commit with the current changes (see Git Basics: Undoing Things for details).	Git, Mercurial
Sign-off commit	Select this option if you want to sign off your commit, i.e. to certify that the changes you are about to check in have been made by you, or that you take the responsibility for the code in question. When this option is enabled, the following line is automatically added at the end of the commit message: Signed off by: <username>	Git
Keep files locked	Select this check box to keep the changed files locked after they are checked in.	Subversion
Jobs	This area is available only if you select the Enable Perforce Jobs Support check box on the Perforce page of the Settings dialog box. Use the controls in this area to search for Perforce jobs , link jobs to the selected changelist, and detach the currently linked jobs. <ul style="list-style-type: none">– Unlink selected jobs: click this button to detach the selected job from the changelist.–  Edit associated jobs: click this button to open the Edit Jobs Linked to Changelist dialog where you can search for available jobs, view their details, and link jobs to the selected changelist.–  Find and link job matching the pattern: click this button to start quick search for the job that matches the pattern specified in the text box and attach the job to the changelist. In the text box, specify the exact name of the job or a search pattern according to the Perforce jobs syntax rules . Tip If only one job matching the pattern is found, it is attached to the changelist automatically. Otherwise, to select a job among several available jobs, click the  button and find the desired job using the Edit Jobs Linked to Changelist dialog box.	Perforce

The list box in the bottom of the area displays the jobs that are currently attached to the selected changelist.

Before Submit / Before Commit section

Use the controls in this area to define which additional actions you want PyCharm to perform before committing the selected files.

These controls are available for the following version control systems:

- Git
- CVS
- Subversion
- Perforce

Item	Description
------	-------------

Reformat code	Select this check box to perform code formatting according to the Project Code Style settings .
Rearrange code	Select this check box to rearrange your code according to the .
Optimize imports	Select this check box to remove redundant import statements .
Perform code analysis	Select this check box to run code inspection on the files you are about to commit.
Check TODO (<filter name>)	Select this check box to review the TODO items matching the specified filter. Click the Configure link to choose an existing TODO filter , or open the TODO settings page and define a new filter to be applied.
Cleanup	Select this check box if you want to automatically apply the current inspection profile to the files you are going to commit.
Update copyright	Select this check box to add or update a copyright notice according to the selected copyright profile - scope combination .

Revert unchanged files Select this check box to revert the files that have not been modified. This option is only available for Perforce.

After Submit / After Commit section

Use the controls in this area to define which additional actions you want PyCharm to perform after committing the selected files.

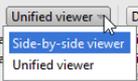
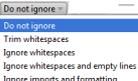
Item	Description	Available for
Run tool	From this drop-down list, select the external tool that you want PyCharm to launch after the selected changes have been committed. You can select a tool from the list, or click the Browse button  and configure an external tool in the External Tools dialog box that opens.	All VCSs
Upload files to	From this drop-down list, select the server access configuration to use for uploading the committed files to a local or remote host, a mounted disk, or a directory. To suppress uploading, choose None. To add a server configuration to the list, click  and fill in the required fields in the Add Server dialog that opens.	All VCSs
Always use selected server	Select this check box to always upload files to the selected server access configuration. The drop-down list and the check box are only available if the Remote Hosts Access plugin is enabled.	All VCSs
Tag committed files	Select this check box to assign a tag to the committed files and type the name of the tag. To replace a previously assigned tag with a new one, select the Override existing tags option.	CVS
Auto-update after commit	Select this check box to automatically update your project after the commit. Enabling this option will help prevent your working copy against the mixed-revision state . The mixed-revision state of a working copy may affect the Move and Rename refactoring applied to folders, in which case items in revisions different from the moved subtree root will be tracked separately, which can be confusing. When the Auto-update after commit option is enabled: <ul style="list-style-type: none">– Merge will fail with an error if the merge target is a mixed-revision working copy.– Your own changes will never cause a 409 conflict.	Subversion

Diff pane

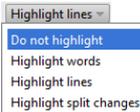
The Diff pane is hidden by default. To unfold it, click the arrow button  next to the pane title.

In this pane you can explore the differences between the base repository version of the selected file, and the version you are about to commit.

Toolbar

Item	Tooltip and Shortcut	Description
	Previous Difference / Next Difference Shift+F7 F7	Use these buttons to jump to the next/previous difference. When the last/first difference is hit, PyCharm suggests to click the arrow buttons  /  once more and compare other files, depending on the Go to the next file after reaching last change option in the Differences Viewer settings . This behavior is supported only when the Differences Viewer is invoked from the Version Control tool window.
	Compare Previous/Next File Alt+Left Alt+Right	Click these buttons to compare the local copy of the previous/next file with its update from the server. Tip These buttons are only available when there is more than one file in the selected changelist.
	Jump to Source F4	Click this button to open the selected file in the active pane in the editor. The caret will be placed in the same position as in the Differences Viewer.
Viewer type		Use this drop-down list to choose the desired viewer type. The side-by-side viewer has two panels; the unified viewer has one panel only. Both types of viewers enable you to <ul style="list-style-type: none">– Edit code. Note that one can change text only in the right-hand part of the default viewer, or, in case of the unified viewer, in the lower ("after") line, i.e. in your local version of the file.– Perform the Apply/Append/Revert actions.
Whitespace		Use this drop-down list to define how the differences viewer should treat white spaces in the text. <ul style="list-style-type: none">– Do not ignore: white spaces are important, and all differences are highlighted. This option is selected by default.– Trim whitespaces: (" \t", " ") , if they appear in the end and in the beginning of a line.<ul style="list-style-type: none">– If two lines differ in trailing whitespaces only, these lines are considered equal.– If two lines are different, such trailing whitespaces are not highlighted in the By word mode.– Ignore whitespaces: white spaces are not important, regardless of their location in the source code.– Ignore whitespaces and empty lines: the following entities are ignored:<ul style="list-style-type: none">– all whitespaces (as in the 'ignore whitespaces' option)– all added or removed lines consisting of whitespaces only– all changes consisting of splitting or joining lines without changes to non-whitespace parts. For example, changing <code>a b c</code> to <code>a \n b c</code> is not highlighted in this mode.

Highlighting mode



Select the way differences granularity is highlighted.

The available options are:

- Highlight words: the modified words are highlighted
- Highlight lines: the modified lines are highlighted
- Highlight split changes: if this option is selected, big changes are split into smaller 'atomic' changes.

For example, `A \n B` vs. `A X \n B X` will be treated as two changes instead of one.

- Do not highlight: if this option is selected, the differences are not highlighted at all. This option is intended for significantly modified files, where highlighting only introduces additional difficulties.

	Collapse unchanged fragments	Click this button to collapse all unchanged fragments in both files. The amount of non-collapsible unchanged lines is configurable in the Diff & Merge settings page.
	Synchronize scrolling	Click this button to scroll both differences panes simultaneously. If this button is released, each pane can be scrolled independently.
	Disable editing	Click this button to enable editing of the local copy of the selected file, which is disabled by default. When editing is enabled, you can make last-minute changes to the modified file before committing it.
	Editor settings	Click this button to open a drop-down list of available options. Select or clear these options to show or hide line numbers, indentation guides, white spaces, and soft wraps.
	Show diff in external tool	Click this button to invoke an external differences viewer, specified in the External Diff Tools settings page. This button only appears on the toolbar when the Use external diff tool option is enabled in the External Diff Tools settings page.
	Help	Click this button to show the corresponding help page.



Note that the options listed above are available for text files only. PyCharm cannot compare binary files, so most commands will be unavailable for them.

Submit / Commit button

Click this button to commit the selected files, or hover your mouse over this button to display one of the following available commit options:

- Commit and Push: select this option to push the changes to the remote repository immediately after the commit. This option is available if you are using [Git](#) or [Mercurial](#) as a version control system.
- Create MQ Patch: select this option to create an MQ patch based on your changes. This option is only available if you are using Mercurial as a version control system.
- Create Patch: select this option if you want PyCharm to generate a patch based on the changes you are about to commit. In the Create Patch dialog that opens, type the name of the patch file and specify whether you need a reverse patch.
- Remote Run: select this option to [run your personal build](#). This option is only available when you are logged in to [TeamCity](#). Refer to [TeamCity plugin documentation](#) for details.

Configure Ignored Files Dialog

Version Control tool window | Local Changes - 

Use this dialog to configure a list of files and directories that you do not want to put under version control. These can be file names associated with VCS administration, backup files, and any other artifacts that you want to remain unversioned. You can also specify [patterns](#) of files you want to ignore.

Tip You can only ignore unversioned files, i.e. files that have not yet been put under version control.

Item	Keyboard shortcut	Description
		Use this icon or shortcut to add an item to the list. The Ignore Unversioned Files dialog box opens where you can type an exact path to a file or directory to be ignored or specify a pattern that defines the names of files and directories to be ignored.
		Use this icon or shortcut to edit the selected path or pattern in the Ignore Unversioned Files dialog box.
		Use this icon or shortcut to remove the selected path or pattern from the list.

Create Patch Dialog

VCS | Create Patch

View | Tool Windows | Version Control - Local Changes - Context menu of a file or changelist - Create Patch

Use this dialog box to generate a patch file for the specified changelist or files.

Use this dialog box to create a patch from the selected changelist or files.

This dialog box consists of several areas:

- [Modified files pane](#)
 - [Toolbar](#)
- [Commit Message pane](#)
 - [Toolbar](#)
- [Details pane](#)
 - [Toolbar](#)

Modified files pane

This section contains a list of files that were modified since the last commit. All files in this list are selected by default. Deselect the check-boxes next to the files that you want to exclude from the patch.

Toolbar

Item	Tooltip and Shortcut	Description	Available in
		Show Differences Ctrl+D	Click this button to open the Differences dialog box that highlights the differences between your local working copy of the selected file and its repository version. All VCSs
		Refresh Changes Ctrl+F5	Click this button to reload the Changed files tree view so it is up-to-date. All VCSs
		Show Unversioned Files	Click this button if you want to see newly added files that have not been added to version control yet under the Unversioned Files node. All VCSs
		Add to VCS Ctrl+Alt+A	Click this button to move the files selected under the Unversioned Files node to the active changelist, so that they are added to your version control system during the commit. All VCSs
		Move to Another Changelist F6	Click this button to add the selected file(s) to another changelist. The Move to Another Changelist dialog box opens where you can select an existing changelist or create a new one. All VCSs
		Delete	Click this button to delete the selected file.
		Ignore	Click this button to leave the selected files unversioned. All VCSs
		Revert	Click this button to revert all changes made to the local working copy of the selected files. All VCSs
		Jump to source F4	Click this button to open the source code of the selected file in the editor. All VCSs
		Revert Unchanged Files	Click this button to revert the files that have not been modified locally. Subversion Perforce
		Group by Directory Ctrl+P	Click this button to toggle between the flat view and the directory tree view. All VCSs
 		Expand or collapse all nodes Ctrl+NumPad Plus Ctrl+NumPad -	Click these buttons to expand or collapse all nodes in the directory tree. These buttons are not available in flat view. All VCSs
Change list		NA	From this drop-down list, select the changelist that contains the modified files to be checked in or included in the patch. The active changelist is selected by default. All VCSs

The summary under the modified files pane shows statistics on the currently selected changelist, such as the number of modified, new and deleted files. This area also shows how many files of each type are shown, and how many of them will be included in the patch.

Commit Message pane

The comment you enter in this area will be used as the name of the patch file.

Toolbar

Icon and
Tooltip

ShortcutDescription

 Commit Message History  Click this icon to invoke the Commit Message History dialog that contains a list of your twenty-five last commits and the corresponding commit messages.

Details pane

The Details pane is hidden by default. To unfold it, click the arrow button  next to the pane title.

In this pane you can explore the differences between the base repository version of the selected file, and the version you want to include in the patch.

Toolbar

ItemTooltip
and
Shortcut

Description

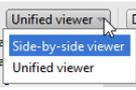
 Previous Difference / Next Difference   Use these buttons to jump to the next/previous difference. When the last/first difference is hit, PyCharm suggests to click the arrow buttons  /  once more and compare other files, depending on the [Go to the next file after reaching last change](#) option in the [Differences Viewer settings](#).

This behavior is supported only when the Differences Viewer is invoked from the [Version Control](#) tool window.

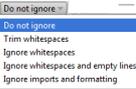
 Compare Previous/Next File   Click these buttons to compare the local copy of the previous/next file with its update from the server.

Tip These buttons are only available when there is more than one file in the selected changelist.

 Jump to Source  Click this button to open the selected file in the active pane in the editor. The caret will be placed in the same position as in the Differences Viewer.

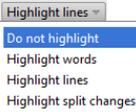
Viewer type  Use this drop-down list to choose the desired viewer type. The side-by-side viewer has two panels; the unified viewer has one panel only. Both types of viewers enable you to

- Edit code. Note that one can change text *only* in the right-hand part of the default viewer, or, in case of the unified viewer, in the lower ("after") line, i.e. in your local version of the file.
- Perform the Apply/Append/Revert actions.

Whitespace  Use this drop-down list to define how the differences viewer should treat white spaces in the text.

- Do not ignore: white spaces are important, and all differences are highlighted. This option is selected by default.
- Trim whitespaces: (" \t", " ") , if they appear in the end and in the beginning of a line.
 - If two lines differ in trailing whitespaces only, these lines are considered equal.
 - If two lines are different, such trailing whitespaces are not highlighted in the [By word](#) mode.
- Ignore whitespaces: white spaces are not important, regardless of their location in the source code.
- Ignore whitespaces and empty lines: the following entities are ignored:
 - all whitespaces (as in the 'Ignore whitespaces' option)
 - all added or removed lines consisting of whitespaces only
 - all changes consisting of splitting or joining lines without changes to non-whitespace parts.

For example, changing `a b c` to `a \n b c` is not highlighted in this mode.

Highlighting mode  Select the way differences granularity is highlighted.

The available options are:

- Highlight words: the modified words are highlighted
- Highlight lines: the modified lines are highlighted
- Highlight split changes: if this option is selected, big changes are split into smaller 'atomic' changes.

For example, `A \n B` vs. `A X \n B X` will be treated as two changes instead of one.

- Do not highlight: if this option is selected, the differences are not highlighted at all. This option is intended for significantly modified files, where highlighting only introduces additional difficulties.

 Collapse unchanged fragments Click this button to collapse all unchanged fragments in both files. The amount of non-collapsible unchanged lines is configurable in the [Diff & Merge](#) settings page.

 Synchronizè scrolling Click this button to scroll both differences panes simultaneously. If this button is released, each pane can be scrolled independently.

 Disable editing Click this button to enable editing of the local copy of the selected file, which is disabled by default. When editing is enabled, you can make last-minute changes to the modified file before committing it.

 Editor settings Click this button to open a drop-down list of available options. Select or clear these options to show or hide line numbers, indentation guides, white spaces, and soft wraps.

 Show diff in external tool Click this button to invoke an external differences viewer, specified in the [External Diff Tools](#) settings page.

This button only appears on the toolbar when the Use external diff tool option is enabled in the [External Diff Tools](#) settings page.



Help

Click this button to show the corresponding help page.



Note that the options listed above are available for text files only. PyCharm cannot compare binary files, so most commands will be unavailable for them.

After you've selected the files you want to commit, click the Create Patch button and specify the patch file options in the dialog that opens.

Enable Version Control Integration Dialog

VCS | Enable Version Control Integration

In this dialog box, choose one of the registered Version Control Systems to use in your project and assign it to the Project Root.

The dialog box and the menu item are available only if the project does not use any Version Control System.

Item Description

Select a version control system to associate with the project root

From this drop-down list, select one of the supported version control systems that you want to associate with the project root.

File Status Highlights

In PyCharm each file has its own status marked with a specific color. The file status denotes correspondence of the actual file content with the one marked as 'current'.

In the editor, each line in a file is checked whether it corresponds to the state at the 'current' point and marked with a specific color at the left gutter area.

You can customize the default colors:

- For files - in the File Status page of the [Colors and Fonts](#) settings.
- For the lines in the editor - in the VCS page of the [Colors and Fonts](#) settings.

In this section:

- [File Status in Views](#).
- [Line Status in the Editor](#).

File Status in Views

Color	File Status	Description
Black	Up to date	File is unchanged. 
Gray	Deleted	File is scheduled for deletion from the repository. 
Blue	Modified	File has changed since the last synchronization. 
Green	Added	File is scheduled for addition to the repository. 
Violet	Merged	File is merged by your VCS as a result of an update. 
Brown	Unversioned	File exists locally, but is not in the repository, and is not scheduled for adding. 
Olive	Ignored	File will be ignored in any VCS operation. 
Light brown	Hijacked	File is modified without checkout . This status is valid for the files under Perforce, ClearCase and VSS. 
Red	Merged with conflicts	During the last update, file was merged with conflicts. 
Lilac	Externally deleted	File is deleted locally, but was not scheduled for deletion, and still exists in the CVS repository. 
Dark cyan	Switched	The file is taken from a different branch than the whole project. This status is valid for CVS and SVN. 

Line Status in the Editor

Color	File Status	Description
	Modified	Denotes the lines modified since the last synchronization.
	Added	Denotes the lines added since the last synchronization.
	Deleted	Denotes the lines removed since the last synchronization.

New Changelist Dialog

Version Control tool window - +

Alt+Insert

Use this dialog box to create a new changelist.

ItemDescription

Name	Type the name of the new changelist.
Comment	Type optional comment. When the new changelist will be submitted to the repository, this comment will appear in the Comment text area of the Commit Changes dialog box.
Make this changelist active	Select this check box to have PyCharm automatically give the active status to the new change list immediately after the changes are restored in it. When this check box is cleared, the current active changelist remains active. See Changelist for details.

Patch File Settings Dialog

VCS | Create Patch - Create Patch

View | Tool Windows | Version Control - Local Changes - Context menu of a file or changelist - Create Patch - Create Patch

Use this dialog to configure the patch file settings.

ItemDescription

Patch file	Specify the name of the patch file. By default, the text in the Commit Message section of the Create Patch dialog is used as the file name. If the Commit Message section is empty, the default name is <code>unnamed.patch</code> .
Base path	Specify the path relative to which paths inside the patch file will be written. Normally, this is your project directory, but you may want to use a relative path, for example, if the modified files are stored inside your VCS repository.
Reverse patch	Select this option if you want to create a patch that reverts the changes you have made.
Encoding	Select the encoding for the patch file from the drop-down list.

Push Dialog (Mercurial, Git)

VCS | Git | Push

VCS | Mercurial | Push

This dialog is available for the following version control systems:

- Git
- Mercurial

The dialog consists of two panes (the Repositories pane and the Commit details pane) and the Push controls area:

- [Repositories pane](#)
- [Commit details pane](#)
- [Push controls](#)

Repositories pane

The left pane shows a list of Git and/or Mercurial repositories (as well as which local branch/active bookmark will be pushed to which remote branch), and a list of commits performed in each repository.

- Hover the mouse over a commit: a tooltip is displayed showing the commit number, date and time, author, and the commit message. If the author of a commit is different from the current user, this commit is marked with an asterisk.
- Select the checkbox next to each repository to which you want to push.
If you have a multirooted project where repositories are not controlled synchronously, only the current repository is selected by default (or multiple repositories selected in the Project View). For details on how to enable/disable synchronous repositories control, refer to the following sources:
 - for **Git**: [Version Control Settings: Git](#)
 - for **Mercurial**: [Version Control Settings: Mercurial](#)
- To modify the target branch where you want to push (it is highlighted in blue), click it. The label turns into a text field where you can specify the target branch. You can also switch into the editing mode by selecting the branch that you want to modify and pressing `Enter`.
- You can also edit the remote repository (if there are multiple ones) in the same way as the remote branch. Note that if no remotes have been specified, the Define remote link will appear instead of a remote name. Click it to add a remote.
- If there are no remotes in the repository, the Define remote link appears. Click this link and specify the remote name and URL in the dialog that opens.

Commit details pane

The right pane shows which files are included in the selected commit. If you select multiple branches in the left pane, all corresponding commits will be shown.

The toolbar in this area provides the following options:

ItemTooltip Description and shortcut

	Show Diff <code>Ctrl+D</code>	Click this button to open the Differences Viewer for Files dialog that shows the differences between the committed version of the selected file and its previous version.
	Edit Source <code>F4</code>	Click this button to open the selected file in the editor.
	Group by Directory <code>Ctrl+P</code>	Click this button to toggle between the flat view and the directory view.
	Collapse All / Expand All <code>Ctrl+NumPad - /</code> <code>Ctrl+NumPad Plus</code>	Click these buttons to fold/unfold all nodes in the directory tree. These buttons are unavailable if the flat view is selected.

Push controls

The controls in this area allow you to select the following push options:

ItemDescription

Push Tags	This option is only available if you are using Git. By default, when you perform the <code>push</code> operation, tags are not sent to the remote repositories. Select this option if you want to push tags with your commits. <ul style="list-style-type: none">- Select All if you want to push all tags, including the tags that do not belong to the selected branches you are about to push (equivalent to <code>push --tags</code>).- Select Current Branch if you want to push only the tags that belong to the selected branches you are about to push (equivalent to <code>push --follow-tags</code> available since Git 1.8.3).
Export Active Bookmarks	This option is only available if you are using Mercurial. By default, when you perform the <code>push</code> operation, bookmarks are not sent to the remote repositories. Select this option if you want to push active bookmarks with your commits.

Push

Click this button and select which operation you want to perform from the drop-down menu: `push` or `push --force`.

For instructions on how to use the `push --force` command and where it may be useful, refer to:

- For Git: [Pushing Changes to the Upstream \(Git Push\)](#)
- For Mercurial: [Pushing Changes to the Upstream \(Push\)](#)

Note For Git, these choice options are only available if the Allow force push option is enabled (see [Version Control Settings: Git](#)), otherwise, you can only perform the `push` operation.

Select Target Changelist Dialog

VCS | Show Changes View - Repository / Incoming - Context menu of a file or change list - Revert Changes

View | Tool Windows | Changes - Repository / Incoming - Context menu of a file or change list - Revert Changes

Use this dialog box to roll back changes from a certain change list.

ItemDescription

Existing Changelist	Choose this option restore the shelved changes in one of the existing changelists. Choose the target changelist from the drop-down list.
New Changelist	<p>Choose this option to create a new changelist and restore the changes from the shelf in it.</p> <ul style="list-style-type: none">- Name: in this text box, type the name of the changelist to be created.- Comment: in this text box, type an optional description of the new changelist.- Make this changelist active: select this check box to have PyCharm automatically give the active status to the new change list immediately after the changes are restored in it. When this check box is cleared, the current active changelist remains active. See Changelist for details.- Track context: select this check box to have PyCharm preserve the context of the task associated with the new changelist on its deactivation and restore the context when the changelist becomes active. See Managing Tasks and Context for details.
Remove successfully applied files from the shelf	<ul style="list-style-type: none">- Clear this check box to have PyCharm still display already unshelved changes in the Shelf tab so that you can apply them once more if necessary.- When this check box is selected, the changes are not displayed in the Shelf tab after they are unshelved.

Shelve Changes Dialog

VCS | Shelve Changes

Use this dialog box to [shelve](#) the selected files or changelists.

This dialog consists of several areas:

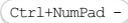
- [Modified files pane](#)
 - [Toolbar](#)
- [Commit Message pane](#)
 - [Toolbar](#)
- [Before Submit / Before Commit section](#)
- [After Submit / After Commit section](#)
- [Diff pane](#)
 - [Toolbar](#)

Modified files pane

This section contains a list of files that have been modified since the last commit. All files in the list are selected by default. Deselect the files that you do not want to shelve.

Toolbar

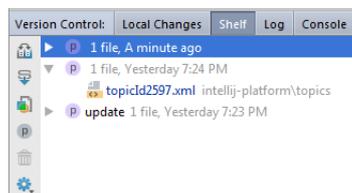
Icon**Tooltip** **Description**
and
Shortcut

	Show Differences	Click this button to open the Differences dialog box that highlights the differences between your local working copy of the selected file and its repository version.	All VCSs
			
	Refresh Changes	Click this button to reload the Changed files tree view so it is up-to-date.	All VCSs
			
	Move to Another Changelist	Click this button to add the selected file(s) to another changelist. The Move to Another Changelist dialog box opens where you can select an existing changelist or create a new one.	All VCSs
			
	Revert	Click this button to revert all changes made to the local working copy of the selected files.	All VCSs
	Jump to source	Click this button to open the source code of the selected file in the editor.	All VCSs
			
	Group by Directory	Click this button to toggle between the flat view and the directory tree view.	All VCSs
			
	Expand or collapse all nodes	Click these buttons to expand or collapse all nodes in the directory tree. These buttons are not available in flat view.	All VCSs
			
			
	Revert Unchanged Files	Click this button to have unchanged files reverted.	Perforce

The summary under the modified files pane shows statistics on the currently selected changelist, such as the number of modified, new and deleted files. This area also shows how many files of each type are shown, and how many of them will be shelved.

Commit Message pane

In this area, enter a string that will be used as the shelf name. When you unshelve your changes, a new changelist with the same name will be created in the [Local Changes tab](#). If you leave this field empty, the shelf name will be generated using the following pattern: <number of files in the shelf>, <date and time when the shelf was created>:



Toolbar

Icon and **Shortcut****Description**
Tooltip

	Commit Message History	Click this icon to invoke the Commit Message History dialog that contains a list of your twenty-five
---	------------------------	--

Before Submit / Before Commit section

Use the controls in this area to define which additional actions you want PyCharm to perform before putting the selected files to a shelf.

These controls are available for the following version control systems:

- Git
- CVS
- Subversion
- Perforce

ItemDescription

Reformat code	Select this check box to perform code formatting according to the Project Code Style settings .
Rearrange code	Select this check box to rearrange your code according to the .
Optimize imports	Select this check box to remove redundant import statements .
Perform code analysis	Select this check box to run code inspection on the files you are about to commit.
Check TODO (<filter name>)	Select this check box to review the TODO items matching the specified filter. Click the Configure link to choose an existing TODO filter , or open the TODO settings page and define a new filter to be applied.
Cleanup	Select this check box if you want to automatically apply the current inspection profile to the files you are going to commit.
Update copyright	Select this check box to add or update a copyright notice according to the selected copyright profile - scope combination .
Revert unchanged files	Select this check box to revert the files that have not been modified. This option is only available for Perforce.

After Submit / After Commit section

Use the controls in this area to define which additional actions you want PyCharm to perform after putting the selected files to a shelf.

ItemDescription

Upload files to selected server	From this drop-down list, select the server access configuration to use for uploading the selected files to a local or remote host, a mounted disk, or a directory. To suppress uploading, choose None. To add a server configuration to the list, click  and fill in the required fields in the Add Server dialog that opens.
Always use selected server	Select this check box to always upload files to the selected server access configuration. The drop-down list and the check box are only available if the Remote Hosts Access plugin is enabled.

Diff pane

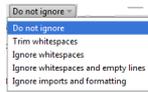
The Diff pane is hidden by default. To unfold it, click the arrow button  next to the pane title.

In this pane you can explore the differences between the base repository version of the selected file, and the version you are about to shelve.

Toolbar

ItemTooltip and Description and Shortcut

	<p>Previous Difference / Next Difference</p> <p> </p>	<p>Use these buttons to jump to the next/previous difference.</p> <p>When the last/first difference is hit, PyCharm suggests to click the arrow buttons  /  once more and compare other files, depending on the Go to the next file after reaching last change option in the Differences Viewer settings.</p> <p>This behavior is supported only when the Differences Viewer is invoked from the Version Control tool window.</p>
	<p>Compare Previous/Next File</p> <p> </p>	<p>Click these buttons to compare the local copy of the previous/next file with its update from the server.</p> <p>Tip These buttons are only available when there is more than one file in the selected changelist.</p>
	<p>Jump to Source</p> <p></p>	<p>Click this button to open the selected file in the active pane in the editor. The caret will be placed in the same position as in the Differences Viewer.</p>
Viewer type		<p>Use this drop-down list to choose the desired viewer type. The side-by-side viewer has two panels; the unified viewer has one panel only.</p> <p>Both types of viewers enable you to</p> <ul style="list-style-type: none"> - Edit code. Note that one can change text <i>only</i> in the right-hand part of the default viewer, or, in case of the unified viewer, in the lower ("after") line, i.e. in your local version of the file. - Perform the Apply/Append/Revert actions.
Whitespace		<p>Use this drop-down list to define how the differences viewer should treat white spaces in the text.</p> <ul style="list-style-type: none"> - Do not ignore: white spaces are important, and all differences are highlighted. This option is

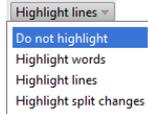


selected by default.

- Trim whitespaces: (" \t", " ") , if they appear in the end and in the beginning of a line.
 - If two lines differ in trailing whitespaces only, these lines are considered equal.
 - If two lines are different, such trailing whitespaces are not highlighted in the **By word** mode.
- Ignore whitespaces: white spaces are not important, regardless of their location in the source code.
- Ignore whitespaces and empty lines: the following entities are ignored:
 - all whitespaces (as in the 'Ignore whitespaces' option)
 - all added or removed lines consisting of whitespaces only
 - all changes consisting of splitting or joining lines without changes to non-whitespace parts.

For example, changing `a b c` to `a \n b c` is not highlighted in this mode.

Highlighting mode



Select the way differences granularity is highlighted.

The available options are:

- Highlight words: the modified words are highlighted
- Highlight lines: the modified lines are highlighted
- Highlight split changes: if this option is selected, big changes are split into smaller 'atomic' changes.

For example, `A \n B` vs. `A X \n B X` will be treated as two changes instead of one.

- Do not highlight: if this option is selected, the differences are not highlighted at all. This option is intended for significantly modified files, where highlighting only introduces additional difficulties.

	Collapse unchanged fragments	Click this button to collapse all unchanged fragments in both files. The amount of non-collapsible unchanged lines is configurable in the Diff & Merge settings page.
	Synchronize scrolling	Click this button to scroll both differences panes simultaneously. If this button is released, each pane can be scrolled independently.
	Disable editing	Click this button to enable editing of the local copy of the selected file, which is disabled by default. When editing is enabled, you can make last-minute changes to the modified file before committing it.
	Editor settings	Click this button to open a drop-down list of available options. Select or clear these options to show or hide line numbers, indentation guides, white spaces, and soft wraps.
	Show diff in external tool	Click this button to invoke an external differences viewer, specified in the External Diff Tools settings page. This button only appears on the toolbar when the Use external diff tool option is enabled in the External Diff Tools settings page.
	Help	Click this button to show the corresponding help page.



Note that the options listed above are available for text files only. PyCharm cannot compare binary files, so most commands will be unavailable for them.

Show History for File / Selection Dialog

VCS | Local History | Show History or Show History for Selection

Use this dialog to explore changes to a file, or selection. There are two views in this dialog:

- [History view](#) in the left-hand part
- [Differences view](#) in the right-hand part

Tip The same dialog boxes are available on the context menu of a file or selected text in the editor.

History view

This view shows the list of revisions (states) of a file, with the date and time when the revision was stored. Some of the revisions are supplied with tags and labels.

Revisions are tagged automatically, for example, on opening a project, committing changes, or performing test. You can also [set your own labels](#).

ItemDescription

	Click this button to revert the selected action.
	Click this button to create a patch based on the selected local version.
	Click this button to open the corresponding help topic.

Tip The same actions are available on the context menu of each revision.

Differences view

The Differences view is a powerful editor that supports [basic search and replace](#), [undo/redo actions](#), and [code completion](#).

If a revision is selected in the [History view](#), the left-hand pane of the Differences view shows this read-only revision, with the differences against the current revision which is displayed in the right-hand pane. The current revision can be edited.

Item ShortcutDescription

		Click this button to copy the current line or the selected fragment to the clipboard.
		Use these buttons to move to the next or previous difference.
Ignore whitespace		Use this drop-down list to define how the differences viewer should treat white spaces in the text. <ul style="list-style-type: none">- Do not ignore - when this option is selected, white spaces are considered unimportant and the differences are highlighted.- Leading and Trailing - select this option to have differences in the end and in the beginning of a line ignored.- All - when this option is selected, white spaces are considered unimportant regardless of their location in the source code.
		Use these buttons to apply differences.
Legend		This area shows summary information about the encountered differences: the number of differences found and the color map. The color map for the Differences viewer is configured on the Colors and Fonts page .

Show History for Folder Dialog

VCS | Local History | Show History

Use this dialog to explore changes to a folder. There are two views in this dialog:

- [History view](#) in the left-hand part
- [Changes view](#) in the right-hand part

Tip The same dialog box is available on the context menu of a folder.

History view

The History view shows a list of folder revisions (states), each one being supplied with a time stamp, revision number, and an indication of the action that resulted in that state.

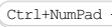
ItemDescription

	Click this button to revert the selected action.
	Click this button to create a patch based on the selected local version.
	Click this button to open the corresponding help topic.

Changes view

The Changes view shows the differences between the current state and the one selected in the [History view](#). The differences are shown as a tree of changed (new, modified and deleted) files and subfolders.

ItemShortcutDescription

		Click this button to show the differences between the current local version and the one selected in the History view. Alternatively, double-click the current local version in the Changes view.
		With a file selected in the Changes view, click this button to roll back the selected action.
		Click this button to show changed files as a tree view of folders. If this button is not pressed, the files are shown as a flat list.
	 or 	Click these buttons to have all nodes expanded or collapsed. These buttons are only available when the changed files are shown as a tree view.
		Click this button to select all the files in the list or a tree view.
		In this area, type the search string. Note also that speed search is available in the Changes pane.
		Click this button to clear the search area.

Tip [Speed search](#) is available in the Version Control view.

Revert Changes Dialog

VCS | Show Changes View - Local - Context menu of a file or change list - Revert Changes

View | Tool Windows | Changes - Local - Context menu of a file or change list - Revert Changes

Use this dialog box to roll back changes that have not yet been committed to the repository.

Toolbar

Item Tooltip and Shortcut	Description
	<p>Show Differences</p> <p>Click this button to open the Differences dialog box that points at the inconsistencies between your local working copy of the selected file and the file in the repository.</p> <p>Ctrl+D</p>
	<p>Move to Another Changelist</p> <p>Click this button to add the selected file(s) to another changelist. The Choose Changelist dialog box opens where you can select an existing changelist or create a new one.</p> <p>F6</p>
	<p>Group by Directory</p> <p>Click this button to toggle between the flat view and the directory tree view.</p> <p>Ctrl+P</p>
 	<p>Expand or collapse all nodes</p> <p>Click these buttons to expand or collapse all nodes in the directory tree. These buttons are not available in flat view.</p> <p>Ctrl+NumPad Plus</p> <p>Ctrl+NumPad -</p>
	<p>Select All</p> <p>Click this button to select all the files in the list or directory tree.</p> <p>Ctrl+A</p>

Controls

Item	Description
Changed files	This tree view displays the list of changed files. Select check boxes next to the files to be reverted.
Change list	Use the drop-down list to select the change list that contains the modified files to be reverted. By default, the active change list is suggested.
Delete local copies of added files	Use this check box to revert added files as well as your changes in modified ones.

Unshelve Changes Dialog

Version Control Tool window | context menu | Unshelve Changes

Use this dialog box to restore shelved changes from a shelf to a changelist.

ItemDescription

Existing Changelist	Choose this option restore the shelved changes in one of the existing changelists. Choose the target changelist from the drop-down list.
New Changelist	<p>Choose this option to create a new changelist and restore the changes from the shelf in it.</p> <ul style="list-style-type: none">- Name: in this text box, type the name of the changelist to be created.- Comment: in this text box, type an optional description of the new changelist.- Make this changelist active: select this check box to have PyCharm automatically give the active status to the new change list immediately after the changes are restored in it. When this check box is cleared, the current active changelist remains active. See Changelist for details.- Track context: select this check box to have PyCharm preserve the context of the task associated with the new changelist on its deactivation and restore the context when the changelist becomes active. See Managing Tasks and Context for details.
Remove successfully applied files from the shelf	<ul style="list-style-type: none">- Clear this check box to have PyCharm still display already unshelved changes in the Shelf tab so that you can apply them once more if necessary.- When this check box is selected, the changes are not displayed in the Shelf tab after they are unshelved.

Diagram Reference

In this section:

- [Class Diagram Toolbar, Context Menu and Legend](#)
- [Diagram Toolbar and Context Menu](#)
- [Diagram Preview](#)
- [General Techniques of Using Diagrams](#)

Class Diagram Toolbar, Context Menu and Legend

In this section:

- [Toolbar](#)
- [Context Menu](#)
- [Legend of a Class Diagram](#)

Toolbar

ItemDescription

	Click this button to show methods in the class nodes.
	Click this button to show inner classes in the class nodes.
	Click this button to show fields in the class nodes.
	Click this button to increase the scale of the diagram, or press <code>NumPad+</code> .
	Click this button to decrease the scale of the diagram, or press <code>NumPad-</code> .
	Click this button to restore the actual size of the diagram.
	Click this button to make the contents fit into the current diagram size.
	Click this button to apply the current layout, selected on the context menu of the diagram, or press <code>F5</code> .
	Click this button to save the current diagram as a <code>*.uml</code> file.
	Click this button to save the diagram in an image file with the specified name and path. The possible formats are: <code>jpeg</code> , <code>png</code> , <code>svg</code> , <code>svgz</code> , or <code>gif</code> .
	Click this button to print the diagram.
	Click this button to open the diagram preview in a separate frame, where you can configure the page layout, scale, and headings information.

Context Menu

This section describes only those context menu commands that are not available from the toolbar.

Item	Shortcut	Description
Add class to diagram	<code>Space</code>	Choose this command to add existing class to the diagram background.
Collapse nodes	<code>C</code>	Choose this command to show the containing package of the selected node.
Expand nodes	<code>E</code>	Choose this command to show class diagram of the selected package.
New	<code>Alt+Insert</code>	Choose this command to create a new node element or member .
Refactor		Point to this node to select one of the refactoring commands available in this context.
Analyze		Point to this node to select one of the code analysis commands available in this context.

Legend of a Class Diagram

ItemDescription

	The blue arrow corresponds to the class extension.
--	--

Diagram Toolbar and Context Menu

In this section:

- [Toolbar](#)
- [Context menu](#)

Toolbar

ItemDescription

	Click this button to increase the scale of the diagram. Alternatively, press <code>NumPad+</code> .
	Click this button to decrease the scale of the diagram. Alternatively, press <code>NumPad-</code> .
	Click this button to restore the actual size of the diagram.
	Click this button to make the contents fit into the current diagram size.
	Click this button to apply the current layout, selected on the context menu of the diagram.
	Click this button to save the current diagram in the specified location as <code>xml</code> file.
	Click this button to save the diagram in an image file with the specified name and path. The possible formats are: <code>jpeg</code> , <code>png</code> , <code>svg</code> , <code>svgz</code> , or <code>gif</code> .
	Click this button to print the diagram.
	Click this button to open the diagram preview in a separate frame, where you can configure the page layout, scale, and headings information.

Context menu

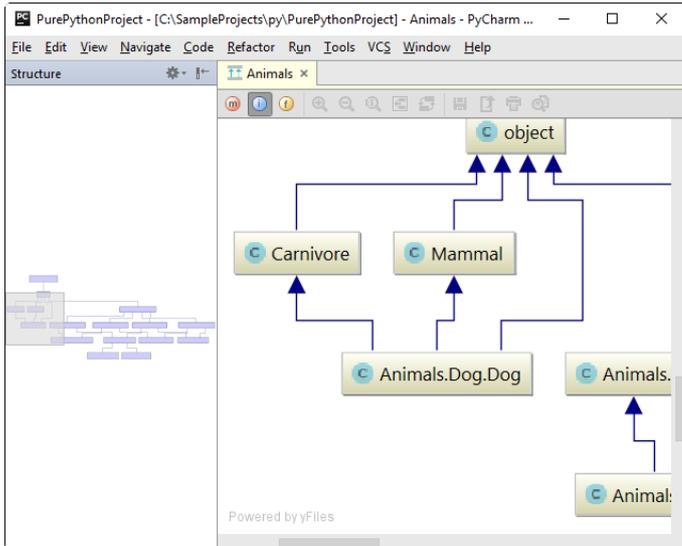
The table below contains commands that are not available from the toolbar.

ItemDescription

New	Use this node to add new elements to a diagram.
Refactor	This node contains refactoring commands, enabled in the current context.
Jump to Source	Choose this command to open the selected diagram node element in the editor.
Find Usages	Choose this command to search for usages of the selected node element.
Layout	Select the desired diagram layout from the submenu.
Show Edge Labels	Check this command to show multiplicities in diagram.

Diagram Preview

Use the [Structure view](#) as a Preview, that allows you to get a "10,000-feet" look on a diagram. The shadow area represents the visible part of a diagram. As you zoom in or out, or change the shape of the PyCharm windows, the size of the shadow area changes accordingly.



To enable the diagram preview

- [Open](#) the Structure tool window.

In the Preview pane, the following actions are available:

- Keeping the mouse button pressed, move the shadow area to obtain the desired view.
- Select one or more nodes in the diagram, and the corresponding nodes in the Preview are marked dark gray.

General Techniques of Using Diagrams

In this section:

- [Selecting elements in diagram.](#)
- [Managing diagram layout.](#)
- [Zooming in and out.](#)
- [Using the magnifier tool.](#)
- [Navigating to source code.](#)
- [Invoking refactoring commands.](#)
- [Finding usages of the selected node element.](#)

To select elements in diagram

- To select an element, just click it in diagram.
- To select multiple adjacent elements, keep `Shift` pressed and click the desired elements, or just drag a *lasso* around the elements to be selected.
- To select multiple non-adjacent elements, keep `Ctrl+Shift` pressed and click the desired elements.
- To select a member of a node element, double-click the node element, and then use the arrow keys, or the mouse pointer.

To manage diagram layout

- Right-click the diagram background, and choose Layout command of the diagram context menu. Next, select the desired layout from the submenu.
- Use Drag-and-drop technique to lay out entities in diagram manually.
- Apply the current layout selected on the context menu of the diagram, by clicking .

To zoom in and out, do one of the following

- Use the  and  toolbar buttons.
- Keeping `Ctrl` key pressed, rotate your mouse wheel up or down.
- Press `NumPad+` or `NumPad-`.

Tip As you zoom in or out, the size of the shadow area in the diagram preview changes accordingly.

To use the magnifier tool

- Keep the `Alt` key pressed, and hover your mouse pointer over the most interesting, or problematic areas of the diagram.



To jump from an element in diagram to the underlying source code

1. Select an element in diagram.
2. Do one of the following:
 - On the context menu of the diagram, choose Jump to Source
 - Press `F4`.
 - Double-click selected element.

The source code of the corresponding source file opens in a separate tab in the editor.

Icons Reference

- [File Types Recognized by PyCharm](#)
- [Symbols](#)

File Types Recognized by PyCharm

PyCharm recognizes numerous file types. Each file type is denoted with a special icon. Custom files types are also allowed. Each file type is associated with one or more extensions that match a certain pattern.

The file types and their extensions are configurable in the [File Types](#) dialog.

Note The file recognized types depend on the installed plugins.

The default types include:

File Type	Icon	Recognized in
Archive files		Professional Edition, Community Edition
Buildout config		Professional Edition
Chameleon template files		Professional Edition
C# files		Professional Edition, Community Edition
C/C++ files		Professional Edition, Community Edition
CSS files		Professional Edition
CoffeeScript files		Professional Edition
Cython files		Professional Edition
Dart files		Professional Edition, Community Edition
Diagram files		Professional Edition
Erlang files		Professional Edition, Community Edition
Files marked as plain text		Professional Edition, Community Edition
Files opened in associated applications		Professional Edition, Community Edition
Handlebars files		Professional Edition
HAML files		Professional Edition
HTML files		Professional Edition, Community Edition
IDL files		Professional Edition, Community Edition
Image files		Professional Edition, Community Edition
JavaFX files		Professional Edition, Community Edition
JavaScript files		Professional Edition
JavaScript test files		Professional Edition
JavaScript files that can be executed on Node.js		Professional Edition
Jade files (refer to the section Using Pug (Jade) Template Engine).		Professional Edition
JSHint configuration files		Professional Edition
JSON files		Professional Edition
JSTestDriver Config files		Professional Edition
Less files		Professional Edition
Localization files		Professional Edition
Mako template files		Professional Edition
Qt UI designer form files		Both editions
reStructuredText files		Both editions
Patch files		Professional Edition, Community Edition
Perl files		Professional Edition, Community Edition
Python scripts		Both editions
Pug files (refer to the section Using Pug (Jade) Template Engine).		Professional Edition
Regular expressions		Professional Edition, Community Edition
RELAX NG Compact Syntax		Professional Edition, Community Edition
Sass files		Professional Edition
SCSS files		Professional Edition
SQL files		Professional Edition, Community Edition
Stylus files		Professional Edition
Text files		Professional Edition, Community Edition
TypeScript files		Professional Edition

XHTML files		Professional Edition, Community Edition
XML DTD files		Professional Edition, Community Edition
XML files		Professional Edition, Community Edition
YAML files		Professional Edition

Symbols

In this section:

- [Common](#)
- [Data Sources](#)

Common

IconDescription

	Python script
	Method
	Function
	Field
	Variable
	Property
	Parameter
	Element
	Directory
	Package
	Source root
	Excluded root
	Template roots
Visibility modifiers	
	private
	protected
	public
	The lock decorator in the upper-left part of a symbol marks a symbol whose name begins with one or more underscores. Such names are considered pseudo-private and are specially treated by an interpreter to restrict their visibility scope.
	The blue bubble in the upper-left part of a symbol marks a symbol whose name begins and ends with two underscores, and is specially treated by an interpreter and standard library .

Data Sources

IconDescription

	DB data source. Also, DBMS-specific icons are used:  DB2  Derby  H2  HSQLDB  MySQL  Oracle  PostgreSQL  SQL Server  SQLite  Sybase
	DB data source with the read-only status, e.g.  for Derby.
	DDL data source
	Database
	Schema
	Table
	View
	Column
	A <code>NOT NULL</code> column
	Column with a primary key
	Column with a foreign key
	Column with an index

	Primary key
	Foreign key
	Index
	Trigger
	Stored procedure or function

Regular Expression Syntax Reference

This section provides a brief summary of regex syntax that can be helpful for creating search and issue navigation patterns.

In this section:

– [RegEx Syntax Reference](#)

– [Tips and Tricks](#)

RegEx Syntax Reference

Character Description

<code>\</code>	Marks the next character as either a special character or a literal. For example: <ul style="list-style-type: none">– <code>n</code> matches the character <code>n</code>. "<code>\n</code>" matches a newline character.– The sequence <code>\\</code> matches <code>\</code> and <code>\(</code> matches <code>(</code>.
<code>^</code>	Matches the beginning of input.
<code>\$</code>	Matches the end of input.
<code>*</code>	Matches the preceding character zero or more times. For example, " <code>zo*</code> " matches either <code>z</code> or <code>zoo</code> .
<code>+</code>	Matches the preceding character one or more times. For example, " <code>zo+</code> " matches <code>zoo</code> but not <code>z</code> .
<code>?</code>	Matches the preceding character zero or one time. For example, <code>a?ve?</code> matches the <code>ve</code> in <code>never</code> .
<code>.</code>	Matches any single character except a newline character.
<code>(subexpression)</code>	Matches <i>subexpression</i> and remembers the match. If a part of a regular expression is enclosed in parentheses, that part of the regular expression is grouped together. Thus a regex operator can be applied to the entire group. <ul style="list-style-type: none">– If you need to use the matched substring within the same regular expression, you can retrieve it using the backreference (<code>\num</code>, where <code>num = 1..n</code>).– If you need to refer the matched substring somewhere outside the current regular expression (for example, in another regular expression as a replacement string), you can retrieve it using the dollar sign (<code>\$num</code>, where <code>num = 1..n</code>).– If you need to include the parentheses characters into a <i>subexpression</i>, use <code>\(</code> or <code>\)</code>.
<code>x y</code>	Matches either <code>x</code> or <code>y</code> . For example, <code>z wood</code> matches <code>z</code> or <code>wood</code> . <code>(z w)oo</code> matches <code>zoo</code> or <code>wood</code> .
<code>{ n }</code>	<code>n</code> is a nonnegative integer. Matches exactly <code>n</code> times. For example, <code>o{2}</code> does not match the <code>o</code> in <code>Bob</code> , but matches the first two <code>o</code> 's in <code>foooooo</code> .
<code>{ n , }</code>	<code>n</code> is a nonnegative integer. Matches at least <code>n</code> times. For example, <code>o{2,}</code> does not match the <code>o</code> in <code>Bob</code> and matches all the <code>o</code> 's in "foooooo". <code>o{1,}</code> is equivalent to <code>o+</code> . <code>o{0,}</code> is equivalent to <code>o*</code> .
<code>{ n , m }</code>	<code>m</code> and <code>n</code> are nonnegative integers. Matches at least <code>n</code> and at most <code>m</code> times. For example, <code>o{1,3}</code> matches the first three <code>o</code> 's in "foooooo". <code>o{0,1}</code> is equivalent to <code>o?</code> .
<code>[xyz]</code>	A character set. Matches any one of the enclosed characters. For example, <code>[abc]</code> matches the <code>a</code> in <code>plain</code> .
<code>[^ xyz]</code>	A negative character set. Matches any character not enclosed. For example, <code>[^abc]</code> matches the <code>p</code> in <code>plain</code> .
<code>[a-z]</code>	A range of characters. Matches any character in the specified range. For example, "[<code>a-z</code>]" matches any lowercase alphabetic character in the range <code>a</code> through <code>z</code> .
<code>[^ m-z]</code>	A negative range characters. Matches any character not in the specified range. For example, <code>[^m-z]</code> matches any character not in the range <code>m</code> through <code>z</code> .
<code>\b</code>	Matches a word boundary, that is, the position between a word and a space. For example, <code>er\b</code> matches the <code>er</code> in <code>never</code> but not the <code>er</code> in <code>verb</code> .
<code>\B</code>	Matches a non-word boundary. <code>ea*r\b</code> matches the <code>ear</code> in <code>never early</code> .
<code>\d</code>	Matches a digit character. Equivalent to <code>[0-9]</code> .
<code>\D</code>	Matches a non-digit character. Equivalent to <code>[^0-9]</code> .
<code>\f</code>	Matches a form-feed character.
<code>\n</code>	Matches a newline character.
<code>\r</code>	Matches a carriage return character.
<code>\s</code>	Matches any white space including space, tab, form-feed, etc. Equivalent to <code>[\f\n\r\t\v]</code> .
<code>\S</code>	Matches any nonwhite space character. Equivalent to <code>[^ \f\n\r\t\v]</code> .
<code>\t</code>	Matches a tab character.
<code>\v</code>	Matches a vertical tab character.
<code>\w</code>	Matches any word character including underscore. Equivalent to <code>[A-Za-z0-9_]</code> .
<code>\W</code>	Matches any non-word character. Equivalent to <code>[^A-Za-z0-9_]</code> .
<code>\ num</code>	Matches <code>num</code> , where <code>num</code> is a positive integer, denoting a reference back to remembered matches. For example, <code>(.)\1</code> matches two consecutive identical characters.
<code>\ n</code>	Matches <code>n</code> , where <code>n</code> is an octal escape value. Octal escape values should be 1, 2, or 3 digits long.

For example, `\11` and `\011` both match a tab character.

`\0011` is the equivalent of `\001` & `1`.

Octal escape values should not exceed 256. If they do, only the first two digits comprise the expression. Allows ASCII codes to be used in regular expressions.

`\x n` Matches `n`, where `n` is a hexadecimal escape value. Hexadecimal escape values must be exactly two digits long. For example, `\x41` matches `A`. `\x041` is equivalent to `\x04` & `1`. Allows ASCII codes to be used in regular expressions.

`\\$` Escapes `$`.

`\l` Changes the case of the next character to the lower case.

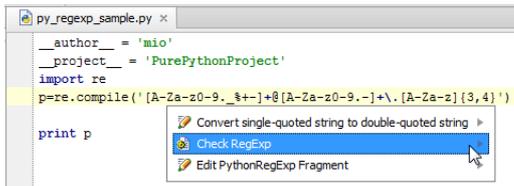
`\u` Changes the case of the next character to the upper case.

`\L` Changes the case of all the subsequent characters up to `\E` to the lower case.

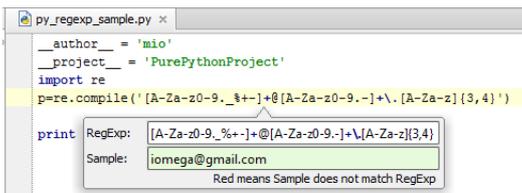
`\U` Changes the case of all the subsequent characters up to `\E` to the upper case.

Tips and Tricks

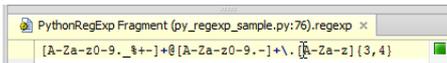
PyCharm provides intention actions to check validity of the regular expressions, and edit regular expressions in a scratchpad. Place the caret at a regular expression, and press `Alt+Enter`. The suggestion list of intention actions, available in this context, appears:



Choose `Check RegExp`, and press `Enter`. The dialog box that pops up, shows the current regular expression in the upper pane. In the lower pane, type the string to which this expression should match. If the regular expression matches the entered string, the background becomes green. If the regular expression doesn't match, then the background is red.



Choose `Edit RegExp Fragment`, and press `Enter`. The regular expression opens for editing in a separate tab in the editor. However, this is but a scratchpad, and no file is physically created:



As you type in the scratchpad, all changes are synchronized with the original regular expression. Press `Escape` to close the editor tab.

Copy and Paste Between PyCharm and Explorer/Finder

PyCharm enables tight interaction with the native file managers (Explorer on Windows, or Finder on Mac), and allows you to exchange files and directories via the system clipboard, using numerous techniques:

- Cut, copy and paste files and directories from the Project tool window of PyCharm to a directory in the file manager, and vice versa, using menu commands and keyboard shortcuts:

Keyboard shortcut	Function	Use this shortcut to...
Ctrl+C	Copy	Copy selected text to the Clipboard.
Ctrl+X	Cut	Cut to the Clipboard.
Ctrl+V	Paste	Paste from the Clipboard.

- Move (drag) or copy (Ctrl+drag) a file or directory from the file manager to a directory in the Project tool window.
- Move (drag) or copy (Ctrl+drag) a file from the Project tool window to a directory in the file manager.
- Open any file for editing, by dragging it from a file manager to the editor.

Scope Language Syntax Reference

The **scopes language** is used in specifying project **scopes** involved in various kinds of analysis.

In this section:

- [Sets of Classes](#)
- [Sets of Files](#)
- [Modifiers](#)
- [Logical Operators](#)
- [Defining Scopes](#)
- [Examples](#)

Sets of Classes

- Single class is defined by a class name, i.e. `com.intellij.openapi.MyClass`
- Set of all classes in a package, not recursing into subpackages, is defined by an asterisk after dot, for example: `com.intellij.openapi.*`
- Set of all classes in a package including contents of subpackages, is defined by an asterisk after double dot, for example `com.intellij.openapi..*`

Sets of Files

- Single file is defined by a file name, i.e. `MyDir/MyFile.txt`
- Set of all files in a directory, not recursing into subdirectories, is defined by an asterisk after slash, for example: `file:src/main/myDir/*`
- Set of all files in a directory including contents of subdirectories, is defined by an asterisk after double slash, for example `file:src/main/myDir/**`

Modifiers

Location modifiers

help you specify whether the desired set is located in the source files, library classes or test code in the form of location modifiers `src:`, `lib:`, `file:`, or `test:`.

For example, the following scope

```
src:com.intellij.openapi.*
```

implies all classes under the source root in the `com.intellij.openapi` package, excluding subpackages.

The default location is the module root.

Module modifiers

help you narrow down the scope by specifying the name of the related module in one of the following ways:

```
src[module name]:<E>
lib[module name]:<E>
test[module name]:<E>
```

For example, the following scope

```
src[MyModule]:com.intellij.openapi.*
```

implies all classes under the source folders related to the module `MyModule` in the package `com.intellij.openapi`, excluding subpackages.

Logical Operators

The scope language allows you to use common logical operators:

```
&& for AND
|| for OR
! for NOT
```

Besides that, the parentheses can be used to join the logical operators into groups. For example, the following scope

```
(<a>||<b>)&&<c>
```

implies either `<a>` and `<c>`, or `` and `<c>`.

Defining Scopes

Scopes are defined in the [Scopes](#) dialog box in the following ways:

- Manually
- With the pointing device

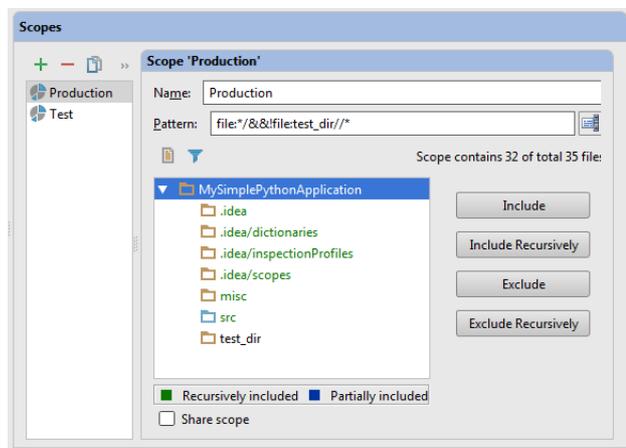
Manually

Specify file masks in the Pattern text box, or click  and type the pattern in the editor.

Using the Mouse Pointer

Select files and folders in the project tree view and click the buttons Include, Include Recursively, Exclude, and Exclude Recursively. For information about the controls, refer to [Scope](#) page description.

Based on the inclusion/exclusion of file and directories, PyCharm creates an expression and displays it in the Pattern field.



Examples

- `file:*.js||file:*.coffee` - include all JavaScript and CoffeeScript files.
- `file:*js&&!file:*.min.*` - include all JavaScript files except those that were generated through minification, which is indicated by the `min` extension.

Project and IDE Settings

In this topic:

- [Overview](#)
- [Project Settings](#)
- [IDE Settings](#)
- [Locations of directories](#)

Overview

There are two types of settings that define your preferred environment:

- Project Settings, that apply to a specific project. They are marked with  in the Settings/Preferences dialog.
- IDE Settings, that are common for all projects and refer to the project-independent aspects.

Project Settings

Project settings are stored with each specific project as a set of `xml` files under the `.idea` folder. If you specify the [default project settings](#), these settings will be automatically used for each newly created project.

The settings that pertain to a project, are marked with the icon  in the Settings/Preferences dialog.

IDE Settings

IDE settings are stored in the dedicated directories under PyCharm home directory. The PyCharm directory name is composed of the product name and version.

For example:

Windows

- `<User home>\.PyCharmXX\config` that contains user-specific settings.
- `<User home>\.PyCharmXX\system` that stores PyCharm data caches.

`<User home>` in WindowsXP is `C:\Documents and Settings\<User name>\`; in Windows Vista it is `C:\Users\<User name>\`

Linux

- `/.PyCharmXX/config` that contains user-specific settings.
- `~/.PyCharmXX/system` that stores PyCharm data caches.

macOS

- `~/Library/Application Support/PyCharmXX` contains the catalog with plugins.
- `~/Library/Preferences/PyCharmXX` contains the rest of the configuration settings.
- `~/Library/Caches/PyCharmXX` contains data caches, logs, local history, etc. These files can be quite significant in size.
- `~/Library/Logs/PyCharmXX` contains logs.

Locations of directories

The `config` directory has several subfolders that contain xml files with your personal settings. You can easily share your preferred keymaps, color schemes, etc. by copying these files into the corresponding folders on another PyCharm installation. Prior to copying, make sure that PyCharm is not running, because it can erase the newly transferred files before shutting down.

The following is the list of some of the subfolders under the `config` folder, and the settings contained therein.

Folder name	User Settings
<code>codestyles</code>	Contains code style schemes .
<code>colors</code>	Contains editor colors and fonts customization schemes.
<code>filetypes</code>	Contains user-defined file types .
<code>inspection</code>	Contains code inspection profiles .
<code>keymaps</code>	Contains PyCharm keyboard shortcuts customizations.
<code>options</code>	Contains various options, for example, feature usage statistics and macros.
<code>templates</code>	Contains user-defined live templates .
<code>tools</code>	Contains configuration files for the user-defined external tools .
<code>shelf</code>	Contains shelved changes .

Locations of the `config`, `system`, `plugins` directories [can be modified in](#) `idea.properties` file.

PyCharm makes it possible to change `*.vmoptions` and `idea.properties` files without editing them in the PyCharm installation folder.

To create an empty `idea.properties` file or to copy `*.vmoptions` file, choose the Help | Edit Custom Properties or Help | Edit Custom VM Options on the main menu respectively. Refer to the [menu items description](#) for details.

To learn how to change the `idea.properties` file, read the section [File 'idea.properties'](#).

Directories Used by PyCharm to Store Settings, Caches, Plugins and Logs

Location of the IDE files depends on the operating system, and PyCharm version.

In this section:

- [Windows](#)
- [Linux and the other UNIX systems](#)
- [macOS](#)

Windows

All the files are located under this directory by default:

Windows Vista, 7, 8, 10

```
<SYSTEM DRIVE>\Users\<<USER ACCOUNT NAME>\.<PRODUCT><VERSION>
```

Windows XP

```
<SYSTEM DRIVE>\Documents and Settings\<<USER ACCOUNT NAME>\.<PRODUCT><VERSION>
```

Example

- PyCharm 2016.1

```
c:\Users\John\.PyCharm2016.1\
```

Refer to the page [Project and IDE Settings](#).

Linux and the other UNIX systems

Product directory starting with dot can be found in the user home directory. The pattern is:

```
~/.<PRODUCT><VERSION>
```

~ is an alias for the home directory, for example `/home/john`.

macOS

- Configuration

```
~/Library/Preferences/<PRODUCT><VERSION>
```

- Caches

```
~/Library/Caches/<PRODUCT><VERSION>
```

- Plugins

```
~/Library/Application Support/<PRODUCT><VERSION>
```

- Logs

```
~/Library/Logs/<PRODUCT><VERSION>
```

where <PRODUCT> is PyCharm.

Synchronizing and Sharing Settings

In this section:

- [Synchronizing settings](#)
- [Prerequisite](#)
- [Basics](#)
- [What is synched?](#)
- [Connecting to Settings Repository](#)
- [Configuring login to Settings Repository](#)
- [Sharing settings](#)

Synchronizing settings

Prerequisite

Before you start working with Settings Repository, make sure that the Settings Repository plugin is downloaded and enabled, as described in [Installing, Updating and Uninstalling Plugins](#).

Basics

PyCharm makes it possible to synchronize settings across several different PyCharm installations, or auto-configure new installations, using settings from the server.

Once logged in, the settings are synchronized with the server based on "updated" timestamp stored in each settings group. The most recent ones are used and are either sent to server or applied locally from the server.

PyCharm always works with settings stored locally. So, the local settings are used, if, at some moment, the server is not available. On the following connection all updated settings will be pushed to the server.

When logged in to the server, once some settings are updated locally inside PyCharm, they are subsequently sent to the server. But local settings are only updated on PyCharm startup.

The server stores only one set of settings for each user.

What is synched?

Settings Repository stores almost all of the IDE and project settings, except for some platform-specific ones, and those containing local paths. This includes code style settings, keymaps, fonts and colors scheme, inspection profiles, and others.

Connecting to Settings Repository

You can connect to the Settings Repository in various ways:

- During the first PyCharm startup after installing the plugin, through the [Login to IntelliJ Configuration Server](#) dialog box.
- At any time [during a session](#) through the Login to IntelliJ Configuration Server dialog box.
- During PyCharm startup according to the previously configured [settings](#):
 - Silently
 - Through the [Login to IntelliJ Configuration Server](#) dialog box

To connect to the Settings Repository during a session

1. At any time during a session, click the IntelliJ Configuration Server Status button (↔ when connected to the server, or ↔, when disconnected) on the Status bar.
2. In the dialog box that opens, click the Login button.
3. Specify the login and password of your JetBrains Account or create a [JetBrains Account](#) if you do not have it.
To connect to the IntelliJ Configuration Server through a Proxy server, select the Use HTTP Proxy check box and specify the Proxy server parameters in the dialog box controls. See a detailed description of controls in the [Login to IntelliJ Configuration Server](#) dialog box reference.

Configuring login to Settings Repository

To configure login to the Server during the next PyCharm startup

1. Click the IntelliJ Configuration Server Status button ↔ on the Status bar.
2. In the dialog box that opens, specify whether you want to log in to the IntelliJ Configuration Server during the next startup and how you want to do it. The following options are available:
 - Show [login](#) dialog box
 - Login silently
 - Do not login

Select the desired option and click OK.

Tip The specified configuration settings will be applied during the next PyCharm startup.

Sharing settings

For sharing project settings, `.idea` project configuration directory should be shared via [version control](#).

If you decide to share PyCharm project files with the other developers, follow these guidelines.

Directory based project format (`.idea` directory) is used by all the PyCharm versions by default. Here is what you need to share:

- All the files under `.idea` directory in the project root, except the `workspace.xml` and `tasks.xml` files, which store user-specific settings.

Be careful about sharing the following:

- `dataSources.ids` , `datasources.xml` (can contain database passwords)

You may consider not to share the following:

- user `dictionaries` folder (to avoid conflicts if another developer has the same name).

Networking in PyCharm

PyCharm requires Internet connection for a wide variety of tasks. For example:

- Checking for PyCharm [updates](#)
- [Code inspections](#) that can verify external resources
- Communication with the [version control](#) servers, [task servers](#)
- Anonymous usage statistics
- Working with [remote interpreters](#)
- [Installing and upgrading Python packages](#)

Besides that, PyCharm provides IPC for commands (for example, open files), and the built-in Web server.

Some of the communication requirements are configurable:

- Checking for updates can be turned off. To disable checking for updates, open the page [Settings | Updates](#), and clear the check box Check for updates in channel.
- To disable code inspection that highlights dead links, open the page [Settings | Inspections](#), and clear the check box to the left of the HTML inspection Non-existent web resources.
- You can control the frequency of sending usage statistics, or even completely disable this function in the page [Settings | Usage statistics](#).

Tuning PyCharm

In this part:

- Tuning PyCharm
 - [Changing PyCharm properties](#)
 - [Managing *.vmoptions file](#)
 - [Example: Increasing the heap size](#)
 - [Managing idea.properties file](#)
 - [Example: Changing case of unicode literals](#)
 - [Specifying custom JDK, properties or vmoptions files across platforms](#)
- [File 'idea.properties'](#)

Changing PyCharm properties

PyCharm makes it possible to change `*.vmoptions` and `idea.properties` files without editing them in the PyCharm installation folder.

To create an empty `idea.properties` file or to copy `*.vmoptions` file, choose the Help | Edit Custom Properties or Help | Edit Custom VM Options on the main menu respectively. Refer to the [menu items description](#) for details.

Managing *.vmoptions file

The location of the `*.vmoptions` file depends on the operating system you are currently using:

- For **Windows** systems: `<PyCharm installation folder>/bin/pycharm.exe.vmoptions` or `<PyCharm installation folder>/bin/pycharm64.exe.vmoptions`
- For ***NIX** systems: `<PyCharm installation folder>/bin/pycharm.vmoptions` or `<PyCharm installation folder>/bin/pycharm64.vmoptions`
- For **macOS** systems, you have to make a copy of the `pycharm.vmoptions` file in the IDE preferences folder and then edit this copy. The reason is that app bundle is signed and you should not modify any files inside the bundle.

For the older versions, the settings are stored in `/Applications/PyCharm <version>.app/Contents/Info.plist`.

Tip Mind the abbreviation scheme of the PyCharm name: `xx` corresponds to the PyCharm version.

To avoid editing files in the PyCharm installation folder, one should:

1. Do one of the following:
 - Use the main menu command Help | Edit Custom VM Options to create a copy of the `pycharm.vmoptions` file in the user home. Refer to [Edit Custom VM Options](#) for details.
 - Copy the existing file from the PyCharm installation folder somewhere and save the path to this location in the environment variable `PYCHARM_VM_OPTIONS`.
 - Copy the existing file `<PyCharm installation folder>/bin/pycharm.exe.vmoptions` or `<PyCharm installation folder>/bin/pycharm64.exe.vmoptions` from the PyCharm installation folder into the location under your user home.
2. Edit this file in the new location.

If `PYCHARM_VM_OPTIONS` environment variable is defined, or the `*.vmoptions` file exists, then this file is used instead of the one located in the PyCharm installation folder.

Example: Increasing the heap size

To increase PyCharm heap size, you should copy the original `pycharm.vmoptions` file to the above-mentioned location, and then modify the `-Xmx` setting.

Managing IDEA.properties file

The file `idea.properties`, located in the `bin` directory of the PyCharm installation folder, should not be edited. Instead of editing the original `idea.properties`, create file `idea.properties` in the location specified below, open it for editing and add the required properties.

So, depending on your platform:

- For **Windows**: in `%USERPROFILE%\PyCharm XX`
- For ***NIX**: in `~/PyCharm XX`
- For **macOS**: in `~/Library/Preferences/PyCharm XX`

Example: Changing case of unicode literals

PyCharm allows defining whether non-ascii characters should use literals like `'\u00AB'` or `'\00ab'`.

This behavior is controlled by the system property `idea.native2ascii.lowercase`. By default, upper case characters are used.

If it is desirable to use lower case characters, create the file `idea.properties` in the location specified above, open it for editing and add the following line:

```
idea.native2ascii.lowercase=true
```

Specifying custom JDK, properties or vmoptions files across platforms

A custom JDK, `*.properties` and `*.vmoptions` files are specified across platforms in a unified way.

All the launchers look at the following environment variables:

- `$(IDE-NAME>_JDK (<IDE-NAME>_JDK_64)`
- `$(IDE-NAME>_PROPERTIES`
- `$(IDE-NAME>_VM_OPTIONS`

File 'idea.properties'

The file `idea.properties`, located in the `bin` directory of the PyCharm installation folder, should not be edited. Instead of editing the original `idea.properties`, create file `idea.properties` in the location specified below, open it for editing and add the required properties.

So, depending on your platform:

- For **Windows**: in `%USERPROFILE%\PyCharmXX`
- For ***NIX**: in `~/PyCharmXX`
- For **macOS**: in `~/Library/Preferences/PyCharmXX`

You can create an empty file `idea.properties` and open it in the editor by choosing the Help | Edit Custom Properties command on the main menu.

Refer to the description of [Edit custom properties](#) command.

NameDescriptionProperty setting

Name	Description	Property
Note The Windows users should use forward slashes, i.e. <code>c:/idea/system</code> .		
<code>\$(idea.home)</code> macro	Use <code>\$(idea.home)</code> macro to specify location relative to IDE installation home. Also use <code>\${xxx}</code> where <code>xxx</code> is any java property (including defined in the previous lines of this file) to refer to its value.	
Path to the IDE config folder	Uncomment this option if you want to customize path to IDE config folder.	<code>idea.config.path=\${user.home}/.PyCharm/config</code>
Path to IDE system folder	Uncomment this option if you want to customize path to IDE system folder.	<code>idea.system.path=\${user.home}/.PyCharm/system</code>
Path to user installed plugins	Uncomment this option if you want to customize path to user installed plugins folder.	<code>idea.plugins.path=\${user.home}/.PyCharm/config/plugins</code>
Path to IDE logs folder	Uncomment this option if you want to customize path to IDE logs folder.	<code>idea.log.path=\${user.home}/.PyCharm/system/log</code>
Maximum file size	Maximum file size (kilobytes) IDE should provide code assistance for. The larger file is, the slower its editor works and higher overall system memory requirements are, if code assistance is enabled. Remove this property or set to a very large number, if you need code assistance for any files to be available, regardless of their size.	<code>idea.max.intellisense.filesize=2500</code>
Console cyclic buffer	This option controls console cyclic buffer: keeps the console output size not higher than the specified buffer size (Kb). Older lines are deleted. In order to disable cycle buffer, use <code>idea.cycle.buffer.size=disabled</code>	<code>idea.cycle.buffer.size=1024</code>
Launcher	Configure if a special launcher should be used when running processes from within IDE. Using Launcher enables "soft exit" and "thread dump" features.	<code>idea.no.launcher=false</code>
Classpath	To avoid too long classpath	<code>idea.dynamic.classpath=false</code>
ProcessCanceledException	Uncomment this property to prevent IDE from throwing <code>ProcessCanceledException</code> when user activity detected. This option is only useful for plugin developers, while debugging PSI related activities performed in background error analysis thread. DO NOT UNCOMMENT THIS UNLESS YOU'RE DEBUGGING THE IDE ITSELF. Significant slowdowns and lockups will happen otherwise.	<code>idea.ProcessCanceledException=disabled</code>
Pop-up window weight	There are two possible values of <code>idea.popup.weight</code> property: "heavy" and "medium". If you have WM configured as "Focus follows mouse with Auto Raise" then you have to set this property to "medium". It prevents problems with the pop-up menus on some configurations.	<code>idea.popup.weight=heavy</code>
System anti-aliasing	Use default anti-aliasing in system, i.e. override value of "Settings Editor Appearance Use anti-aliased font" option. May be useful when using Windows Remote Desktop Connection for instance.	<code>idea.use.default.anti aliasing.in.editor=false</code>
Repaint	Disabling this property may lead to visual glitches like blinking	

	and fail to repaint on certain display adapter cards.	<code>sun.java2d.noddraw=true</code>
Editor performance	Removing this property may lead to editor performance degradation under Windows.	<code>sun.java2d.d3d=false</code>
Slow scrolling	Workaround for slow scrolling in JDK6.	<code>swing.bufferPerWindow=false</code>
Editor performance under X Window	Removing this property may lead to editor performance degradation under X Window.	<code>sun.java2d.pmoFFscreen=false</code>
Avoid long hangs	Workaround to avoid long hangs while accessing clipboard under macOS.	<code>ide.mac.useNativeClipboard=True</code>
Maximum load size	Maximum size (kilobytes) IDEA will load for showing past file contents - in Show Diff or when calculating Digest Diff	<code>idea.max.vcs.loaded.size.kb=20480</code>
Copy library jars	IDE copies library jars to prevent their locking. If copying is not desirable, specify "true"	<code>idea.jars.nocopy=false</code>
Start the JVM in debug mode	The VM option value to be used to start the JVM in debug mode. Some JREs define it in a different way (-XXdebug in Oracle VM)	<code>idea.xdebug.key=-Xdebug</code>
Switch into JMX 1.0 compatibility mode.	Uncomment this option to be able to run PyCharm using J2SDK 1.5+ while working with application servers (like WebLogic) running 1.4.	<code>jmx.serial.form=1.0</code>
Fatal errors notifications	Change to 'enabled' if you want to receive instant visual notifications about fatal errors that happen to an IDE or plugins installed.	<code>idea.fatal.error.notification=disabled</code>

Index of Menu Items

PyCharm menu structure doesn't align with PyCharm help structure. This page lists PyCharm menu items, linked to the corresponding help topics.

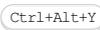
Warning! – This table corresponds to the Windows and Linux platforms. Commands specific for macOS have special notes.
– Any additional plugins and external tools make changes to the main menu.

In this section:

- [File](#)
- [Edit](#)
- [View](#)
- [Navigate](#)
- [Code](#)
- [Refactor](#)
- [Run](#)
- [Tools](#)
- [VCS](#)
- [Window](#)
- [Help](#)

File

MenuKeyboardDescription
item shortcut

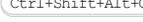
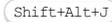
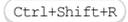
New Project...		Use this command to create a new project .
New...		Use this command to create a new element in a project . This command is only available in the corresponding context.
New Scratch File		Choose this command to create a new scratch file .
Open...		Use this command to open the specified directory, or an existing PyCharm project. A directory that contains a project is marked with  icon. Refer to the section Opening, Reopening, and Closing Projects . This command is duplicated with  icon on the main toolbar.
Save as...		Use this command to save a file currently opened in the editor, in the specified directory. Refer to the section Saving and Reverting Changes . This operation produces the same result, as copy refactoring .
Attach Project		This command appears on the File menu, ONLY when the options Open project in a new window or Open project in the same window have been selected in the Project opening section of the System Settings page of Settings/Preferences. It allows attaching projects to the already opened one. See Opening Multiple Projects .
Open Recent		Use this command to open one of the recent projects. Refer to the reopening projects procedure.
Close Project in Current Window		Use this command to close all the projects opened in the current window. Refer to the sections Opening Multiple Projects and Opening, Reopening and Closing Projects for details.
Settings... (on Windows/*nix)/PyCharm Preferences (on macOS)		Use this command to change the project and IDE configurations in the Settings/Preferences dialog . See also the section Configuring Project and IDE Settings . This command is duplicated with  icon on the main toolbar. This command is available on Windows/Linux. On Mac OS it appears on the PyCharm menu and has the name Preferences.
Default Settings...		Choose this command to change settings that will apply to all newly created projects. Refer to the section Accessing Default Settings .
Import Settings...		Choose this command to import settings from an archive .
Export Settings...		Choose this command to export settings to an archive .
Settings Repository...		Choose this command to invoke the Settings Repository dialog .
Save All		Choose this command to save all changes , when editing is over. This command is duplicated with  icon on the main toolbar.
Synchronize		Choose this command to check the PyCharm caches and bring them up-to-date by keeping in sync with external changes. This command is duplicated with  icon on the main toolbar.
Invalidate Caches/Restart...		Choose this command to clean the system cache .
Export to HTML...		Use this command results to save selected files in HTML format .
Print...		Choose this command to print selected file on the default printer. Refer to the Print dialog description.
Add to Favorites		Use this command to add the selected files to the list of Favorites . Click the right arrow to select the list of

favorites you want to be modified.
Refer to the description of the [Favorites tool window](#).

File Encoding	Use this command to change encoding of an individual file . See also the section Encoding .
Line Separators	Use this command to select the desired line separator style. Refer to the section Configuring Line Separators .
Make File Read-Only / Make File Writable	Use these toggle commands to change read-only status of a file selected in the Project tool window, or currently active in the editor. If a file is made read-only, it is marked with  , and doesn't allow editing. You can also toggle read-only attribute of a file in the Status bar .
Power Save Mode	Use this mode if you are working with a laptop. If Power-Save mode is on, then the background processes are turned off, to minimize the power consumption. You can also turn this mode on or off by clicking  in the Status bar .
Exit	Choose this command to quit PyCharm. This command is available on Windows/Linux. On Mac OS it appears on the PyCharm menu and has the name Quit PyCharm.

Edit

MenuKeyboardDescription item shortcut

Undo <action>		Use this command to roll actions back . This command is duplicated with  icon on the main toolbar.
Redo <action>		Use this command to repeat the last actions . This command is duplicated with  icon on the main toolbar.
Cut		Choose this command to take the selected characters to the clipboard and delete them. Refer to the section Cutting, Copying, and Pasting . This command is duplicated with  icon on the main toolbar.
Copy		Choose this command to take the selected characters to the clipboard. Refer to the section Cutting, Copying, and Pasting . This command is duplicated with  icon on the main toolbar.
Copy Path		Choose this command to take the path to the selected symbol to the clipboard. Refer to the section Cutting, Copying, and Pasting .
Copy as Plain Text		Choose this command to take the selected fragment to the clipboard without formatting. Refer to the section Settings/Preferences Editor General .
Copy Relative Path		Choose this command to take a reference to a symbol to the clipboard. Refer to the section Cutting, Copying, and Pasting .
Paste		Choose this command to place the latest entry from the Clipboard at the insertion point. Refer to the section Cutting, Copying, and Pasting . This command is duplicated with  icon on the main toolbar.
Paste from History...		Choose this command to place at the insertion point the selected entry from the Clipboard. Refer to the section Cutting, Copying, and Pasting .
Paste Simple		Choose this command to place the last entry from the Clipboard at the insertion point as plain text. Refer to the section Cutting, Copying, and Pasting .
Delete		Choose this command to delete the selected files, or folder from the project tool window, or selected fragment of text from the active editor.
Find	Point to this node to reveal the sub-menu of search commands:	
Find/Replace	 / 	Find or replace text in a current file . These commands are duplicated by  and  icons on the main toolbar.
Find Next/Find Previous (Move to Next/Previous Occurrence)	 / 	Use these commands to navigate through the search results in a file. See Finding and Replacing Text in File .
Find Word at Caret		Use this command to jump to the next occurrence of the word where the caret rests. See Finding and Replacing Text in File .
Select All Occurrences		Use this command to find and select all the occurrences of an item.
Add Selection for Next Occurrence		Use this command to select the next occurrence of an item.
Unselect Occurrence		Use this command to remove selection from the last selected occurrence of an item.
Find in Path/Replace in Path	 / 	Use these commands to search for, and replace a text fragment in a whole project. Refer to the section Finding and Replacing text in Project .
Search/Replace		Use these commands to perform structural search or replace.

Structurally		Refer to the section Structural Search and Replace for details.
Find Usages	Alt+F7	Use this command to search for the usages of a symbol across an entire project. Refer to the section Finding Usages in Project .
Find Usages Settings	Ctrl+Shift+Alt+F7	Use this command to search for the usages of a symbol across an entire project, after setting the desired search options. Refer to the section Finding Usages in Project .
Show Usages	Ctrl+Alt+F7	Use this command to bring up a list of the usages of a symbol across the whole project. Refer to the section Viewing Usages of a Symbol .
Find Usages in a File	Ctrl+F7	Refer to the section Finding Usages in the Current File .
Highlight Usages in a File	Ctrl+Shift+F7	Use this command to visualize usage of a symbol in the current file. Refer to the section Highlighting Usages .
Recent Find Usages	Ctrl+E	Choose this command to view the recent search results. Refer to Viewing Recent Find Usages .

Macros		Point to this node to reveal the sub-menu of the macros-related commands. Refer to the section Using Macros in the Editor .
Column Selection Mode	Shift+Alt+Insert	Use this command to toggle between column selection and line selection modes. Refer to the section Selecting Text in the Editor .
Select All	Ctrl+A	Choose this command to select all contents of the current file. Refer to the section Selecting Text in the Editor .
Extend Selection	Ctrl+W	Choose this command to select the current word. Use this command successively to extend selection. Refer to the section Selecting Text in the Editor .
Shrink Selection	Ctrl+Shift+W	Choose this command to unselect the currently selected word. Use this command successively to shrink selection. Refer to the section Selecting Text in the Editor .
Join Lines	Ctrl+Shift+J	Choose this command to join lines or literals .
Fill Paragraph		Choose this command to create soft wraps in a paragraph.
Duplicate Lines	Ctrl+D	Choose this command to duplicate a line or fragment of text. Refer to Adding, Deleting and Moving Code Elements .
Indent Selection/Unindent Selection	Tab / Shift+Tab	Choose this command to change indentation of the line at caret. Refer to the section Changing Indentation .
Toggle Case	Ctrl+Shift+U	Choose this command to change case of the selection. See Toggling Case .
Convert Indents		Point to this node to reveal the sub-menu of the possible indentation and toggle indentation style. Refer to Changing Indentation .
Encode XML/HTML Special Characters		Choose this command to convert the selected special character to its HTML name in the format <code>&char;</code> .
Edit as Table		Choose this command to invoke the table editor for the current documents.

View

Menu item	Keyboard shortcut	Description
-----------	-------------------	-------------

Tool Windows		Point to this node to reveal the list of the available tool windows. Refer to the section Manipulating the Tool Windows .
Quick Definition	Ctrl+Shift+I	Choose this command to open the quick definition popup. Refer to the section Viewing Definition .
Quick Documentation	Ctrl+Q	Choose this command to view quick documentation popup window.
Parameter Info	Ctrl+P	Choose this command to view method parameter information .
Context Info	Alt+Q	Choose this command to show the current cursor position , if it runs out of the visible editor pane.
Jump to Source	F4	Choose this command to edit a file selected in a tool window. The file opens in the editor.
Recent Files	Ctrl+E	Choose this command to show the pop-up list of recently opened files and tool windows, and navigate to them.
Recently Changed Files	Ctrl+Shift+E	Choose this command to show the pop-up list of recently changed files and navigate to them .
Recent Changes	Shift+Alt+C	Choose this command to open the pop-up list of recent changes .
Compare with Clipboard		Choose this command to compare the file currently opened in the editor with the contents of the system clipboard. See Comparing Files .
Quick Switch Scheme	Ctrl+Back Quote	Choose this command to switch between schemes .
Toolbar		Select or clear this check command to show or hide the main toolbar .
Tool Buttons		Select or clear this check command to show or hide the tool window buttons .
Status Bar		Select or clear this check command to show or hide the Status toolbar .
Navigation Bar		Select or clear this check command to show or hide the Navigation bar .
Active Editor		Point to this node to reveal the list of nested check commands. These commands apply to the active editor and is only available when it exists.
Show Whitespaces		Select or clear this check command to show or hide the whitespaces in the text.
Show Line Numbers		Select or clear this check command to show or hide line numbers.

Show Gutter Icons	Select or clear this check command to show or hide the icons in the left gutter.
Show Indent Guides	Select or clear this check command to show or hide vertical indent markers.
Use Soft Wraps	Select or clear this check command to show or hide soft wrap markers in the text.
Show Import Popups	Select or clear this check command to show or hide import popups.

BiDi Text Direction	Point to this node to select the direction of text in the string literals containing RTL strings and tokens. Refer to the page Text Direction .
Enter/Exit Presentation Mode	Choose this command to enter or exit presentation mode .
Enter/Exit Distraction Free Mode	Choose this command to enter or exit distraction-free mode .
Enter/Exit Full Screen	Choose this command to enter or exit full screen mode .

Navigate

MenuKeyboard Description item shortcut

Class/File/Symbol	Ctrl+N / Ctrl+Shift+N Ctrl+Shift+Alt+N	Choose these commands to find and jump to a class, file, or symbol by name .
Custom Folding...	Ctrl+Alt+Period	Choose this command to navigate between custom regions .
Line...	Ctrl+G	Choose this command to navigate to the specified line of code .
Back/Forward	Ctrl+Alt+Left / Ctrl+Alt+Right	Choose these commands to go through the history of the recently navigated items . These commands are duplicated with ↶ and ↷ buttons on the main toolbar.
Last/Next Edit Location	Ctrl+Shift+Backspace	Choose these commands to jump to the latest edit location and back.
Bookmarks		Point to this node to reveal the sub-menu of commands related to using bookmarks .
Select In...	Alt+F1	Choose this command to select the desired component from the pop-up list of possible targets .
Jump to Navigation Bar	Alt+Home	Choose this command to navigate across your project using the Navigation Bar .
Declaration	Ctrl+B	Choose this command to jump to a declaration of a symbol .
Implementation(s)	Ctrl+Alt+B	Choose this command to jump to an implementation of a method .
Type Declaration	Ctrl+Shift+B	Choose this command to jump to the type declaration of a symbol .
Super Method	Ctrl+U	Choose this command to jump to a super method of the method at caret.
Test	Ctrl+Shift+T	Choose this command to navigate to an existing test, or create a test. See section Creating Tests .
Related Symbol...	Ctrl+Alt+Home	.
File Structure	Ctrl+F12	Choose this command to navigate through the source code using the File Structure view .
File Path	Ctrl+Alt+F12	See Navigating to File Path .
Type/Method/Call Hierarchy	Ctrl+H / Ctrl+Shift+H / Ctrl+Alt+H	Choose these commands to navigate using the hierarchy views. Refer to the sections Viewing Structure and Hierarchy of the Source Code .
Next/Previous Highlighted Error	F2 Shift+F2	Choose these commands to navigate through the highlighted errors .
Next/Previous Change	Ctrl+Shift+Alt+Down Ctrl+Shift+Alt+Up	Choose these commands to navigate though the change markers (when VCS integration is enabled).
Next/Previous Method	Alt+Down Alt+Up	Choose these commands to go up and down through the methods and tags .

Code

Menu Keyboard Description item shortcut

Override Methods...	Ctrl+O	Choose this command to override a method. See Overriding Methods of a Superclass .
Implement Methods...	Ctrl+I	Choose this command to implement a method. See Implementing Methods of an Interface .
Generate...	Alt+Insert	Choose this command to create a new element. See (depending on the context) Populating Projects, Generating Code .
Surround With...	Ctrl+Alt+T	Choose this command to surround a logical fragment with code construct .
Unwrap/Remove...	Ctrl+Shift+Delete	Choose this command to unwrap an expression from enclosing statements .
Completion		Point to this node to reveal the nested auto-completion commands.
Folding		Point to this node to reveal the nested folding commands.
Insert Live Template...	Ctrl+J	Choose this command to create code constructs by live templates .
Surround with Live Template...	Ctrl+Alt+J	Choose this command to create code constructs using surround templates .

Comment with Line Comment	<code>Ctrl+Slash</code>	Choose this command to comment an entire line of code. See Commenting and Uncommenting Blocks of Code .
Comment with Block Comment	<code>Ctrl+Shift+Slash</code>	Choose this command to comment out a block of code. See Commenting and Uncommenting Blocks of Code .
Reformat Code...	<code>Ctrl+Alt+L</code>	Choose this command to perform code reformatting. See Reformatting Source Code .
Auto-Indent Lines	<code>Ctrl+Alt+I</code>	Choose this command to change indentation .
Optimize Imports...	<code>Ctrl+Alt+O</code>	Choose this command to optimize import statements. See Optimizing Imports .
Rearrange Code		Choose this command to rearrange code according to the arrangement rules. This action is not supported for Python.
Move Statement Up/Down	<code>Ctrl+Shift+Up</code> / <code>Ctrl+Shift+Down</code>	Choose this command to move a statement up or down .
Move Element Left/Right	<code>Ctrl+Shift+Alt+Left</code> / <code>Ctrl+Shift+Alt+Right</code>	Choose this command to move element at caret left or right.
Move Line Up/Down	<code>Shift+Alt+Up</code> / <code>Shift+Alt+Down</code>	Choose this command to move a line at caret up or down .
Inspect Code...		Choose this command to run an inspection .
Code Cleanup		Choose this command to open the dialog Specify Code Cleanup Scope Dialog .
Run Inspection by Name...	<code>Ctrl+Shift+Alt+I</code>	Choose this command to run the specified inspection .
Configure Current File Analysis...	<code>Ctrl+Shift+Alt+H</code>	Choose this command to change highlighting level of the current file .
View Offline Inspection Results...		Choose this command to see inspection results stored on your computer. See Viewing Offline Inspections Results .
Locate Duplicates...		Choose this command to find code duplicates. Refer to Analyzing Duplicates .

Refactor

Note that the composition of this menu item depends upon the current context.

Menu item	Keyboard shortcut	Description
Refactor This...	<code>Ctrl+Shift+Alt+T</code>	Choose this command to open a popup menu of the refactorings available in the current context. Refer to the section Refactoring Source Code .
Rename...	<code>Shift+F6</code>	Choose this command to rename an element .
Change Signature...	<code>Ctrl+F6</code>	Choose this command to perform the change signature refactoring. See Change Signature and Change Signature in JavaScript for details.
Move...	<code>F6</code>	Choose this command to move a symbol to the specified location.
Copy...	<code>Ctrl+C</code>	Choose this command to create a copy of an element in the specified location. See Copy for details.
Safe Delete	<code>Alt+Delete</code>	Choose this command to delete a symbol , performing search for its usages.
Extract		Choose this command to perform one of the extract refactorings. See Extract Refactorings for details.
Inline...	<code>Ctrl+Alt+N</code>	Choose this command to perform inline refactoring .
Pull Members Up...		Choose this command to perform pull members up refactoring .
Push Members Down		Choose this command to perform push members down refactoring .
Invert Boolean		Choose this command to perform invert boolean refactoring .

Run

Menu item	Keyboard shortcut	Description
Run <current run/debug configuration>	<code>Shift+F10</code>	Choose this command to run the current script with the corresponding temporary run/debug configuration. This command is duplicated with  icon on the main toolbar.
Debug <current run/debug configuration>	<code>Shift+F9</code>	Choose this command to debug the current script with the corresponding temporary run/debug configuration. This command is duplicated with  icon on the main toolbar.
Run <current run/debug configuration> with Coverage		Choose this command to run with coverage the current script with the corresponding temporary run/debug configuration. This command is duplicated with  icon on the main toolbar.
Profile <current run/debug configuration>		Choose this command to perform profiling of the current script. This command is duplicated by  icon on the main toolbar.
Concurrency Diagram for <current script>		Choose this command to explore the multi-threaded applications. Refer to the section Thread Concurrency Visualization for details. This command is duplicated by  icon on the main toolbar.

Run...	Shift+Alt+F10	Choose this command to select the desired run/debug configuration, and then launch it. Refer to the section Creating and Editing Run/Debug Configurations .
Debug...	Shift+Alt+F9	Choose this command to select the desired run/debug configuration, and then launch it in debugging mode. Refer to the section Creating and Editing Run/Debug Configurations .
Attach to local process		Choose this command to attach to a Python script. Refer to the section Attaching to Local Process for details.
Edit Configurations...		Choose this command to change run/debug configurations .
Import Test Results		Choose this command to import test results from a file.
Stop	Ctrl+F2	Choose this command to terminate execution of a run/debug configuration . This command is duplicated with  icon in the toolboxes of the Run and Debug tool windows.
Show Running List		Choose this command to display a popup that lists all currently running/debugging applications. Refer to the section Viewing Running Processes .
Stepping Commands		These commands become enabled with the debugger session on. Refer to the section Stepping Through the Program . See also descriptions of the stepping toolbar buttons in the Debug tool window reference.
Pause Program		Choose this command to pause output of the current run or debug session. This command is duplicated with  icon in the toolboxes of the Run and Debug tool windows. Note that the button is not available for Run/Debug Configuration: Node JS , Run/Debug Configuration: Node JS Remote Debug , and Run/Debug Configuration: NodeUnit .
Resume Program	F9	Choose this command to resume the debugger session with the selected run/debug configuration. This command is duplicated with  icon in the toolbox of the Debug tool windows.
Evaluate Expression	Alt+F8	Choose this command to evaluate expression during the debug session.
Quick Evaluate Expression	Ctrl+Alt+F8	Choose this command to perform quick evaluation of an expression in the editor during the debug session.
Show Execution Point	Alt+F10	Choose this command to show execution point during the debug session.
Toggle Line Breakpoint	Ctrl+F8	Choose this command to turn on or off a line breakpoint. Refer to the section Creating Line Breakpoints .
Toggle Temporary Line Breakpoint	Ctrl+Shift+Alt+F8	Choose this command to turn on or off a temporary line breakpoint. Refer to the section Creating Line Breakpoints .
View Breakpoints...	Ctrl+Shift+F8	Choose this command to show all available breakpoints and change them in the Breakpoints dialog .

Tools

Note that composition of the menu Tools depends on the enabled [plugins](#) and [external tools](#).

MenuKeyboardDescription item shortcut

Tasks and Contexts		Point to this node to reveal the sub-menu of commands related to tasks and contexts management . See also Tasks reference page.
Save File as Template		Choose this command to save the current file as a template file .
IDE Scripting Console		Choose this command to launch the interactive scripting console.
Analyze Stacktrace...		Choose this command to analyze external stacktrace .
Capture Memory Snapshot		Choose this command to get the memory state of the profiled application.
Python Console		Choose this command to launch the Python interactive console.
Create setup.py / Run setup.py		Use these commands to create and run <code>setup.py</code> .
Show Code Coverage Data	Ctrl+Alt+F6	Choose this command to view coverage results . See Code Coverage for details.
Vim Emulator		Select this check command to enable or disable Vim emulation. This command only appears when Vim plugin is installed and enabled. Refer to the tutorial Configuring PyCharm to work as a Vim editor .
Reconfigure Vim Keymap		This command is only visible, when Vim Emulator is checked. Choose it to select a different base keymap for the Vim emulator.
Deployment		Point to this node to reveal the sub-menu of deployment-related commands. Refer to the section Deployment. Working with Web Servers .
Open terminal		Choose this command to run the embedded local terminal .
Start SSH Session		Choose this command to launch a terminal on a remote SSH server. Refer to the section Running SSH Terminal .
Test RESTful Web Service		Choose this command to compose and run requests to a RESTful web service. Refer to the section Testing RESTful Web Services .
Vagrant		Point to this node to reveal the sub-menu of standard Vagrant actions. Refer to the section Vagrant: Working with Reproducible Development Environments , and to the tutorial Configuring PyCharm to work on a VM .

VCS

Note that the VCS menu contains different commands, depending on the enabled version control system. The following table shows the menu commands available when no version control integration is enabled.

MenuKeyboardDescription item shortcut

Local History	Point to this node to reveal the list of commands related to working with the Local History .
Enable Version Control Integration...	Choose this command to associate a project root with one of the supported version control systems.
VCS Operations Popup	Alt+Back Quote Choose this command to invoke the popup list of the most popular VCS actions .
Apply Patch...	Choose this command to apply a patch .
Checkout from Version Control	Point to this node to reveal the sub-menu of the checkout commands, specific for the supported version control systems. With no version control integration enabled, it is possible to check out from SVN , Mercurial , Git , GitHub , and CVS .
Import into Version Control	Point to this node to reveal the sub-menu of the import commands, specific for the supported version control systems. With no version control integration enabled, it is possible to import to SVN , Mercurial , Git , GitHub , and CVS .
Browse VCS Repository	Point to this node to reveal the sub-menu of the browse commands, specific for the supported version control systems. With no version control integration enabled, it is possible to browse Subversion , Git , and CVS repositories that are not associated with the currently opened project. <ul style="list-style-type: none"> – Browse CVS Repository: when you choose this option PyCharm opens the Select CVS Root Configuration dialog, where you can select the relevant CVS root, see Configuring CVS Roots and Browsing CVS Repository. – Browse Git Repository Log: choose this option to view the log for a local Git repository that is associated with another project. When you select the relevant repository in the Select Path dialog box, PyCharm adds a new Log tab to the Version Control tool window and shows the log for the selected repository. The name of the project associated with the selected repository is displayed in the tab title, when you hover the mouse over this tab, the full path to the repository is shown in a tooltip. – Browse Subversion Repository: when you choose this option PyCharm opens the SVN Repositories tool window where you can view, add, and edit location of SVN repositories, see Browsing Subversion Repository and Browsing Contents of the Repository.

It is important to note that with VCS integration enabled, the composition of the VCS menu is different. Refer to the following help sections for details:

- [Version Control concepts](#)
- [Common VCS procedures](#)
- [VCS-specific procedures](#)
- [Version Control reference](#)

Window

MenuKeyboardDescription item shortcut

Store Current Layout as Default	Choose this command to save the current way the tool windows are arranged .
Restore Default Layout	Shift+F12 Choose this command to restore the initial way the tool windows are arranged .
Active Tool Window	Choose this command to reveal the sub-menu of commands, related to the active tool window . Refer to the sections PyCharm Tool Windows , Manipulating the Tool Windows , Specifying the Appearance Settings for Tool Windows , Viewing Modes .
Editor Tabs	Choose this command to reveal the sub-menu of commands, related to the editor tabs. Refer to the section Managing Editor Tabs . Note that these commands can be also found on the context menu of an editor tab.
Background Tasks	Choose this command to reveal the sub-menu of commands, related to performing tasks in background. Refer to the section Working with Background Tasks .
Next Project Window / Previous Project Window	Ctrl+Alt+Close Bracket Choose this command to switch between currently opened projects. See Switching Between Open Projects . Ctrl+Alt+Open Bracket
<project>	Select project to be shown in the active window. See Switching Between Open Projects .

Help

MenuKeyboardDescription item shortcut

Find Action	Ctrl+Shift+A Choose this command to invoke an action by its name .
Keymap Reference	Choose this command to see the PyCharm shortcuts map in PDF format.
Demos and Screencasts	Choose this command to see the PyCharm demo videos and screencasts on YouTube .
Help	Choose this command to visit PyCharm online Help topics.
Tip of the Day	Choose this command to show an arbitrary tip. Refer to the section Using Tips of the Day .
Productivity Guide	Choose this command to show productivity guide .
Submit Feedback	Choose this command to report your overall impression of PyCharm to the support service. Refer to the section Reporting Issues and Sharing Your Feedback .
Show Log in Explorer/Finder	Choose this command to find PyCharm's log. Refer to the section Reporting Issues and Sharing Your Feedback for details.

Edit Custom Properties	Choose this command to open the custom file <code>idea.properties</code> , located under the user home. If this file does not exist, PyCharm suggests to create it. Refer to the section Tuning PyCharm for details.
Edit Custom VM Options	Choose this command to open the custom file <code>*.vmoptions</code> , located under the user home. If this file does not exist, PyCharm suggests to create it. Refer to the section Tuning PyCharm for details.
Debug Log Settings	Choose this command to change logging level for a category. Choosing this command leads to opening the Custom Debug Log Configuration dialog box, where you have to type the log categories names, separated with new lines. Refer to the section Reporting Issues and Sharing Your Feedback .
Developer Community	Choose this command to open the PyCharm community page .
Register...	Choose this command to register PyCharm.
Check for Updates...	Choose this command to obtain information about the current version, and the availability of newer versions of PyCharm. Refer to Updates page. This command is available on Windows/Linux. On Mac OS it appears on the PyCharm menu.
About	Choose this command to obtain information about the current version of PyCharm, current build, etc. Press <code>Escape</code> to close the popup window. This command is available on Windows/Linux. On Mac OS it appears on the PyCharm menu.

Color-Deficiency Adjustment

In this section:

- [Light editor schemes](#)
- [Darcula scheme](#)
- [Test runner adjustment](#)

Light editor schemes

To people with red-green color deficiency, green, red and their hues might look the same. In the default color scheme, red is reserved for the errors and green for the strings:

```
@Nonnull private static final String PROJECT_DEFAULT_PROFILE_NAME = "Project Default";

public DefaultProjectProfileManager(@Nonnull final Project project,
                                   @Nonnull ApplicationProfileManager applicationProfileManager,
                                   @Nonnull DependencyValidationManager holder) {
    myProject = project;
    myHolder = holder;
    myApplicationProfileManager = applicationProfileManager;
}


```

The simulation below shows how the same code fragment will look for a person with green color deficiency. Strings, annotations and unknown symbols are all the same color. The wavy error underline is lighter and less noticeable:

```
@Nonnull private static final String PROJECT_DEFAULT_PROFILE_NAME = "Project Default";

public DefaultProjectProfileManager(@Nonnull final Project project,
                                   @Nonnull ApplicationProfileManager applicationProfileManager,
                                   @Nonnull DependencyValidationManager holder) {
    myProject = project;
    myHolder = holder;
    myApplicationProfileManager = applicationProfileManager;
}


```

The Adjust for color deficiency option changes colors so that they can be differentiated by a person with green or red color deficiency. Strings and annotations are shades of blue, and orange is reserved for the error states:

```
@Nonnull private static final String PROJECT_DEFAULT_PROFILE_NAME = "Project Default";

public DefaultProjectProfileManager(@Nonnull final Project project,
                                   @Nonnull ApplicationProfileManager applicationProfileManager,
                                   @Nonnull DependencyValidationManager holder) {
    myProject = project;
    myHolder = holder;
    myApplicationProfileManager = applicationProfileManager;
}


```

Then, with the option Adjust for color deficiency turned on, and green color deficiency is selected, the simulation for the green color deficiency will look like the following:

```
@Nonnull private static final String PROJECT_DEFAULT_PROFILE_NAME = "Project Default";

public DefaultProjectProfileManager(@Nonnull final Project project,
                                   @Nonnull ApplicationProfileManager applicationProfileManager,
                                   @Nonnull DependencyValidationManager holder) {
    myProject = project;
    myHolder = holder;
    myApplicationProfileManager = applicationProfileManager;
}


```

Highlight for read/write states of identifiers at caret are well distinguished for the persons without color deficiency:

```
Class callerClass = null;
for (final Object o : element.getChildren()) {
    Element listElement = (Element)o;
    if (ATTR_LIST.equals(listElement.getName())) {
        if (callerClass == null) {
            callerClass = ReflectionUtil.findCallerClass(2);
            assert callerClass != null;
        }
    }
}


```

However, these states become indistinguishable for the persons with color deficiency:

```
Class callerClass = null;
for (final Object o : element.getChildren()) {
    Element listElement = (Element)o;
    if (ATTR_LIST.equals(listElement.getName())) {
        if (callerClass == null) {
            callerClass = ReflectionUtil.findCallerClass(2);
            assert callerClass != null;
        }
    }
}


```

With the Adjust for color deficiency option, the difference between read and write states remains visible. In the example below you see the green color deficiency simulation:

```
Class callerClass = null;
for (final Object o : element.getChildren()) {
    Element listElement = (Element)o;
    if (ATTR_LIST.equals(listElement.getName())) {
        if (callerClass == null) {
            callerClass = ReflectionUtil.findCallerClass(2);
            assert callerClass != null;
        }
    }
}


```

Folded text highlight in the default scheme is easily noticeable for the persons without color deficiency:

```
String ABSTRACTION_GROUP_NAME = InspectionsBundle.message("group.names.abstraction.issues");
String ASSIGNMENT_GROUP_NAME = "Assignment issues";
String BUGS_GROUP_NAME = "Probable bugs";


```

But this is a challenge for the persons with color deficiency - it is too light to notice:

```
String ABSTRACTION_GROUP_NAME = InspectionsBundle.message("group.names.abstraction.issues");
String ASSIGNMENT_GROUP_NAME = "Assignment issues";
String BUGS_GROUP_NAME = "Probable bugs";


```

With the Adjust for color deficiency option it becomes noticeable:

```
String ABSTRACTION_GROUP_NAME = InspectionsBundle.message("group.names.abstraction.issues");
String ASSIGNMENT_GROUP_NAME = "Assignment issues";
String BUGS_GROUP_NAME = "Probable bugs";
```

Darcula scheme

The non-adjusted Darcula text looks as follows:

```
@Nonnull private static final String PROJECT_DEFAULT_PROFILE_NAME = "Project Default";

public DefaultProjectProfileManager(@Nonnull final Project project,
    @Nonnull ApplicationProfileManager applicationProfileManager,
    @Nonnull DependencyValidationManager holder) {
    myProject = project;
    myHolder = holder;
    myApplicationProfileManager = applicationProfileManager;
}.
```

The difference between the various elements of text is not noticeable for the persons with color deficiency.

```
@Nonnull private static final String PROJECT_DEFAULT_PROFILE_NAME = "Project Default";

public DefaultProjectProfileManager(@Nonnull final Project project,
    @Nonnull ApplicationProfileManager applicationProfileManager,
    @Nonnull DependencyValidationManager holder) {
    myProject = project;
    myHolder = holder;
    myApplicationProfileManager = applicationProfileManager;
}.
```

Now turn on the option Adjust for color deficiency.

Compare the view of the editor for the persons without color deficiency:

```
@Nonnull private static final String PROJECT_DEFAULT_PROFILE_NAME = "Project Default";

public DefaultProjectProfileManager(@Nonnull final Project project,
    @Nonnull ApplicationProfileManager applicationProfileManager,
    @Nonnull DependencyValidationManager holder) {
    myProject = project;
    myHolder = holder;
    myApplicationProfileManager = applicationProfileManager;
}.
```

and that for the persons with color deficiency:

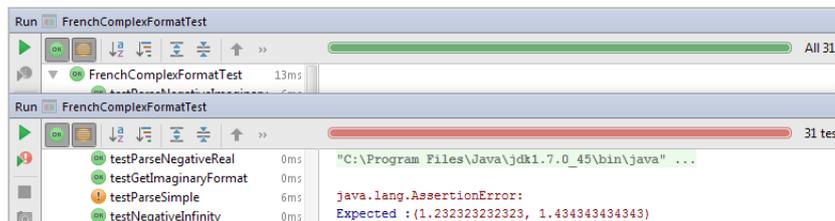
```
@Nonnull private static final String PROJECT_DEFAULT_PROFILE_NAME = "Project Default";

public DefaultProjectProfileManager(@Nonnull final Project project,
    @Nonnull ApplicationProfileManager applicationProfileManager,
    @Nonnull DependencyValidationManager holder) {
    myProject = project;
    myHolder = holder;
    myApplicationProfileManager = applicationProfileManager;
}.
```

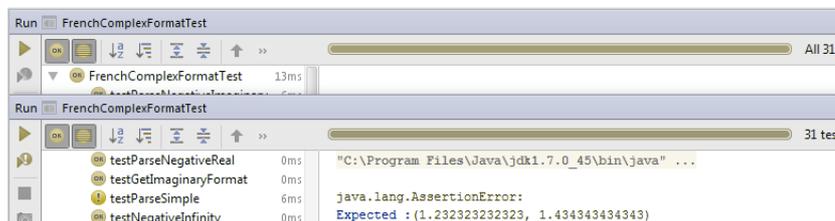
Test runner adjustment

Colors of the test runner progress have also been adjusted. The usual progress bar color is indistinguishable for a person with green or red color deficiency.

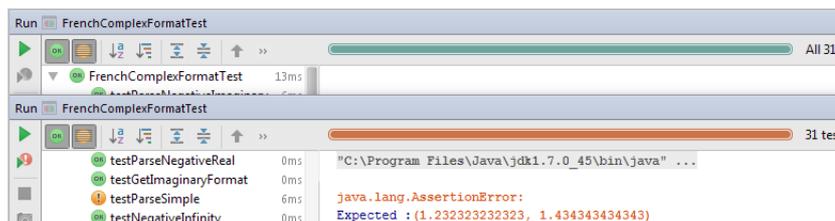
Compare the view of the test runner for the person without color deficiency:



with that for the persons with color deficiency:



Turn on the option Adjust for color deficiency, and compare the view of the test runner for the person without color deficiency:



with that for a person with color deficiency:

Run FrenchComplexFormatTest

FrenchComplexFormatTest 13ms

Run FrenchComplexFormatTest 31 test

Test Method	Duration	Output
testParseNegativeReal	0ms	"C:\Program Files\Java\jdk1.7.0_45\bin\java" ...
testGetImaginaryFormat	0ms	
testParseSimple	6ms	java.lang.AssertionError:
testNeqativeInfinity	0ms	Expected : (1.232323232323, 1.4343434343)

Working with PyCharm Features from Command Line

In this section:

- Working with PyCharm Features from Command Line
- [Command Line Differences Viewer](#)
- [Command Line Code Inspector](#)
- [Command Line Formatter](#)
- [Opening Files from Command Line](#)

Besides working from within PyCharm, it is possible to perform certain actions "offline", without actually launching the IDE.

This way you can:

- Inspect code
- View differences
- Open files
- Format files

Command Line Differences Viewer

On this page:

- [Viewing differences](#)
- [Examples](#)
- [Windows](#)
- [macOS](#)

Viewing differences

To view differences using command line diff tool

- In the command line, type the following:

```
<PyCharm> diff <path1> <path2>
```

where:

- <PyCharm> is the platform-specific product launcher
- <path1>, <path2> are full paths to the files to be compared.

Examples

Windows

```
PyCharm.exe diff C:\SamplesProjects\MetersToInchesConverter\src\javascript\numbers.js  
C:\SamplesProjects\MetersToInchesConverter\src\coffeescript\numbers.coffee
```

macOS

```
/Applications/PyCharm.app/Contents/MacOS/pycharm diff ~/Documents/file1.txt ~/Documents/file2.txt
```

Command Line Code Inspector

On this page:

- [Launching a code inspection from the command line](#)
 - [Examples](#)
 - [Windows](#)
 - [macOS](#)
- [Viewing the results of an offline inspection](#)

Launching a code inspection from the command line

To launch a code inspection from the command line

- Specify the following command line arguments:
 - Path to the launcher: specify the **full path** to one of the following launchers (which reside under the `bin` directory of your PyCharm installation):
 - For **Windows**: `inspect.bat`
 - For **UNIX** and **macOS**: `inspect.sh`
 - Project file path is the **full path** to the directory that contains the project to be inspected.
 - Inspection profile path is the **full path** to the profile, against which the project should be inspected. The inspection profiles are stored under `USER_HOME\PyCharmXX\config\inspection`
 - Output path is the **full path** to an existing directory where the report will be stored.
 - Options. You can specify:
 - The directory to be inspected `-d <full path to the subdirectory>`
 - The verbosity level of output `-vX`, where X is 0 for quiet, 1 for noisy and 2 for extra noisy.

Warning! Please note that you have to specify full paths. Relative paths are not accepted!

Examples

Windows

```
C:\Program Files (x86)\JetBrains\PyCharm home\bin\inspect.bat
E:\SampleProjects\MetersToInchesConverter E:\Work\MyProject\.idea\inspectionProfiles\Project_Default.xml
E:\Work\MyProject\inspection-results-dir -v2 -d E:\SampleProjects\MetersToInchesConverter\subdirectory
```

macOS

```
/Applications/PyCharm.app/Contents/bin/inspect.sh ~/PyCharmProjects/MyTestProject
~/Library/Preferences/pycharmXX/inspection/Default.xml ~/PyCharmProjects/MyTestProject/results-dir -v2
```

Viewing the results of an offline inspection

If you have performed an offline inspection and exported the inspection results to a directory in the XML format you can always download and view these results.

To view the results of an offline inspection, follow these steps

1. Open the project against which the inspection was performed.
2. On the main menu, choose Code | View Offline Inspection Results.
3. In the Select Path dialog box that opens, navigate to the directory that contains inspection results in XML format.
4. Click OK. Inspection results display in the Offline View tab in the [Inspection Results Tool Window](#).

Tip! Alternatively, you can open the relevant XML file in PyCharm or in any other text processor without opening the inspected project.

Command Line Formatter

On this page:

- [Introduction](#)
- [Formatter launcher script](#)
 - [Usage](#)
 - [Parameters](#)
- [Example](#)

Introduction

Command-line source code formatter is a special functionality within PyCharm that lets you format arbitrary files outside of a project. So doing, the formatter makes use of XML files with the [exported code style settings](#) (Settings/Preferences | Editor | Code Style - Manage - Export Code Style XML File) and a file specification that defines a file or a group of files to be formatted.

For the files to be formatted there should be corresponding plugins which support the required file types.

Formatter launcher script

The script is `format.bat/format.sh` (Windows/Linux/Mac) in `<PyCharm_HOME>/bin` directory, where `<PyCharm_HOME>` is PyCharm's root installation directory. The script launches PyCharm, which formats the specified files and quits. If launched without any parameters or with `-h` parameter, the script outputs a list of its options.

Usage

```
format [-h] [-r|-R] [-s|-settings settingsPath] [-m|-mask masks] [path1 [path2]...]
```

Parameters

ParameterDescription

<code>-h</code>	Shows a help message and quits.
<code>-r -R</code>	Scans directories specified in <code>path1,path2...</code> recursively.
<code>-s -settings settingsPath</code>	<code>settingsPath</code> is a path to the file with the exported code style settings . Note If this parameter is omitted, the default code style settings are used.
<code>-m -mask masks</code>	A comma-separated list of file masks, which define the files to be processed. The wildcard characters <code>*</code> (any string), <code>?</code> (any single character) are supported.
<code>pathN</code>	A path to a file or a directory to be processed.

Example

1. Format all the files in `C:\Data\src` directory, including all subdirectories, using the default code style settings:

```
format -r C:\Data\src
```

2. Non-recursively format all `.java` and `.html` files in `C:\Data\src` directory, using code style settings from `C:\Data\settings.xml`:

```
format -s C:\Data\settings.xml -m * .java,*.html C:\Data\src
```

Opening Files from Command Line

On this page:

- [Opening a file in the editor](#)
- [Examples](#)
 - [Windows](#)
 - [macOS](#)

Opening a file in the editor

PyCharm helps opening a file for editing so that the caret rests at the specified line.

To open a file for editing

- In the command line, type the following:

```
<PyCharm> <path1> --line <number> <path2>
```

where:

- `<PyCharm>` is the platform-specific product launcher
- `<path1>` is the path to the project that contains the desired file
- `<number>` is the number of the line, where the caret should rest
- `<path2>` is the path to the file to be opened

Examples

Windows

```
PyCharm.exe C:\SamplesProjects\MetersToInchesConverter --line 3  
C:\SamplesProjects\MetersToInchesConverter\src\javascript\numbers.js
```

macOS

```
/Applications/PyCharm.app/Contents/MacOS/pycharm ~/PyCharmProjects/untitled45 --line 1 ~/PyCharmProjects/untitled45/sample.sass
```

Sending Feedback

Your feedback, including error reports, improvement suggestions, new feature requests and any other things you might have to say to JetBrains team, is welcome at the addresses listed below.

- [JetBrains support](#)
- [Community support](#)