

Exploring the IDE. Quick Start Guide

Welcome to PyCharm! This short guide aims to help you get a grip on the PyCharm IDE.

[Before you start...](#) | [Initial setup](#) | [Setting defaults from the Welcome screen](#) | [Project](#) | [Types of projects](#) | [Start er](#) | [Customize everything for your project!](#) | [Project interpreter](#) | [Local interpreter](#) | [Remote interpreter](#) | [Virtual environment](#) | [Packages and paths](#) | [VCS](#) | [File colors](#) | [IDE and Editor](#) | [Appearance](#) | [Editor](#) | [Keymap](#) | [External editor](#) | [Background tasks](#) | [Write code smartly!](#) | [Use macros](#) | [Work with multiple files](#) | [Analyze code transparently](#) | [Create quality code](#) | [View documentation](#) | [Find your way through](#) | [Source Code](#) | [IDE Components](#) | [Finding an action](#) | [Navigating and searching everywhere](#) | [Refactoring your source code](#) | [Run, debug and test your application](#) | [Running](#) | [REPL console](#) | [Local terminal](#) | [Debugging](#) | [Testing](#) | [Remote development](#) | [Data sources and SQL support](#) | [Polyglot IDE](#) | [That's it, folks!](#)

Before you start...

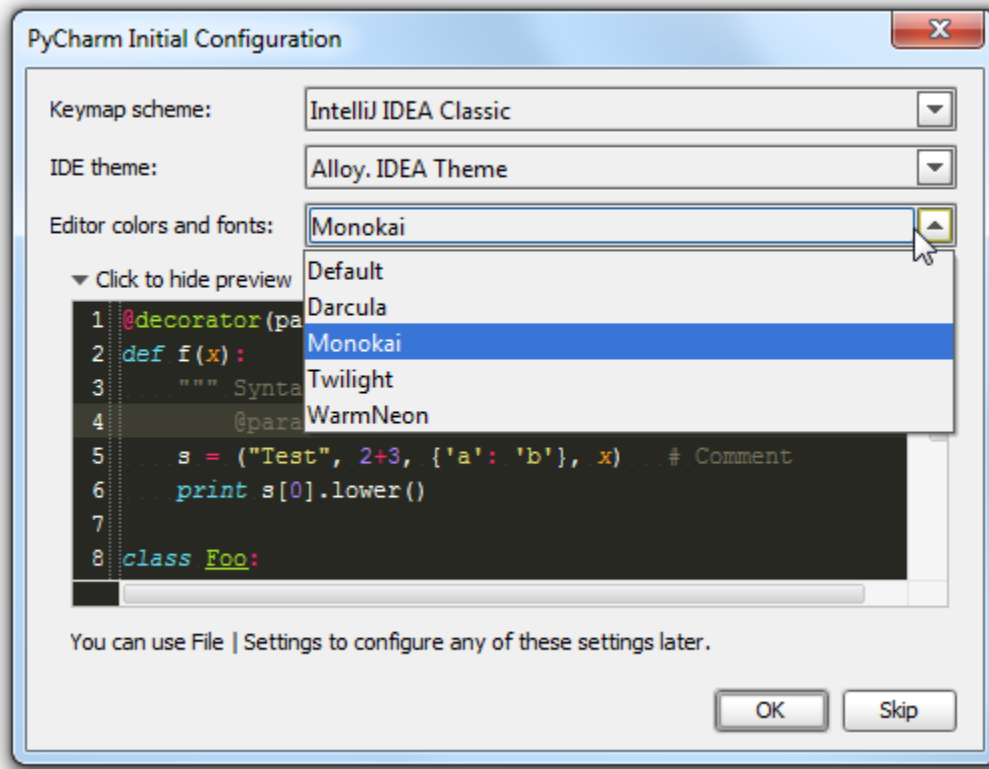
- Make sure that at least one [Python](#) interpreter, version from 2.4 to 3.4 is properly installed on your computer.
- Mind that PyCharm is available in two editions: Community and Professional. The difference between the editions is explored in [Edition Comparison Matrix](#).

Note that the [download page](#) contains installation instructions for the various platforms; these instructions may vary for the different operating systems.

Initial setup

If this is the very first time you launch your PyCharm, it will ask you several important questions:

- First, whether you already have setting you want to preserve (for example, from a previous version)
- Your license information
- And, finally, which keymap and theme you want to use:



Note that PyCharm has several pre-defined keymaps: for those who like Eclipse or Visual Studio, for the Emacs fans, GNOME, KDE and more. Explore the list of available keymaps in the [Keymap page](#) of Settings/Preferences dialog.

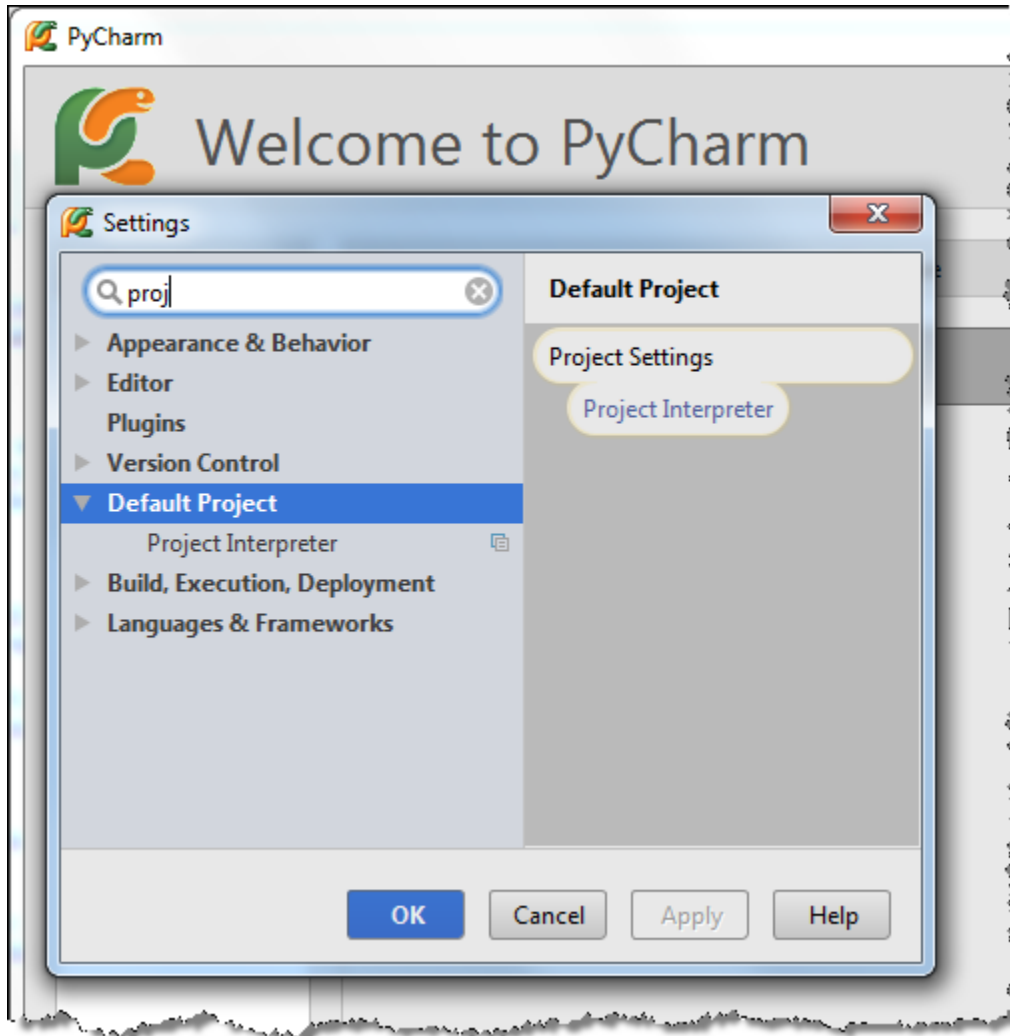
For the dedicated Vim users, PyCharm suggests [IdeaVim plugin](#) that helps coding in PyCharm as if in the Vim editor. For those who cannot live without real Emacs, PyCharm provides the possibility to [use it as an external editor](#).

If later you decide that your initial choice was wrong, you can always change your settings. To learn more about configuring the IDE theme and keymap, refer to our [documentation](#) and tutorials:

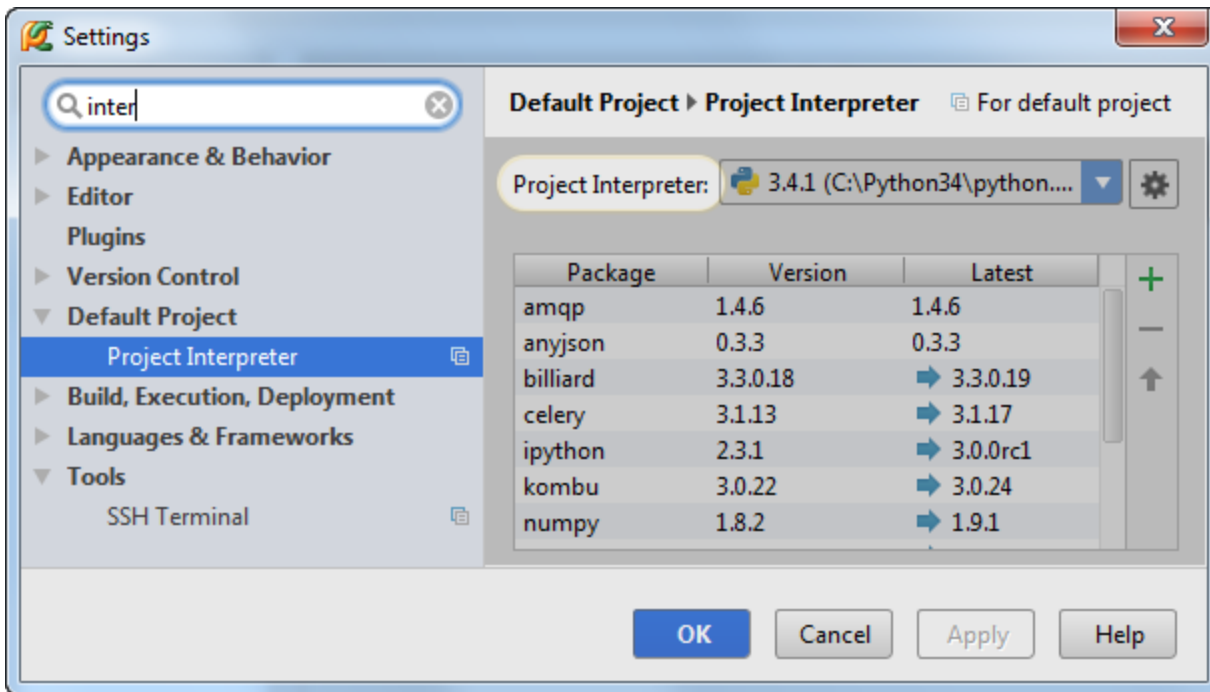
- [Getting started with PyCharm](#)
- [Configuring keyboard schemes](#)

Setting defaults from the Welcome screen

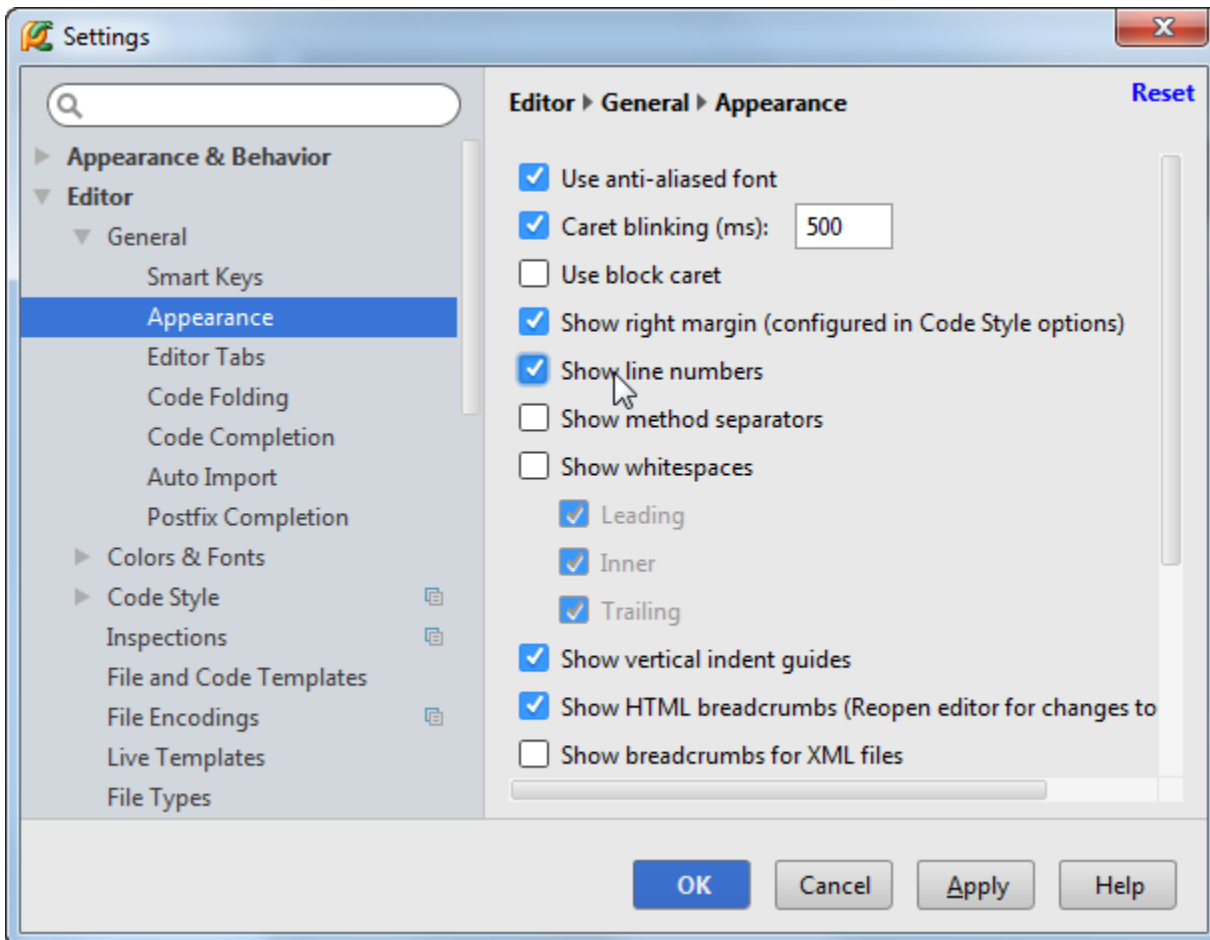
Again, if this is your very first start (or if you have no open project), you'll see the [Welcome screen](#). If you click Configure, you'll see the page of the Welcome screen that suggests you to configure your environment, plugins, import and export settings. You have to click Configure once more - and see the [Settings/Preferences dialog](#). If you look at this dialog, you'll notice the header "Default Project":



What does it mean? These settings will be used every time you create a new project. For example, you want all you newly created projects to use the same interpreter - OK, you can [define such an interpreter](#) in the Default Project settings:



The Editor settings pertain to your working environment. For example, you want your editor to always show line numbers. In the Settings/Preferences dialog, expand the node Editor, and in the [Appearance](#) page, select the check box “Show line numbers”:

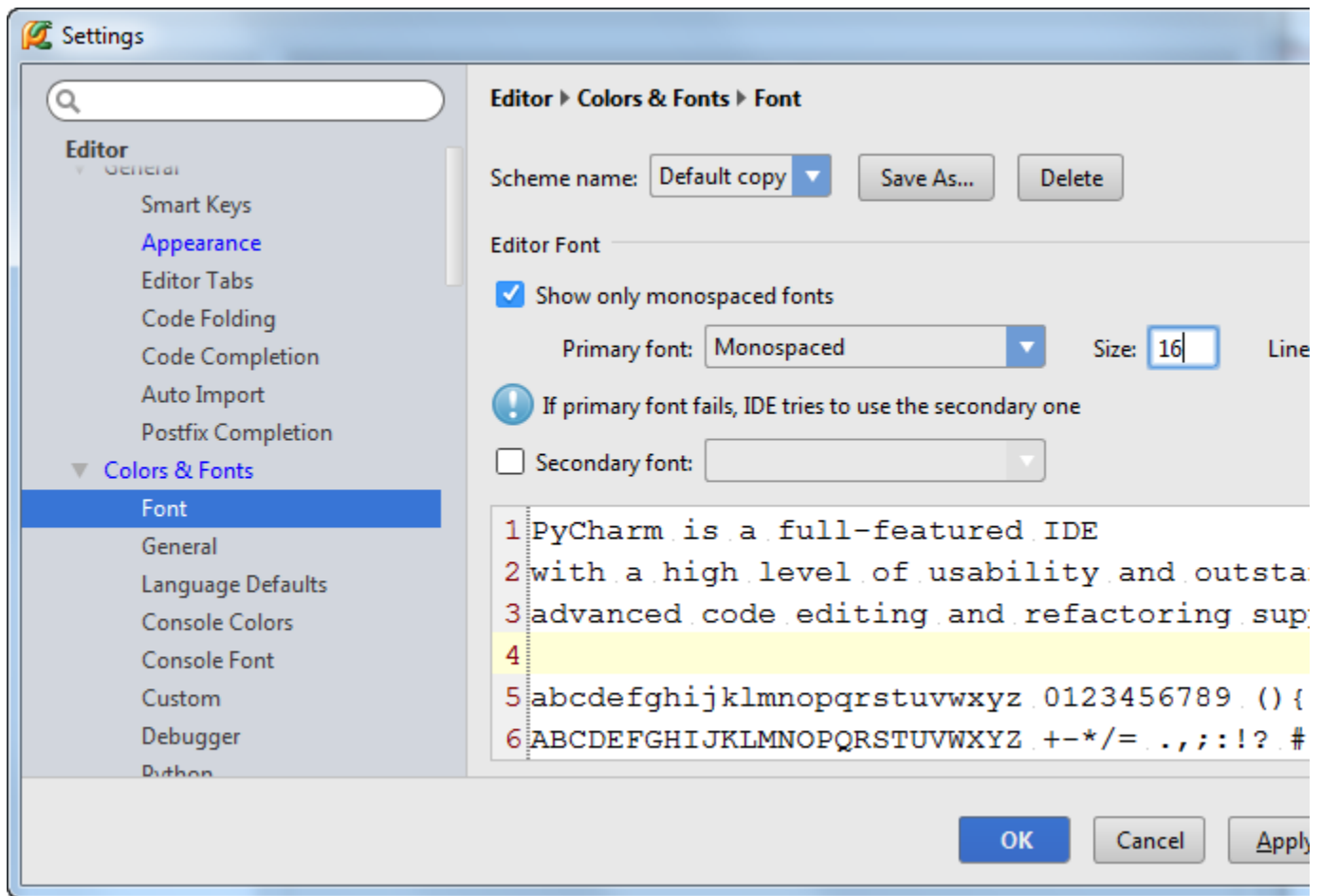


Next, suppose you want to use some particular color scheme for the editor, and you are not happy with the colors suggested by default. OK, select the base scheme, create its copy, and then change colors (the schemes suggested by default are not editable).

It is also possible to set your preferred font size for the editor - this is done in the Fonts page of the [Colors and Fonts settings](#). Again, you have to

create a copy of the scheme first, if you haven't done it already, and then specify the desired font size for the editor. This font size will apply to the current editor tabs, and all the newly opened editors. However, it does not affect the font size of the IDE components.

Note that you can always see the results of your experiments in the Preview pane:



The whole procedure is described step-by-step in the tutorial [What my PyCharm looks like](#).

When a project is already created, you can change its settings at any moment. Configuring settings for the current project will be discussed a little bit later, in the section [Customize everything for your project!](#).

Finally, you can show or hide actually all the UI elements: toolbar buttons, main menu, main toolbar (menu View); PyCharm also enables you to choose viewing mode. Refer to the documentation for details:

- [PyCharm tool windows](#)
- [Presentation and Full Screen viewing modes](#)

Project

Everything you do in PyCharm, is done within the context of a [project](#). What is most interesting about PyCharm project management, is the possibility to [open multiple projects in one frame](#). When you create a new project (File New Project), or open an existing one (File Open), PyCharm suggests you to choose which way you want the project to be opened: in a new window, in the same window after closing the previously opened project, or added to the previously opened project.

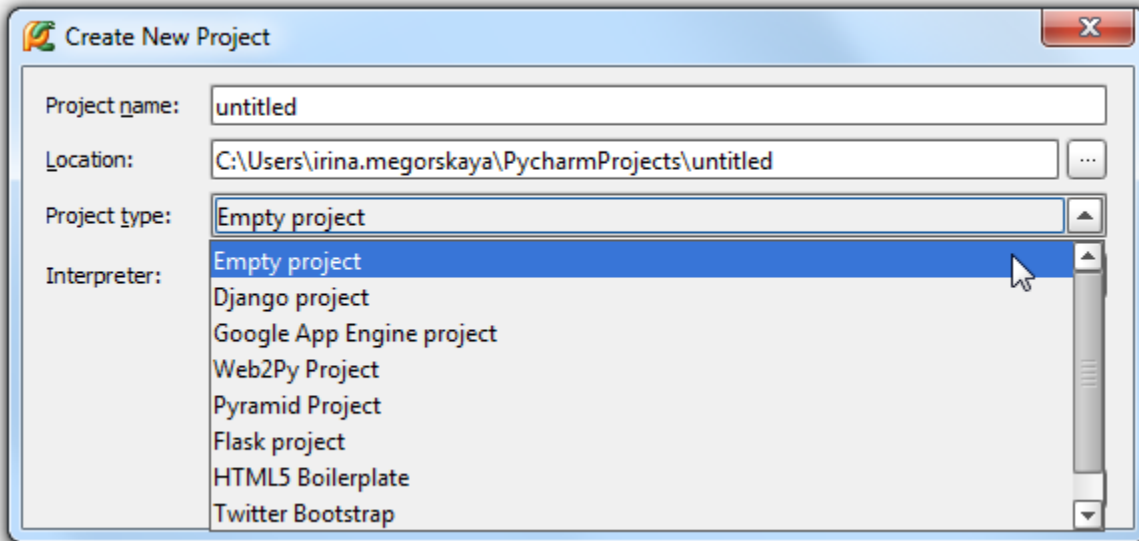
You can have as many projects opened in one window as required. So doing, the first project is considered the primary project. All the symbols of the added projects are visible from the primary project.

As an exercise, create a new empty project, where you can develop some pure Python code. This case is discussed in detail in the tutorial [Getting started with PyCharm](#) - in particular, the section [Creating a simple project](#).

Types of projects

By the way, PyCharm suggests creating projects of the various types: Django, Flask, Pyramid, web2py, etc. You can explore the available types

yourself, when creating a new project - just select the new project type from the drop-down list:

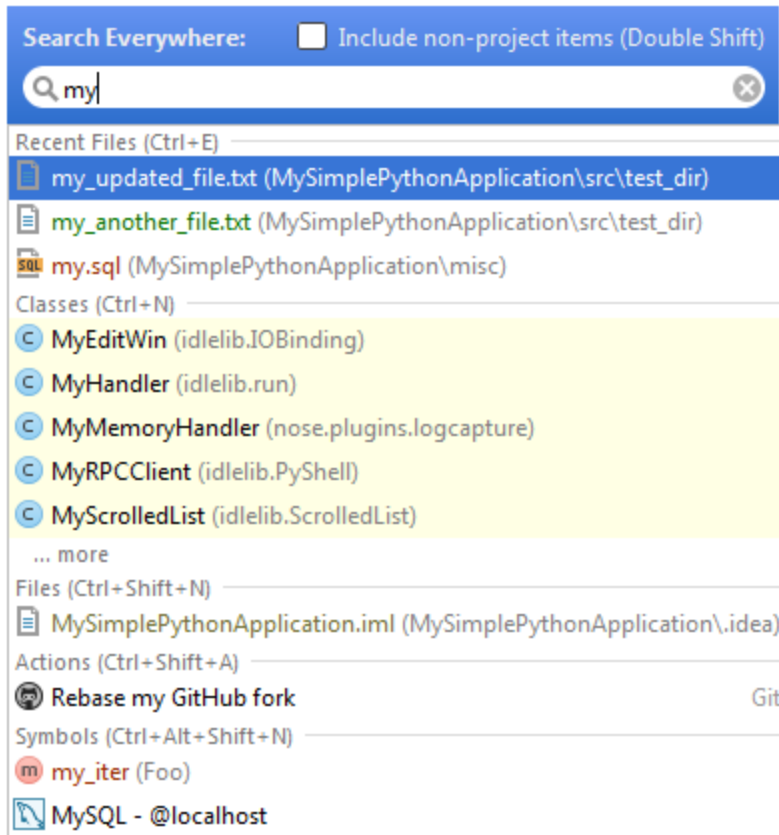


As you see, PyCharm supports all the major Python-based frameworks. For each of the supported project types, PyCharm creates the corresponding file and directory structure, and all the necessary artifacts. Refer to the product documentation:

- [Django](#)
- [Flask](#)
- [Pyramid](#)
- [Google App Engine](#)
- [Web2Py](#)


Starter

OK, your project is ready. Before you start working with it, just press Shift twice. You'll see a pop-up window that allows finding anything and jumping everywhere:



As you see, this way one can search among the actions, settings, files, IDE components, and more. Note that it is just one of the numerous navigation features. We'll return to the PyCharm's search and navigation facilities [a little bit later](#).

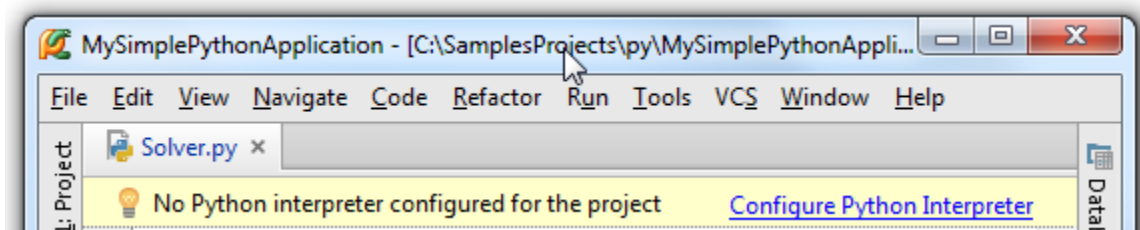
Customize everything for your project!

Look at the main toolbar — there is the Settings button . Clicking this button opens the [Settings/Preferences dialog box](#), where you can change your project structure, set up version control, and tweak your working environment to make the development process a real pleasure.

Some of the settings pertain to a particular project - for example, project interpreter, version control configuration, or file colors. The others - like the Editor settings, keymaps, or live templates - pertain to your whole working environment, and thus can be configured even without an open project.

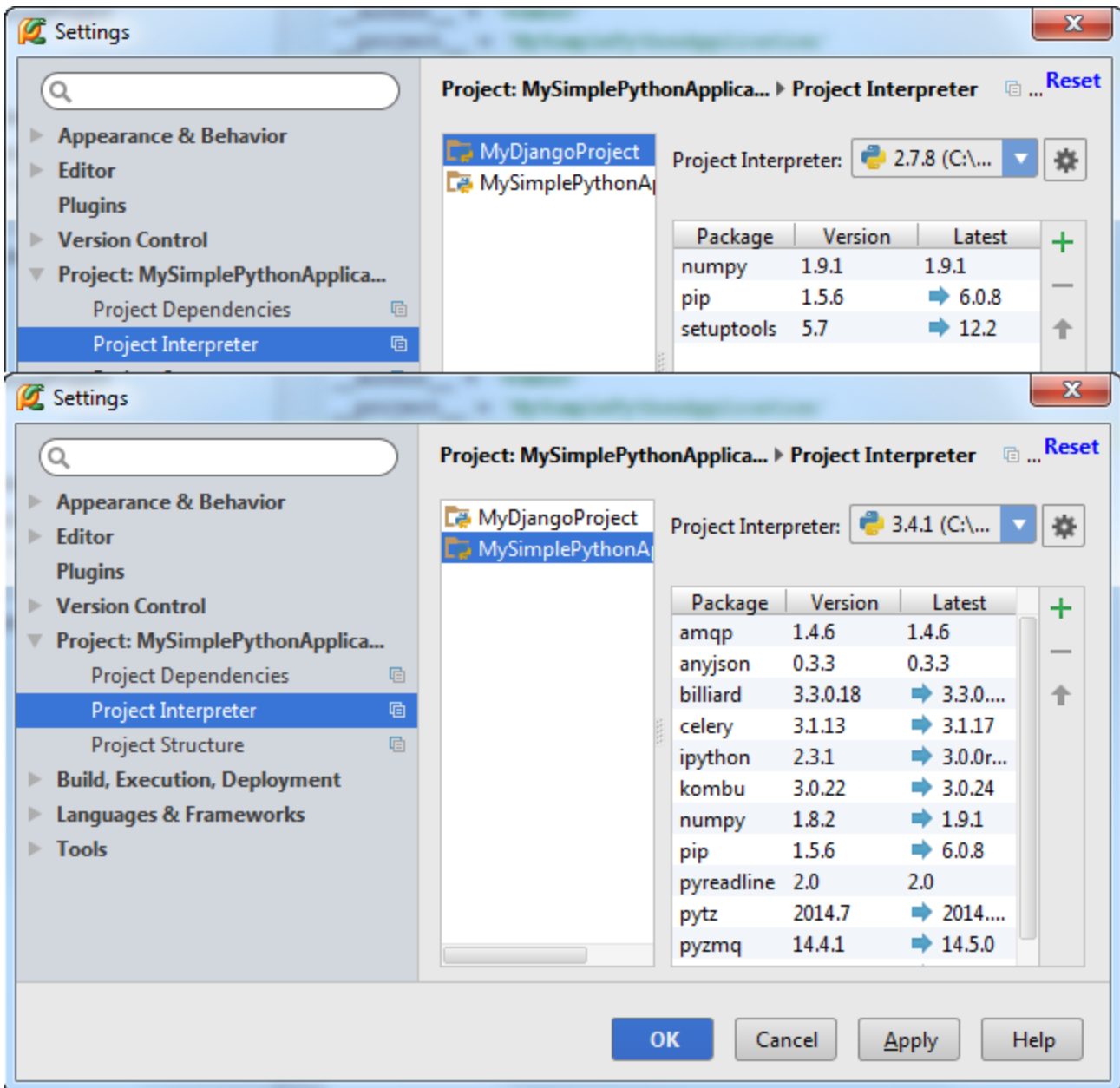
Project interpreter

Python interpreter is vital - without it you will not be able to do anything...PyCharm will warn you, if you manage to create a project without an interpreter:



In PyCharm, you can define several Python interpreters - they just comprise the list of interpreters, available on your machine. From among them, you can choose the one to be used in your project.

You need to tell PyCharm which Python interpreter you want to use since it can [use a different interpreter for each project](#):



PyCharm will use this information for indexing.

You can use Python interpreters of the following types:

- Local
- Remote
- Virtual environments

Local interpreter

This is the most straightforward way of using an interpreter. You download a Python interpreter, install it on your machine, and then specify the Python executable... Refer to the [tutorial](#) or [product documentation](#).

Remote interpreter

With PyCharm, you can use interpreters located remotely, for example, on a reliable server. So doing, PyCharm makes it possible to configure remote interpreters via [SSH connection](#), or via [Vagrant box](#).

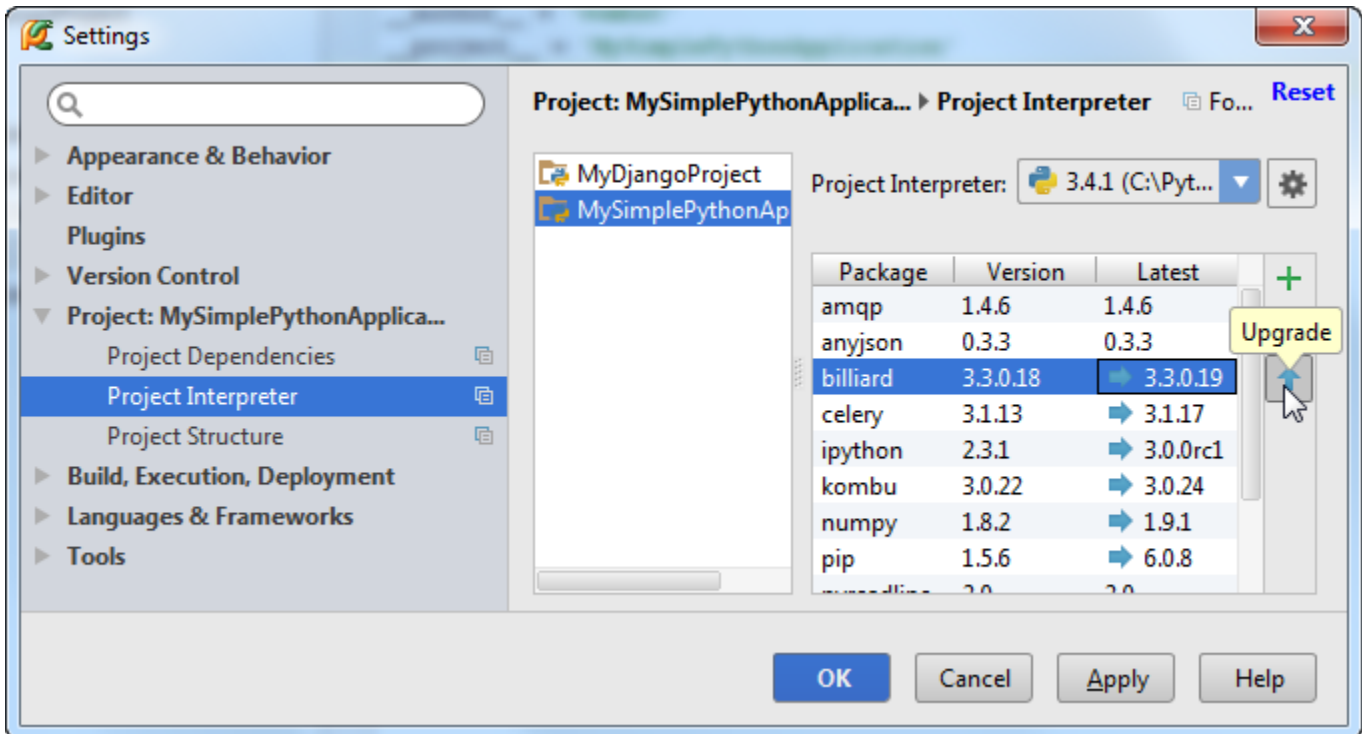
Refer to the tutorial [Configuring interpreters with PyCharm](#).




Virtual environment

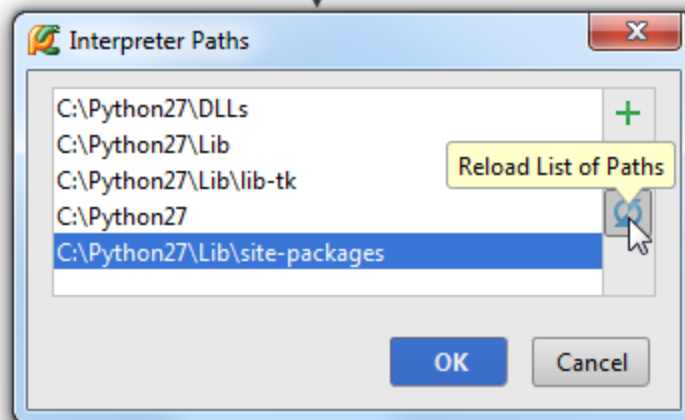
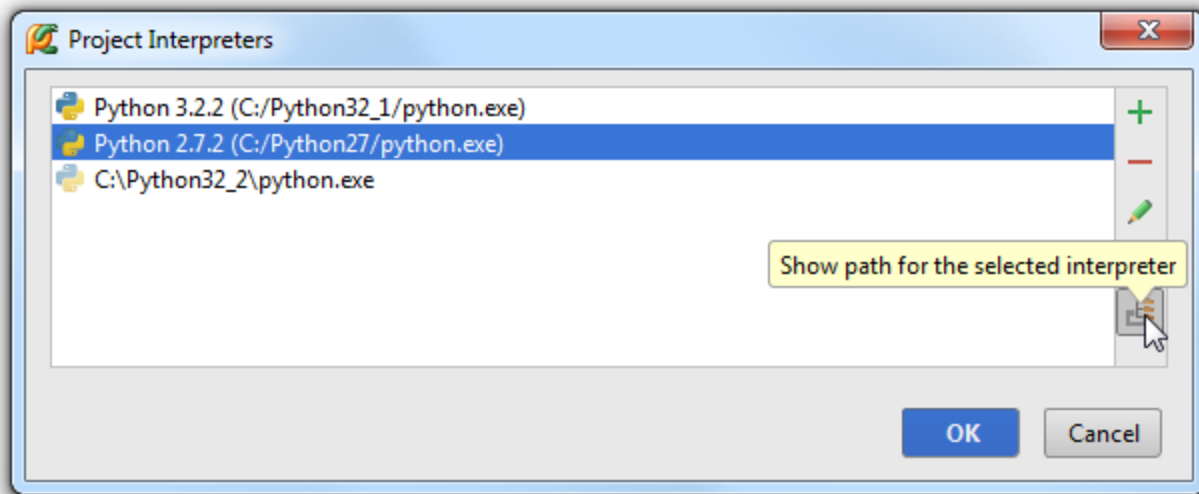
Why do we need it all? Suppose, you are working on one project that makes use of, say, Django 1.6, and at the same time support another project requiring Django 1.2. In such a case, you need something that keeps your environment safe and consistent - a tool that allows creating an isolated working copy of a Python interpreter. Learn how to create a virtual environment in the [documentation](#) and [tutorial](#).

Packages and paths

Regardless of the interpreter type, PyCharm helps install and update the necessary packages and paths. For example, when PyCharm automatically detects that some of the required packages are missing or outdated, it suggests you to install or upgrade them:



Same thing happens with the paths. If you have installed or upgraded libraries, it is a good idea to rescan the Python installation. Click the gear button  located to the right of the project interpreter field, and choose More. Then, in the Project Interpreters dialog, select the interpreter you want to view the paths for, click , and then, in the Interpreter Paths dialog, refresh packages  :



Find details in the [product documentation](#).

VCS

Surely, you keep your sources under a version control, right? Git? SVN? Mercurial? With PyCharm, it's easy to set up, and again the IDE does a good job auto-detecting the VCS already used for existing projects.

But you can fine-tune — just click Version Control node in the [Settings/Preferences dialog](#) (Project Settings/Version Control). By default, you will see project root directory only, but you can break your application down into smaller chunks, and place virtually every directory under its own version control system.

In the [Settings/Preferences dialog](#), you can also define some behaviors that are common to all version control systems: confirmation on creating or deleting files, tasks performed in the background, ignoring unversioned files and more.

Refer to the tutorials:

- [Version control basics](#)
- [Using PyCharm's Git integration locally](#)
- [Sharing via a remote repository](#)

See also the PyCharm documentation:

- [Version control with PyCharm](#)
- [Version control procedures](#)

File colors

Your project might contain several sites, each one with its own set of files with the same names (`init.py`, `models.py`, `tests.py`, `views.py`). When they are opened in the editor, it's rather confusing... How can one tell which site they belong to? PyCharm helps make them stand out by painting

their editor tabs (Settings/PreferencesProject SettingsFile Colors). Break down your project into smaller chunks - scopes (for example, a scope per site), and select a color for each one.


Refer to the page [Configuring scopes and file colors](#) for details.

IDE and Editor

Working in an IDE actually means living in it, and it is quite natural to make your haunted place pleasant for your eyes and comfortable for your fingers. That's why PyCharm makes it possible to choose look and feel of the IDE and the editor, configure your preferred set of keyboard shortcuts (keymap), fine-tune scrolling behavior, highlighting, and more.

It is recommended to familiarize yourself with the matter in the documentation:

- [Configuring project and IDE settings](#)

All these settings are (again) configurable in the [Settings/Preferences dialog](#): click  on the main toolbar, and see the list of pages. Here are three of them, which might be of interest for the starter: [Appearance](#) and [Keymap](#) under Appearance and Behavior node, and the [Editor](#) node.

Appearance

Remember, you've initially selected the look and feel for your IDE on the first start? If you are not happy with the result, now it's time to change your mind. In the page Settings/PreferencesAppearance and BehaviorAppearance you can select "look and feel" of your PyCharm installation. Just click the Look and feel drop-down, and select the scheme you like better. You don't need to close the [Settings/Preferences dialog](#) dialog to observe the results of your experiments: click Apply, see what happens, and close the dialog when you are quite happy with what you've got.

Refer to the tutorial [How do I choose look and feel for my PyCharm?](#), and to the [product documentation](#).

Editor

The whole bunch of pages under the [Editor](#) node (Settings/PreferencesEditor) helps you tune every aspect of the editor's behavior. Note that PyCharm comes with the predefined color scheme, but if you want to make up something very personalized, you are welcome to do it: save the default scheme with a new name, and start changing its background, font, colors of syntactical elements, error highlighting etc., and immediately observe results in the preview pane.

Refer to the tutorial [How do I change color scheme of the editor](#) and to the [product documentation](#).

Keymap

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 rather painful. With this in mind, PyCharm supplies you with a wide range of pre-defined keymaps (Settings/PreferencesAppearance and BehaviorKeymap), for those who prefer Eclipse, or for those who've had long experience with Visual Studio... You can create your very own keymap on the base of an existing one.

And finally, there is a magic shortcut Ctrl + Back Quote that helps you switch between schemes (all of them - keymaps, colors, code styles, and L&F) without the Settings dialog (for Windows and Linux users only).

Refer to the tutorial [Configuring keyboard schemes](#), and to the [product documentation](#).

External editor

Though you can choose any keymap that corresponds to your preferred editor (Emacs, Vim, TextMate, etc), you might still want to open files in your preferred editor. You can easily do it by configuring an external tool. For example, you might want to open a current file in Emacs as an external tool.



Refer to the tutorial [Using Emacs as an external editor](#) for details.

Background tasks

Sometimes, when a long task is in progress, PyCharm shows a Progress bar. You can bring such a task to the background, but still see how it goes on. Refer to page [Working with Background Tasks](#) for details.

Write code smartly!

What makes PyCharm stand out from the numerous IDEs, is its [full-featured editor](#). Whatever you do for developing your source code, PyCharm is always at hand, helping you create error-free applications. Here is a brief outline of the smart PyCharm's coding assistance:

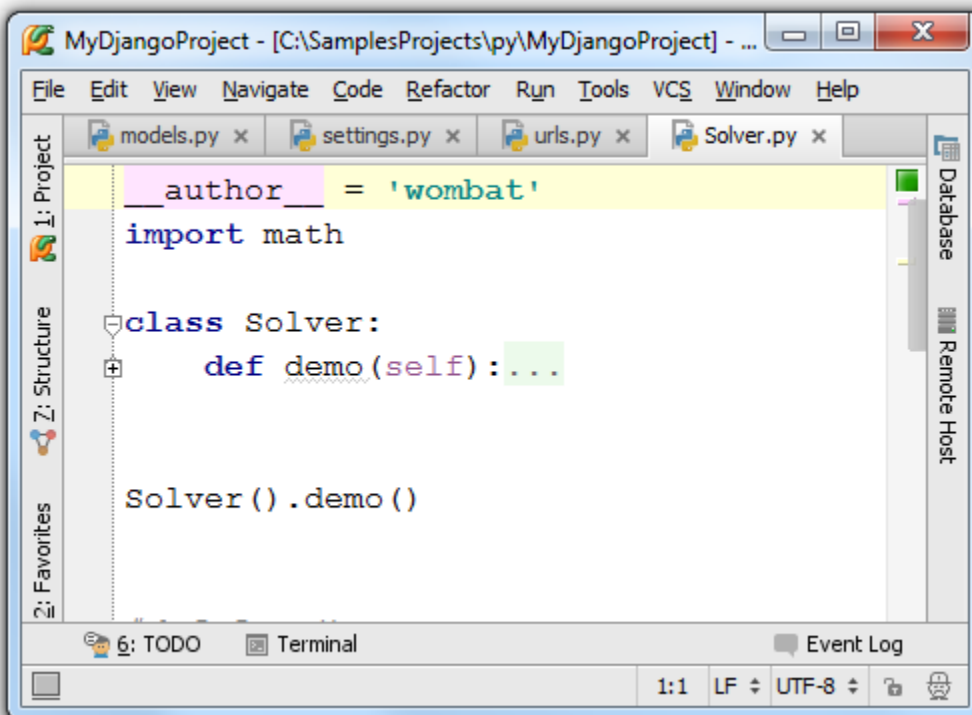
- At every stage of development, use [code completion](#) (Ctrl+Space), which takes into account the current context. For example, depending on the place where you invoke code completion, you can complete keywords or code blocks, infer types, or complete methods and properties.
Refer to the tutorial [Code completion](#) and to the [product documentation](#) for details.
- Use live templates/snippets (Ctrl+J) or surround templates (Ctrl+Alt+J) to produce entire code constructs. PyCharm comes with a wide range of ready-to-use live templates, or snippets, which you can explore in the Settings/Preferences dialog (Editor [Live templates](#)). If you see that you are lacking something especially important for your development goals, extend this set of snippets with your own ones. Don't also miss the possibility to surround with complete code constructs (Ctrl+Alt+T). Refer to the tutorial [Creating and applying live templates \(code snippets\)](#) and to the [product documentation](#) for details.
- Almost like a pair programmer, PyCharm keeps an eye on what you are currently doing, and comes up with smart suggestions, which are marked with a red  or yellow  light bulb sign. If you want to know exactly what is there under the light bulb, click it, or press Alt+Enter. This way you can, for example, auto-create a new method that you are already using in the code, but have not yet declared. Refer to the tutorial [Quick fixes and intention actions once more](#) and [product documentation](#) for details.

Use macros

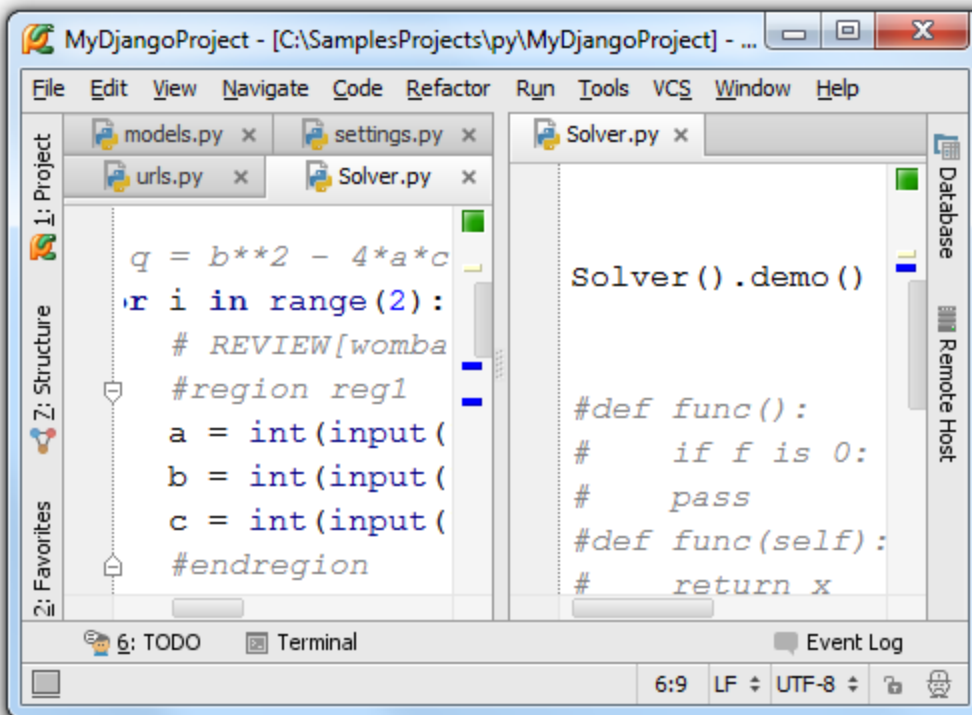
It would be nice to have a chance to simplify your work with the editor. Using macros is just one of the possible ways. Refer to the tutorial [Using macros](#) and to the [documentation](#) for details.

Work with multiple files

As you might have noticed, PyCharm opens each file in a separate tab in the editor:



PyCharm provides a handy way to [switch between the various tabs](#), allows [pinning and unpinning each tab](#), placing editor tabs along any of the four borders of the window, [splitting](#) the editor tabs, and more.

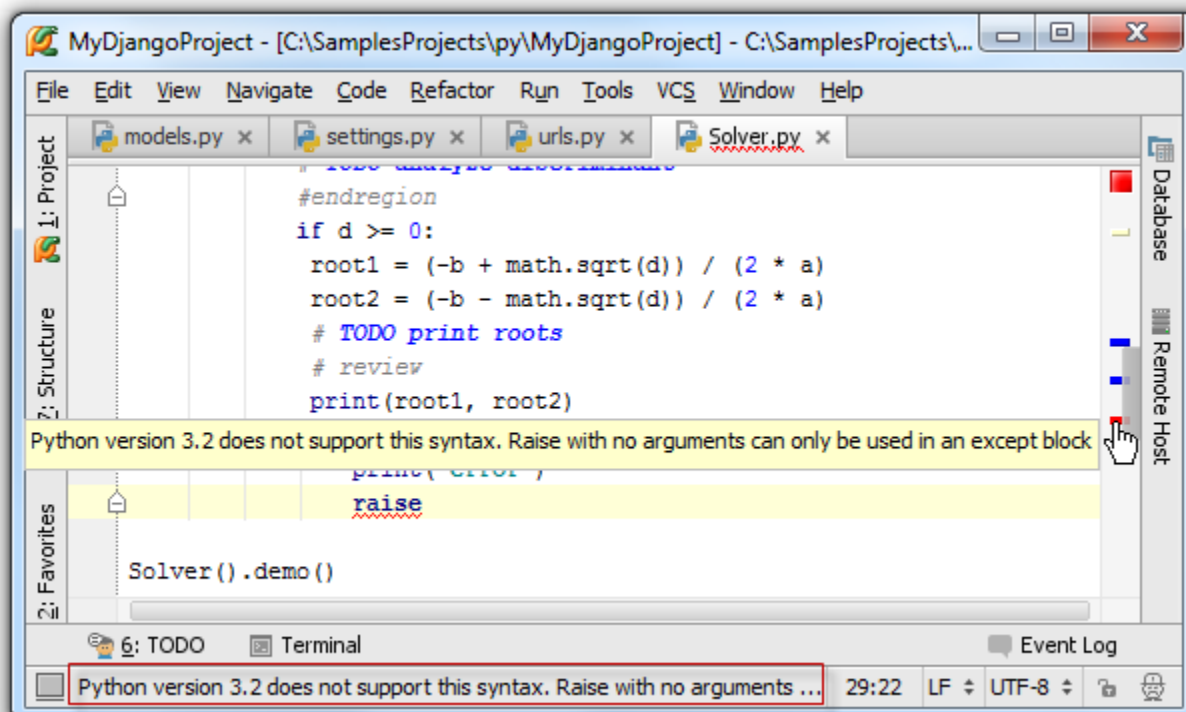


Refer to the [tutorial](#) and documentation [section Managing editor tabs](#) for details.

Analyze code transparently

PyCharm gives you numerous hints and prompts to help you avoid errors, or correct them, if they occur.

First, as you type, you immediately have all syntactical errors underlined with a red wavy line. If you place the caret at an error location, you will see a brief description of the problem at the tooltip, and also in the left side of the [Status bar](#). Besides that, you see red stripes along the [validation bar](#), or the [marker bar](#). If you hover your mouse pointer over such a stripe, you see the error description at the tooltip:



As you type our code, PyCharm, almost like a pair programmer, looks over your shoulder and suggests to fix your errors or just improve your code, by showing you red

or yellow light bulbs (we've mentioned them already in the section [Write code smartly!](#)). Learn how to use them in the [tutorial](#) and in the [product documentation](#).

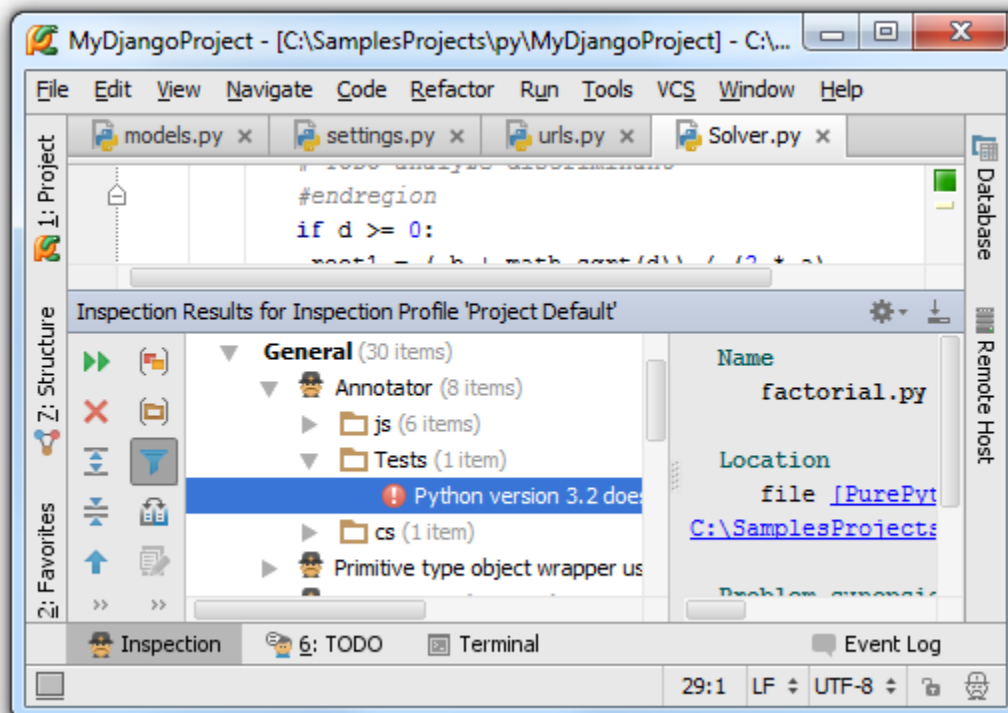
The next level is static code analysis, or [code inspections](#): your code is analyzed without actually executing it. Actually, PyCharm inspects code in the current file on-the-fly, and shows inspection results in the marker bar as colored stripes. If you see that the right side of your IDE frame is bright with red stripes, beware — it means that your code contains serious errors. Less important things, recommendations to improve code, or warnings, are displayed as yellow stripes. Information for the current file is summarized in a colored indicator on top of the marker bar, which works as traffic lights: green means that everything is fine; at least one problem turns the indicator yellow or red.

By the way, this guy in the Status bar - Hector - keeps an eye on each file currently opened in the editor. By default, Hector never sleeps:



If you are sure of yourself, turn him off, and you will get neither warnings, nor suggestions for improvements or error messages.

However, you might want to look deeper into the code of your application. In this case, you have to inspect a whole project, or any of its parts (Code Inspection), and explore results in the [Inspection tool window](#):



PyCharm comes with a wide range of pre-defined inspections; familiarize yourself and configure them in the [Inspections page](#) of Settings/Preferences.

Refer also to the tutorial [Syntax highlighting and error indication](#).

Create quality code

With PyCharm, it's quite easy to create a code of high quality - the IDE is a code quality tool itself. It integrates with PEP8, stands on guard of the code integrity and reports the code style violations.

Refer to the tutorial [Code Quality Assistance Tips and Tricks, or How to make your code look pretty?](#)

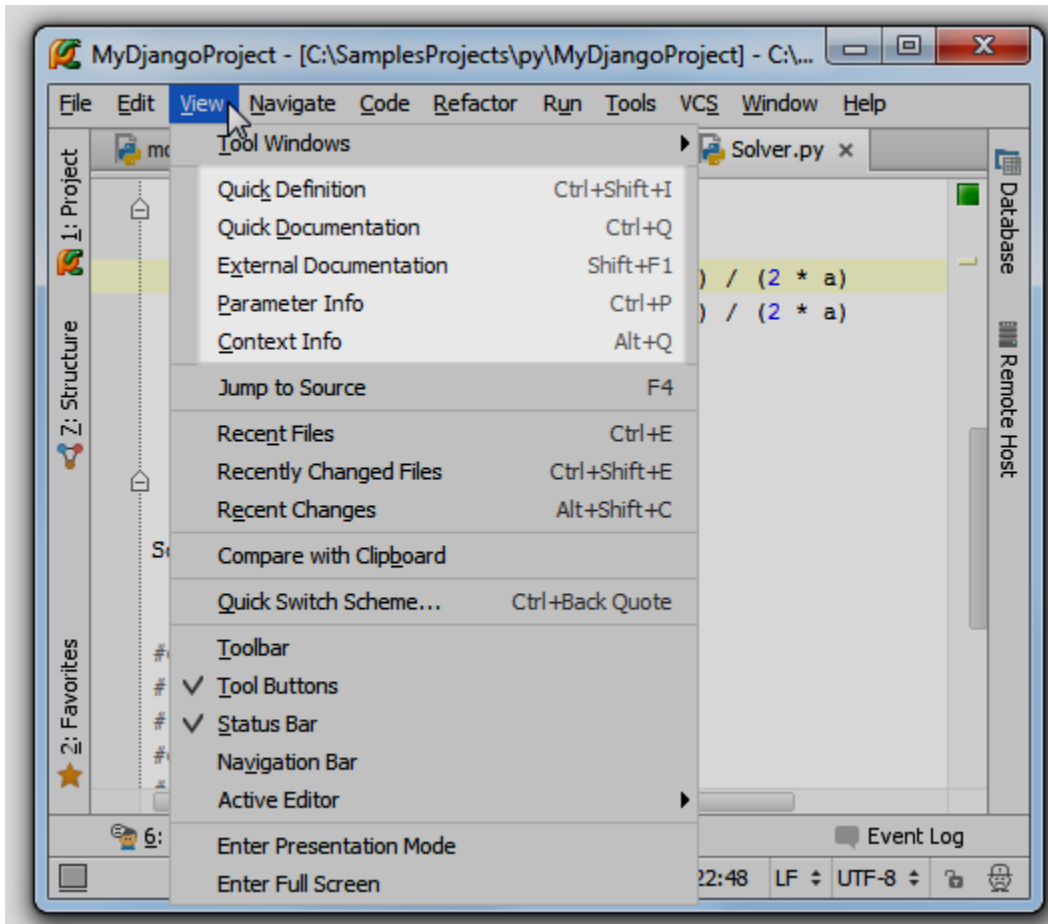
See also the product documentation on [configuring code style](#) and [code inspections](#).

View documentation

PyCharm makes it possible to view existing documentation for the symbols at caret. There are several possible ways to do it:

- Quick definition
- Quick documentation
- External documentation
- Parameter info
- Error description

All these commands are available from the View menu, and by keyboard shortcuts (the shortcuts below belong to the default scheme. If you are using some other keymap scheme, these keyboard shortcuts will be different):



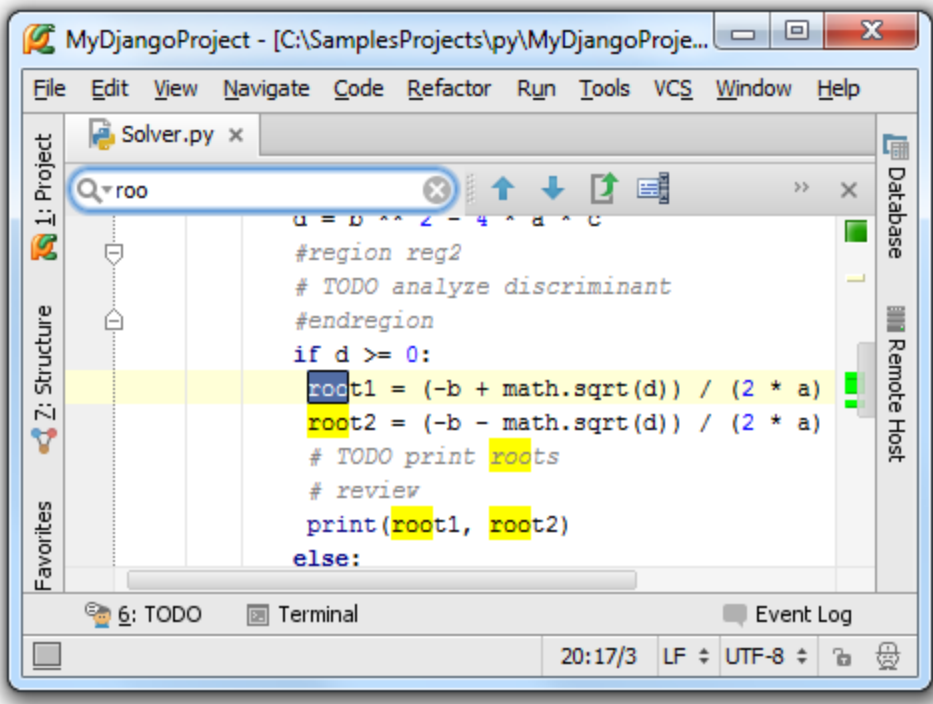
Refer to the tutorial [Viewing documentation](#) and to the product documentation [Viewing reference information](#).

Find your way through

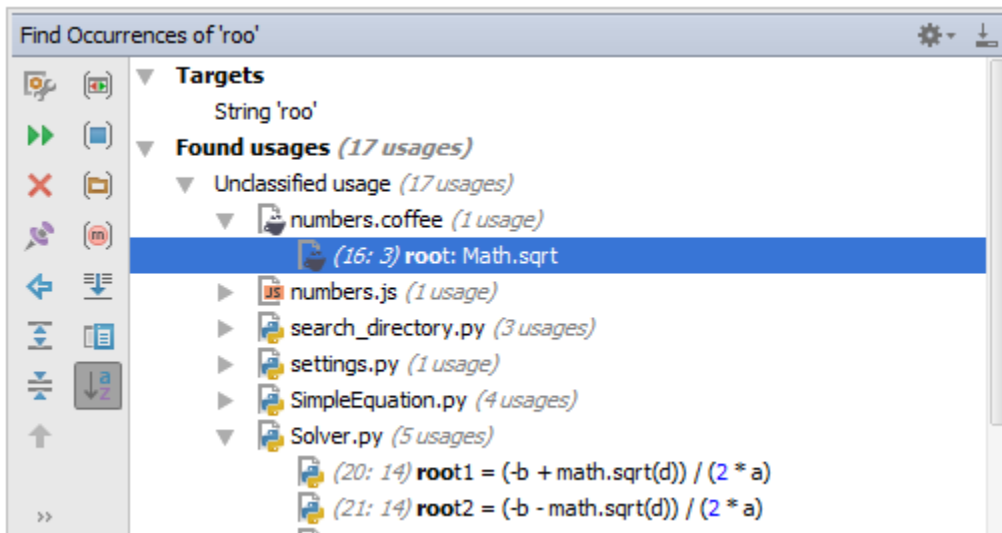
- [Navigating through the source code](#)
- [Navigating between the IDE components](#)
- [Finding an action](#)
- [Navigating everywhere](#)

Source Code

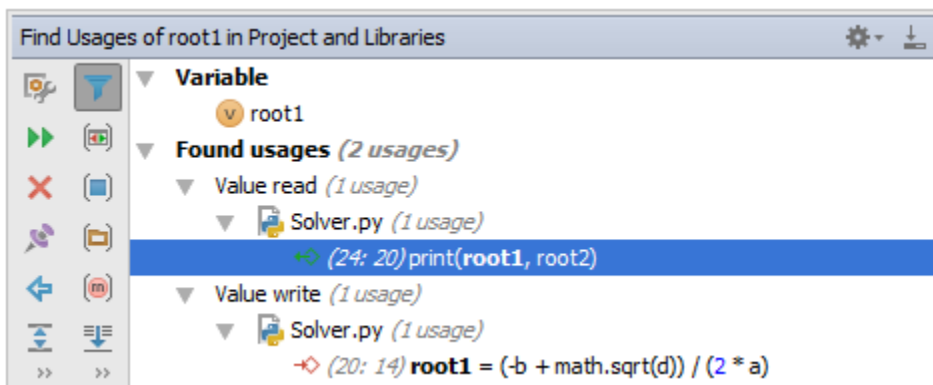
Let's start with [finding fragments of text](#). One of the most basic means of navigation and search in the source code is our good old Ctrl+F command: start typing your search string, and get immediately to its occurrences in the current file:



But PyCharm goes further, and helps you look for a certain string within a directory, or any arbitrary scope, or an entire project (Ctrl+Shift+F):



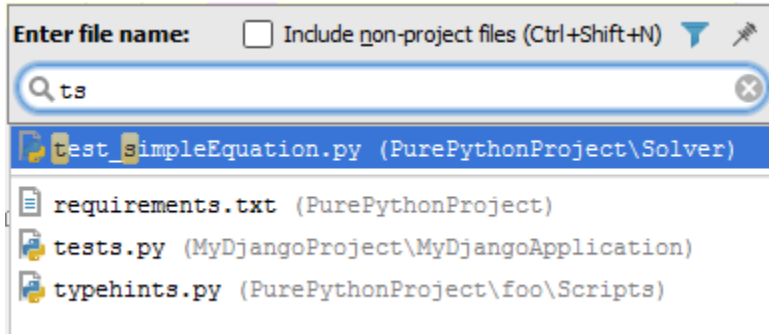
Besides that, PyCharm suggests a more sophisticated approach, namely [search for usages](#). For example, if you want to navigate from a symbol to one of its usages within your application, press Alt+F7, or choose Find Usages on its context menu:



Actually, there are several commands that help you find out where a certain symbol is used: you can jump from one usage to another in the current file (Ctrl+F7), view usages in the current file color-coded (Ctrl+Shift+F7), or across a whole project in a popup list (Ctrl+Alt+F7).

If you want to jump from a symbol to its declaration, just middle-click its name, or press Ctrl+B.

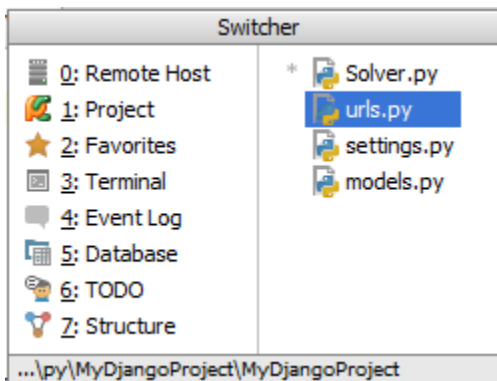
To quickly find an [element by name](#) and open it in the editor, use navigation pop-up: press Ctrl+N (for a class), Ctrl+Shift+N (for a file), or Ctrl+Shift+Alt+N (for a symbol), and start typing the name you are looking for. The list of matching names shrinks as you type, but this is just one of the handy facilities: you can use the asterisk wildcard, all caps for CamelHumps, or spaces for snake_case names, slashes for nested folders and so on, and so forth.



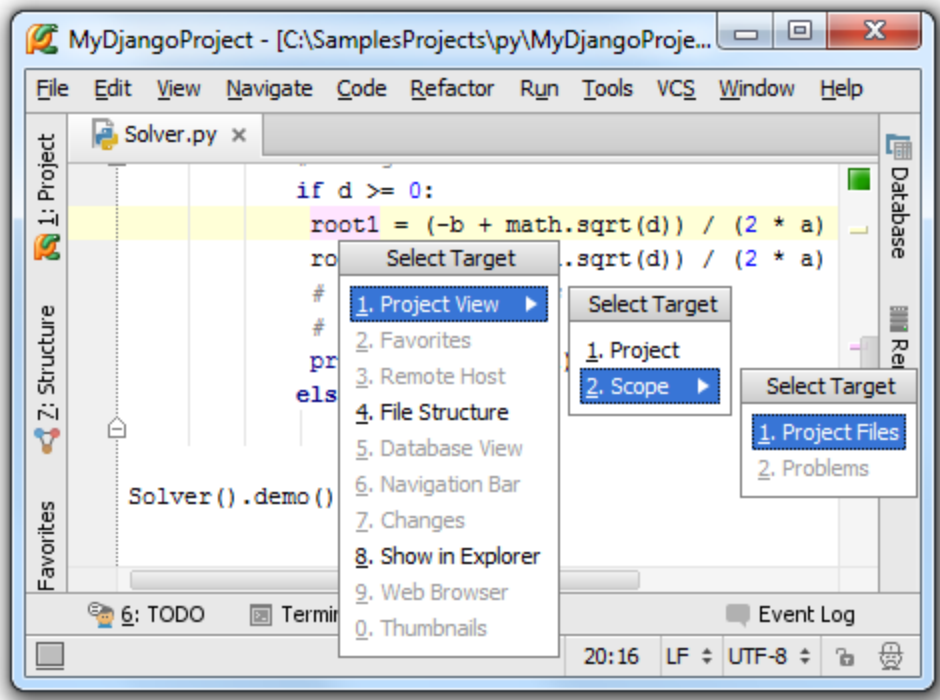
Quick search is the easiest way to find a file: with any tool window having the focus, you just start typing, and see how the matching nodes are highlighted in the tree view.

IDE Components

Ways to navigate across the IDE are numerous, and we'll briefly outline just some of them. Let's start with the [switcher](#): press Ctrl+Tab to show the switcher, which is a list of the PyCharm's tool windows and open files, and then, keeping the Ctrl key pressed, use Tab or arrow keys to scroll to the component you want to go to:



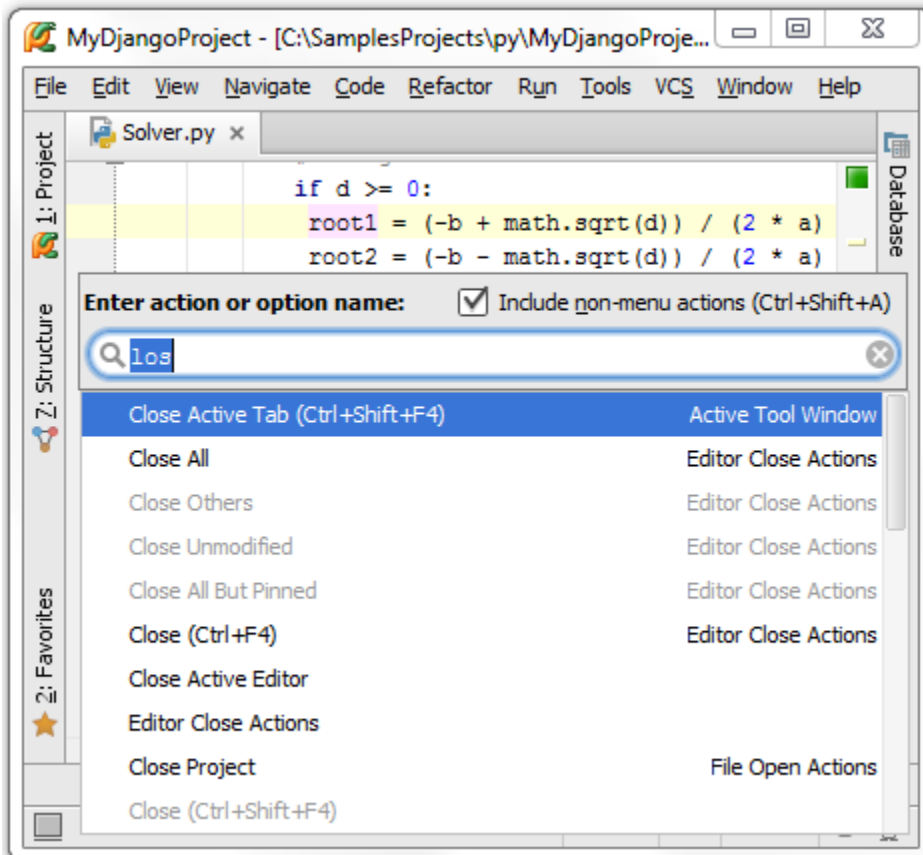
If you select a file in one of the IDE components, and want to view it in another one (the editor, Project view, Navigation bar, or a changelist), then use [Select Target](#) (Alt+F1):



And finally, don't forget that pressing Esc will bring you back to the editor, wherever you are!


Finding an action

You don't need the main menu to invoke an action - you can simply invoke it by name. Press Ctrl+Shift+A, and type characters, which, to your opinion, should be present in an action name:



Note that you can search among the actions not included in the main menu - just press Ctrl+Shift+A once more.

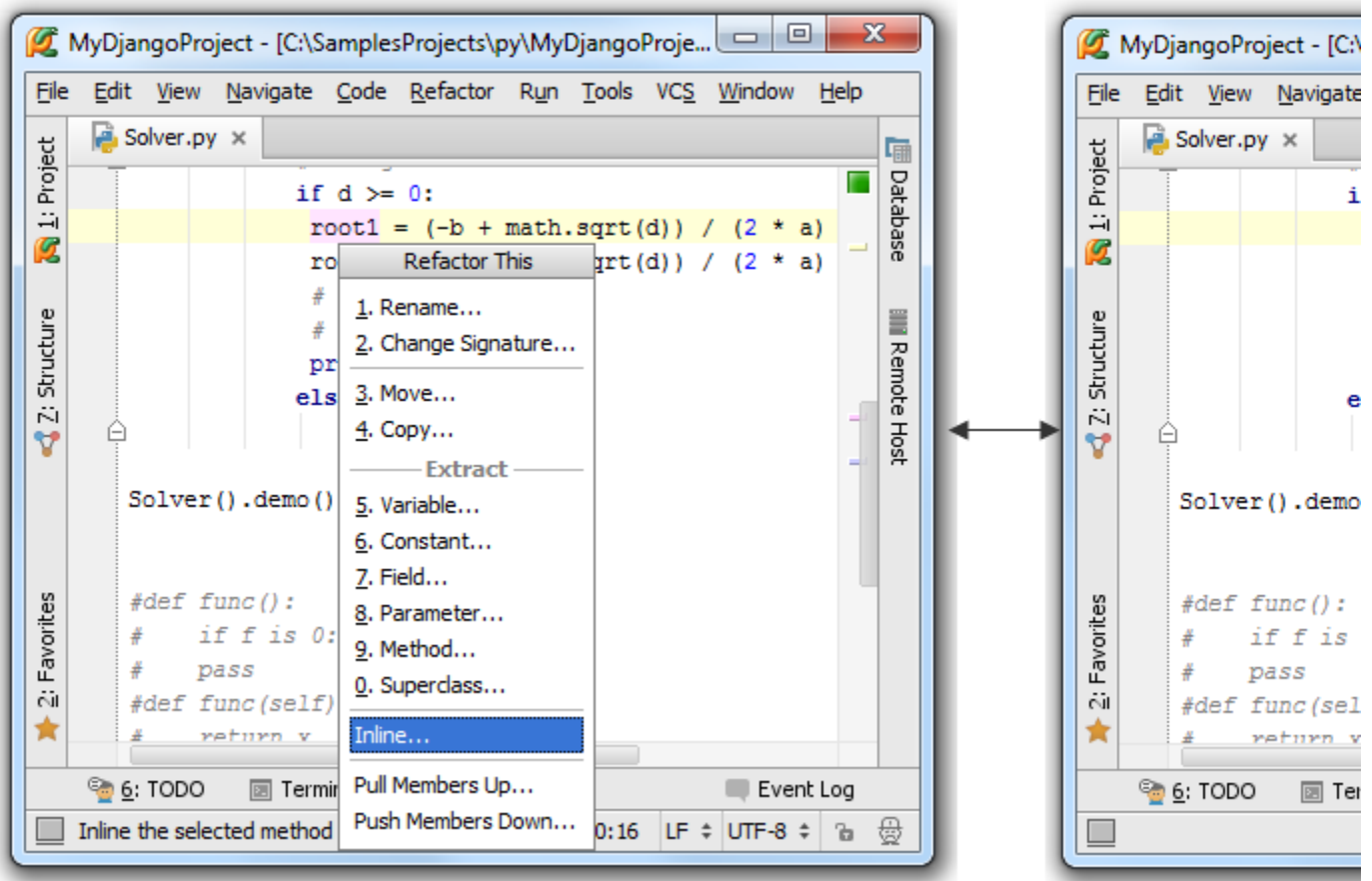
Navigating and searching everywhere

PyCharm suggests a new way of navigation and search. Double press Shift, or click  in the right end of the main toolbar, and see the [Search everywhere](#) dialog. Using this approach, you can look for any item of the source code, databases, actions, elements of the user interface, etc. in a single action.

Refer to the tutorials under [Exploring navigation and search](#), and to the product documentation: [Navigating through the source code](#) and [Searching through the source code](#).

Refactoring your source code

PyCharm provides wide range of [refactorings](#), from mere renaming to such complicated things as changing a method signature. Note that PyCharm suggests available refactorings depending on the current context:



Run, debug and test your application

While working with PyCharm, you'll come to a moment when you need to run or debug an application, a script, or a test. In all these cases, you need a special profile, or a run/debug configuration, which defines script name, working directory, environment variables, and other vital things.

Running

You can easily launch a Python script from its context menu, or with a handy shortcut Ctrl+Shift+F10. However, if you want to use some other run/debug configuration, you have to choose one on the main menu, and then press Shift+F10.

Refer to the tutorial [Code running assistance](#), and to the product documentation [Running](#).

Note that you can launch your applications both locally and remotely: PyCharm allows using remote hosts, [virtual machine](#) and [Vagrant boxes](#).

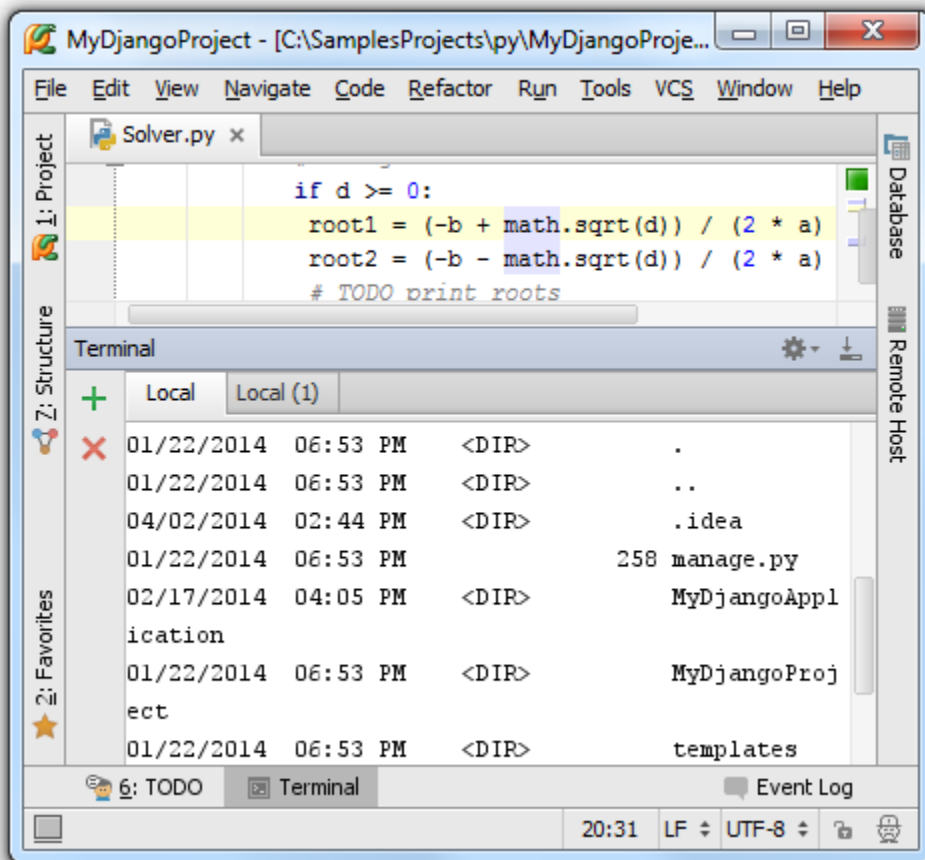
REPL console

PyCharm also helps those who love the full control of an interactive console: on the Tools menu, you can find commands that launch the interactive Python or Django consoles. Here you can type commands and execute them immediately. Moreover, PyCharm's interactive consoles feature syntax highlighting, code completion, and allow scrolling through the history of commands (Ctrl+Up/Down).

PyCharm also makes it possible to run in console source code from the editor - just make your selection, and then press Alt+Shift+I (Execute selection in console on the context menu of the selection). This feature is explored in detail in the tutorial [REPL - running an interactive console](#). See also the product documentation [Working with consoles](#).

Local terminal

Cannot live without command line? OK, PyCharm helps with that too. Choose Tools>Open Terminal on the main menu, and enjoy:



Refer to the [product documentation](#) for details.

Debugging

Oops... your application or script runs into a run-time error? To find out its origin, you will have to do some debugging. It starts with placing breakpoints (quite simple - just click the left gutter of the line where you want the breakpoint to appear), at which program execution will be suspended, so you can explore program data.

Launching the debugging session is quite as simple: either use the context menu of a specific script, or select a suitable run/debug configuration, and then press Shift+F9.

And finally, a very convenient way to select run/debug configuration and immediately launch it is Alt+Shift+F10/ Alt+Shift+F9.

Refer to the tutorials [Debugger](#), [Breakpoints](#), and [Debug run](#), and to the product documentation [Debugging](#).

Testing

It is a good idea to test your applications, and PyCharm helps doing it as simple as possible. With PyCharm, you can:

- Create test classes
- Create special testing run/debug configurations
- Run and debug tests right from the IDE
- Explore results in the test runner window

PyCharm supports all the major Python testing frameworks: [Unittest](#), [Doctest](#), [Nosetest](#), [py.test](#) and [Attest](#). For each of these frameworks, PyCharm provides its own [run/debug configuration](#).

Refer to the tutorials [Creating and running a Python unit test](#), [Unit tests](#), and to the product documentation [Testing](#).

Remote development

Besides the ability to use interpreters located remotely (which we have already discussed in the guide), you can also develop remotely. First, with PyCharm, you can deploy your local applications to some remote server. To learn about deployment servers, refer to the product documentation section

[Configuring Synchronization with a Web Server](#).

Having deployed an application, you can run, debug and test it remotely. PyCharm also helps you [compare local and remote folders](#), and [synchronize local copy with that deployed on the server](#).

Data sources and SQL support

As you might have noticed already, creating projects of the various types (Django, Flask, etc.) requires a data source. It is also quite possible that you inject SQL statements into your source code.

PyCharm does not enable you to create databases, but provides facilities to manage and query them. Once you are granted access to a certain database, you can configure one or more data sources within PyCharm that reflect the structure of the database and store the database access credentials. Based on this information, PyCharm establishes connection to the database and provides the ability to retrieve or change information contained therein.

Refer to the [product documentation](#) for details.

Polyglot IDE

PyCharm features full-scale multi-language support. You can develop not only pure Python code, but also [JavaScript](#), [CoffeeScript](#), [HTML](#), [XML](#), and more. So doing, the majority of the powerful PyCharm's coding assistance (code completion, syntax and error highlighting, code analysis, intention actions and quick fixes, and more) is available for the supported languages. Refer to the PyCharm documentation pages of the corresponding languages to learn more about the scope of support.

With PyCharm, you can not only create JavaScript code, but debug it too. Find details in the [JavaScript-Specific Guidelines](#), and in the tutorial [Debugging JavaScript with PyCharm](#).

That's it, folks!

Here we have given a very concise overview of some vital PyCharm facilities, just to give you a quick start. There are numerous important features that make developer's life nice and easy, and the source code nice and clean. Try now these primary steps, and then dig deeper. Enjoy! We realize you may still have questions. We welcome you to ask them on [PyCharm Discussion Forum](#).

For further reading please check [PyCharm's tutorials space](#) and [Resources page](#).