# PyCharm

tutorialspoint

SIMPLY EASY LEARNING

## About the Tutorial

PyCharm is the most popular IDE for Python, and includes great features such as excellent code completion and inspection with advanced debugger and support for web programming and various frameworks. PyCharm is created by Czech company, Jet brains which focusses on creating integrated development environment for various web development languages like JavaScript and PHP.

## Audience

This tutorial has been prepared for Python developers who focus on using IDE with complete package of running, debugging and creating projects in various python frameworks. Also, interested learners with a basic knowledge of any IDE can take up this tutorial.

## Prerequisites

Before proceeding with this tutorial, you need a basic knowledge of any integrated development environment of Python like Sublime Text or most popular IDE like NetBeans. If you are a beginner, we suggest you to go through tutorials related to these topics first before proceeding further on this tutorial.

## Copyright & Disclaimer

# Table of Contents

# 1. PyCharm- Introduction

PyCharm is the most popular IDE used for Python scripting language. This chapter will give you an introduction to PyCharm and explains its features.

PyCharm offers some of the best features to its users and developers in the following aspects:

- Code completion and inspection
- Advanced debugging
- Support for web programming and frameworks such as Django and Flask

## Features of PyCharm

Besides, a developer will find PyCharm comfortable to work with because of the features mentioned below:

### Code Completion

PyCharm enables smoother code completion whether it is for built in or for an external package.

### SQLAlchemy as Debugger

You can set a breakpoint, pause in the debugger and can see the SQL representation of the user expression for SQL Language code.

### Git Visualization in Editor

When coding in Python, queries are normal for a developer. You can check the last commit easily in PyCharm as it has the blue sections that can define the difference between the last commit and the current one.

### Code Coverage in Editor

You can run **.py** files outside PyCharm Editor as well marking it as code coverage details elsewhere in the project tree, in the summary section etc.

### Package Management

All the installed packages are displayed with proper visual representation. This includes list of installed packages and the ability to search and add new packages.

### Local History

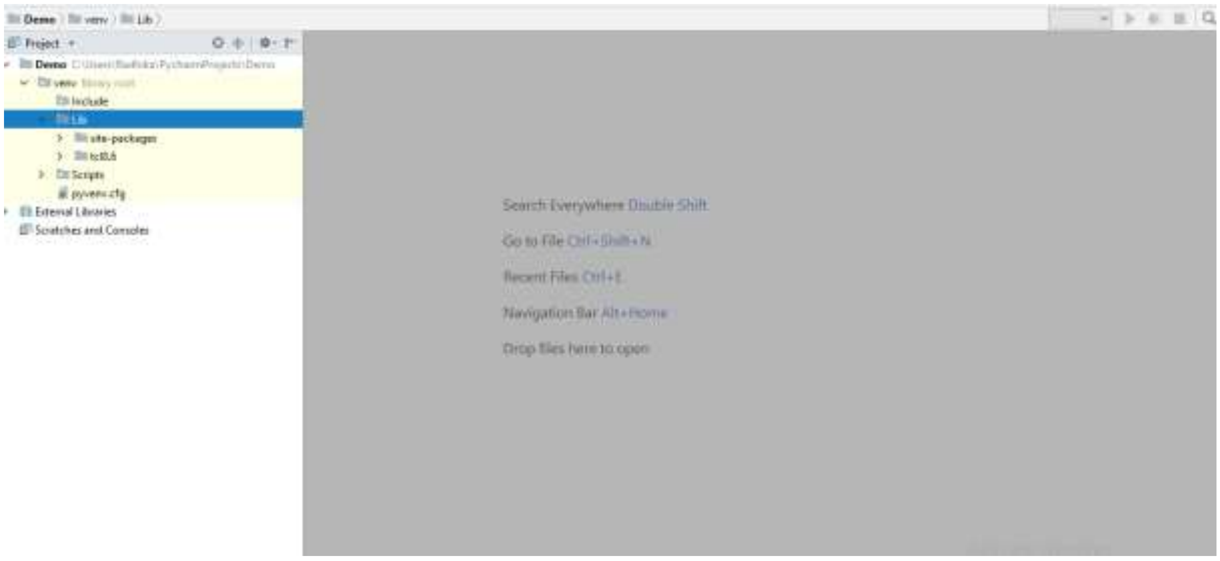
Local History is always keeping track of the changes in a way that complements like Git. Local history in PyCharm gives complete details of what is needed to rollback and what is to be added.

## Refactoring

Refactoring is the process of renaming one or more files at a time and PyCharm includes various shortcuts for a smooth refactoring process.

## User Interface of PyCharm Editor

The user interface of PyCharm editor is shown in the screenshot given below. Observe that the editor includes various features to create a new project or import from an existing project.



From the screenshot shown above, you can see the newly created project **Demo** and the **site-packages** folder for package management along with various other folders.

You can download the PyCharm Editor and read its official documentation at this link:

https://www.jetbrains.com/pycharm/

# 2. PyCharm – Installation

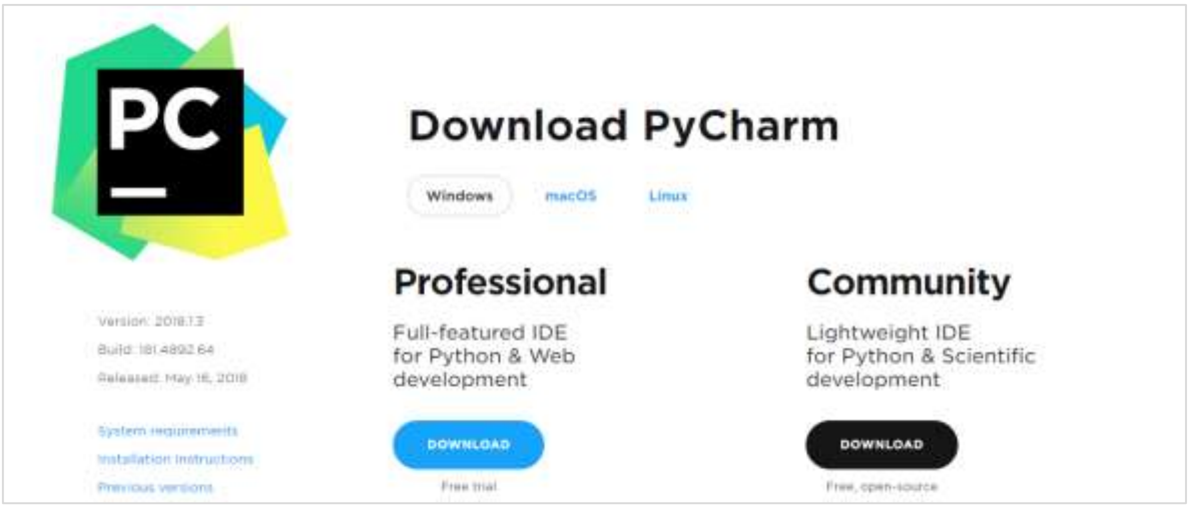
In this chapter, you will learn in detail about the installation process of PyCharm on your local computer.

## Steps Involved

You will have to follow the steps given below to install PyCharm on your system. These steps show the installation procedure starting from downloading the PyCharm package from its official website to creating a new project.

### Step 1

Download the required package or executable from the official website of PyCharm https://www.jetbrains.com/pycharm/download/#section=windows. Here you will observe two versions of package for Windows as shown in the screenshot given below:


Download PyCharm

Windows macOS Linux

Professional

Full-featured IDE for Python & Web development

DOWNLOAD

Free trial

Community

Lightweight IDE for Python & Scientific development

DOWNLOAD

Free, open-source

Version: 2018.1.3

Build: 181.4892.64

Released: May 16, 2018

System requirements

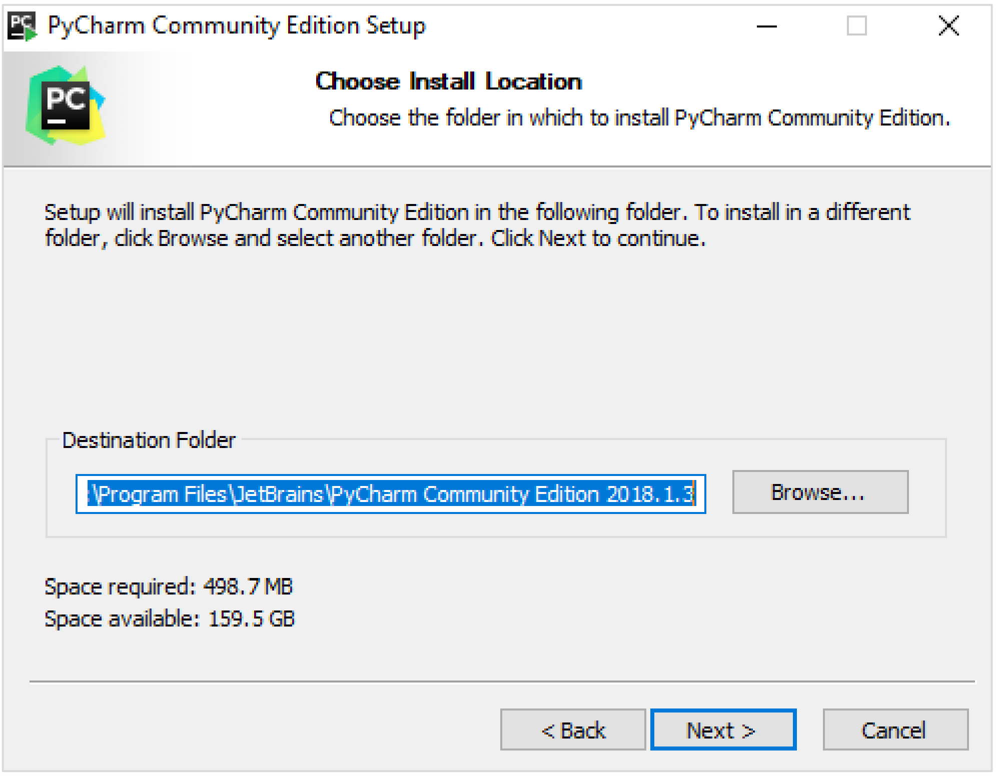
Installation Instructions

Previous versions

Note that the professional package involves all the advanced features and comes with free trial for few days and the user has to buy a licensed key for activation beyond the trial period. Community package is for free and can be downloaded and installed as and when required. It includes all the basic features needed for installation. Note that we will continue with community package throughout this tutorial.

## Step 2

Download the community package (executable file) onto your system and mention a destination folder as shown below:



PyCharm Community Edition Setup

**Choose Install Location**

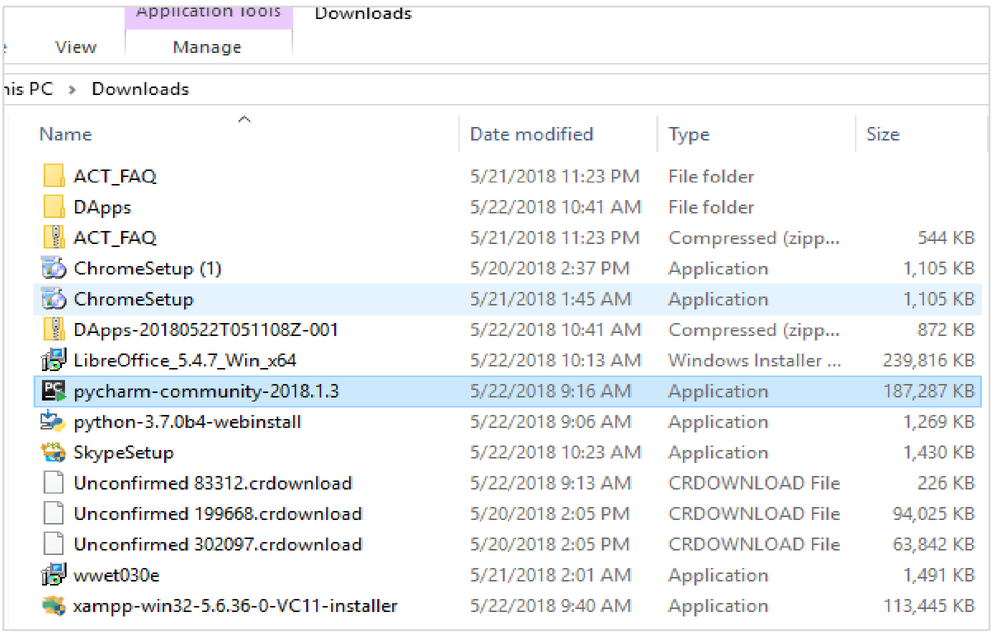
Choose the folder in which to install PyCharm Community Edition.

Setup will install PyCharm Community Edition in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

Destination Folder

\Program Files\JetBrains\PyCharm Community Edition 2018.1.3

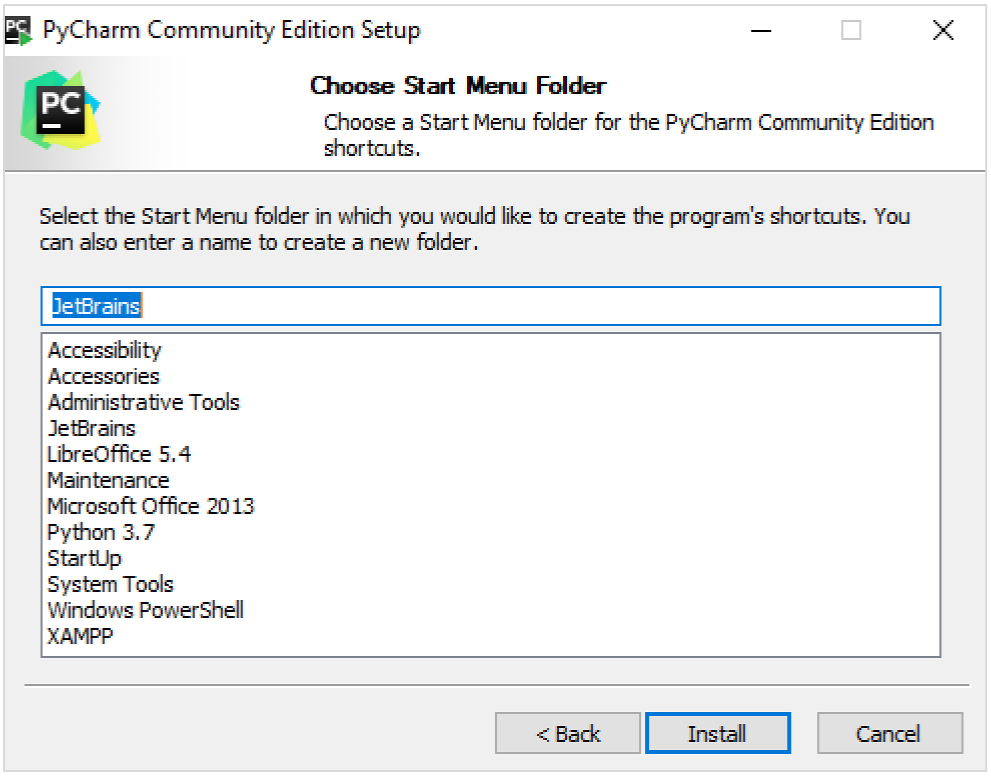
Space required: 498.7 MB
Space available: 159.5 GB



| Name | Date modified | Type | Size |
|---|---|---|---|
| ACT_FAQ | 5/21/2018 11:23 PM | File folder | |
| DApps | 5/22/2018 10:41 AM | File folder | |
| ACT_FAQ | 5/21/2018 11:23 PM | Compressed (zipp... | 544 KB |
| ChromeSetup (1) | 5/20/2018 2:37 PM | Application | 1,105 KB |
| ChromeSetup | 5/21/2018 1:45 AM | Application | 1,105 KB |
| DApps-20180522T051108Z-001 | 5/22/2018 10:41 AM | Compressed (zipp... | 872 KB |
| LibreOffice_5.4.7_Win_x64 | 5/22/2018 10:13 AM | Windows Installer ... | 239,816 KB |
| pycharm-community-2018.1.3 | 5/22/2018 9:16 AM | Application | 187,287 KB |
| python-3.7.0b4-webinstall | 5/22/2018 9:06 AM | Application | 1,269 KB |
| SkypeSetup | 5/22/2018 10:23 AM | Application | 1,430 KB |
| Unconfirmed 83312.crdownload | 5/22/2018 9:13 AM | CRDOWNLOAD File | 226 KB |
| Unconfirmed 199668.crdownload | 5/20/2018 2:05 PM | CRDOWNLOAD File | 94,025 KB |
| Unconfirmed 302097.crdownload | 5/20/2018 2:05 PM | CRDOWNLOAD File | 63,842 KB |
| wwet030e | 5/21/2018 2:01 AM | Application | 1,491 KB |
| xampp-win32-5.6.36-0-VC11-installer | 5/22/2018 9:40 AM | Application | 113,445 KB |

## Step 3

Now, begin the installation procedure similar to any other software package.



PyCharm Community Edition Setup

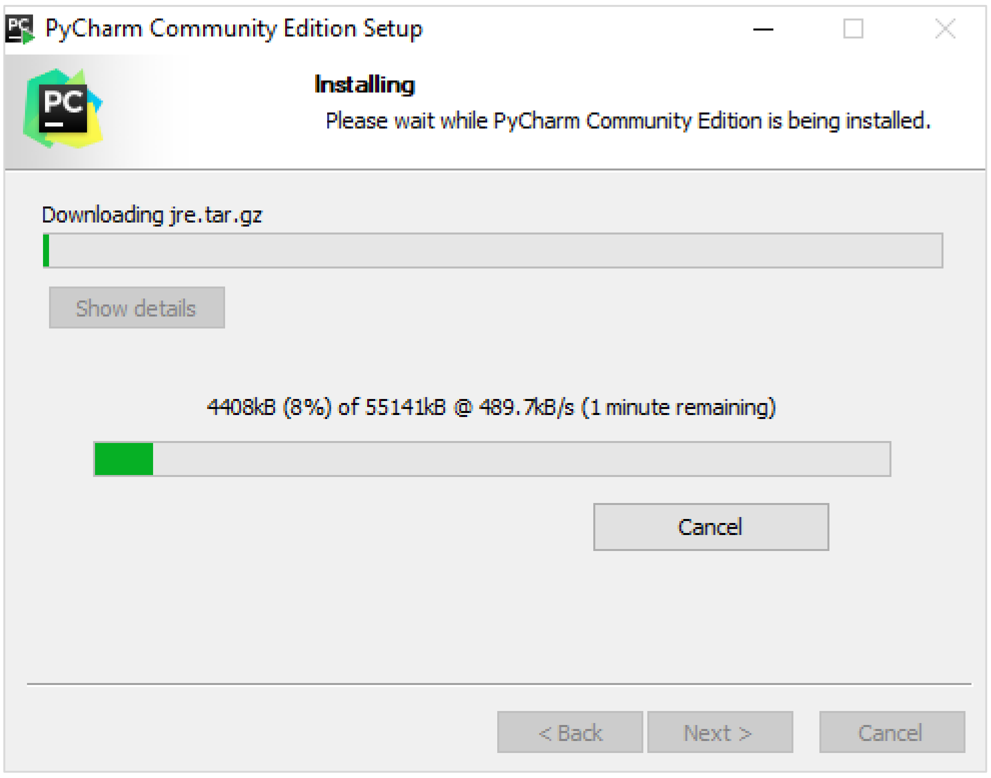
**Choose Start Menu Folder**

Choose a Start Menu folder for the PyCharm Community Edition shortcuts.

Select the Start Menu folder in which you would like to create the program's shortcuts. You can also enter a name to create a new folder.

JetBrains

Accessibility
Accessories
Administrative Tools
JetBrains
LibreOffice 5.4
Maintenance
Microsoft Office 2013
Python 3.7
StartUp
System Tools
Windows PowerShell
XAMPP



PyCharm Community Edition Setup

**Installing**

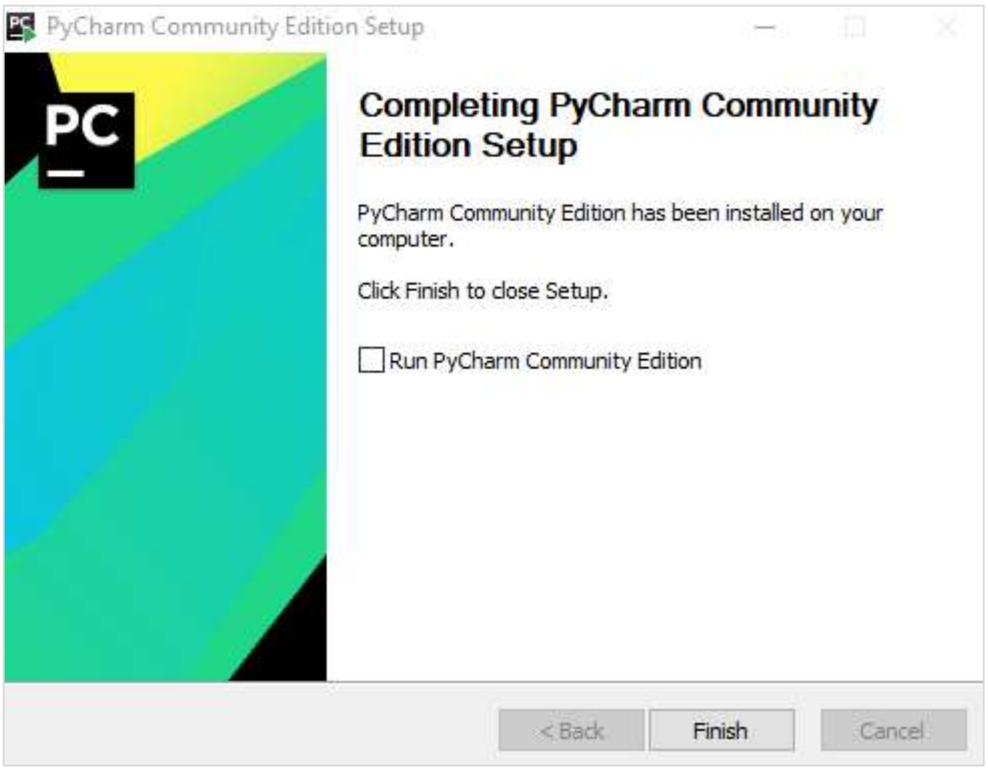
Please wait while PyCharm Community Edition is being installed.

Downloading jre.tar.gz

4408kB (8%) of 55141kB @ 489.7kB/s (1 minute remaining)

## Step 4

Once the installation is successful, PyCharm asks you to import settings of the existing package if any.



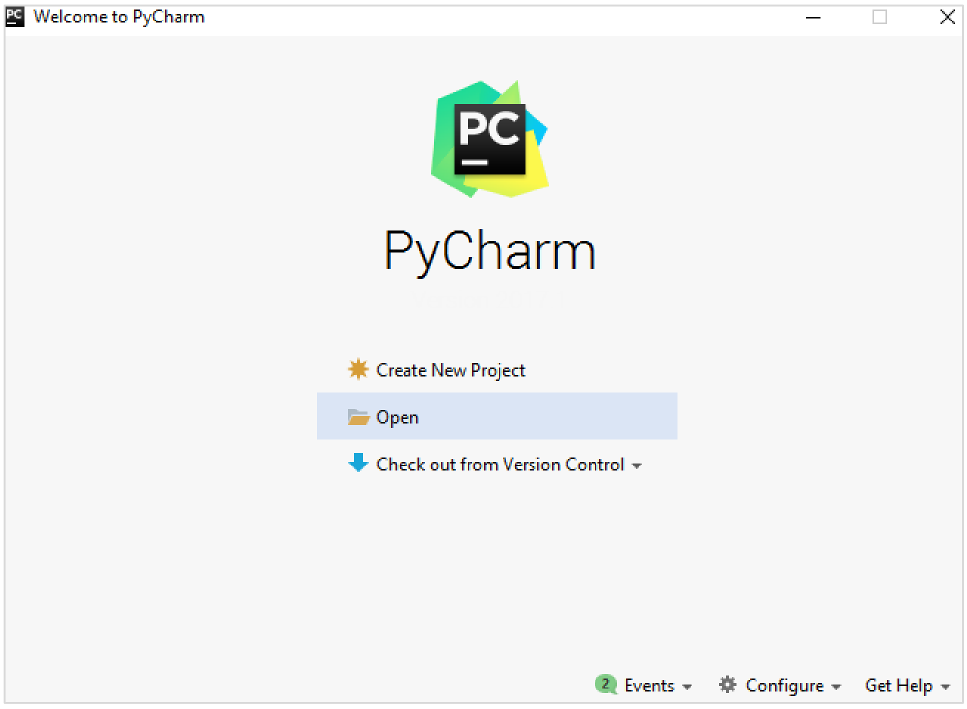
PyCharm Community Edition Setup

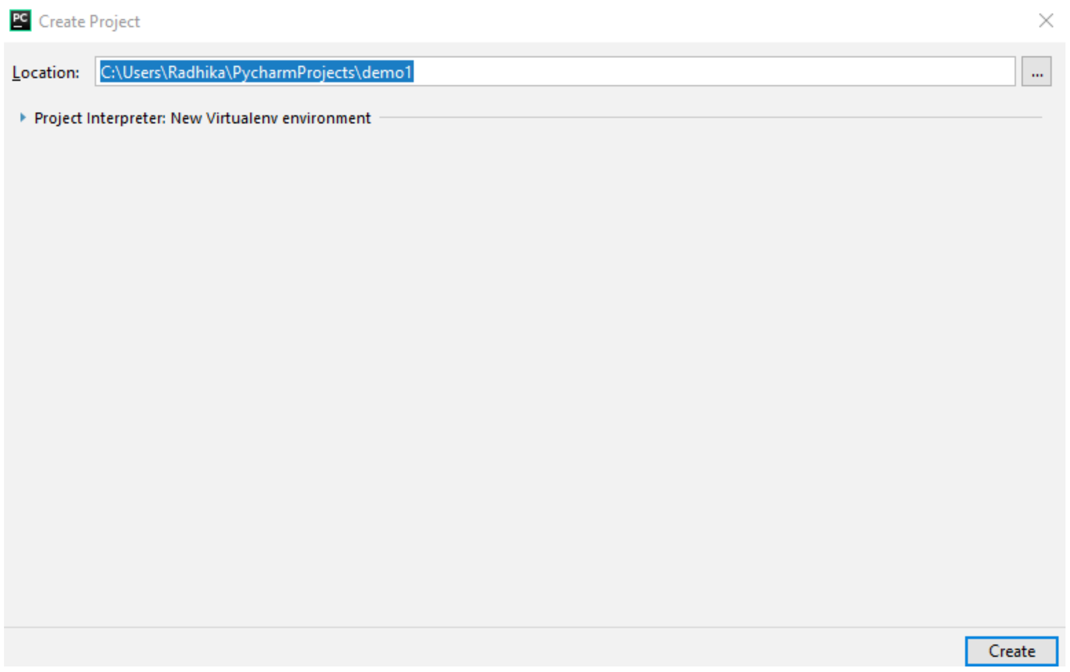
**Completing PyCharm Community Edition Setup**

PyCharm Community Edition has been installed on your computer.

Click Finish to close Setup.

☐ Run PyCharm Community Edition





Location: C:\Users\Radhika\PycharmProjects\demo1

Project Interpreter: New Virtualenv environment

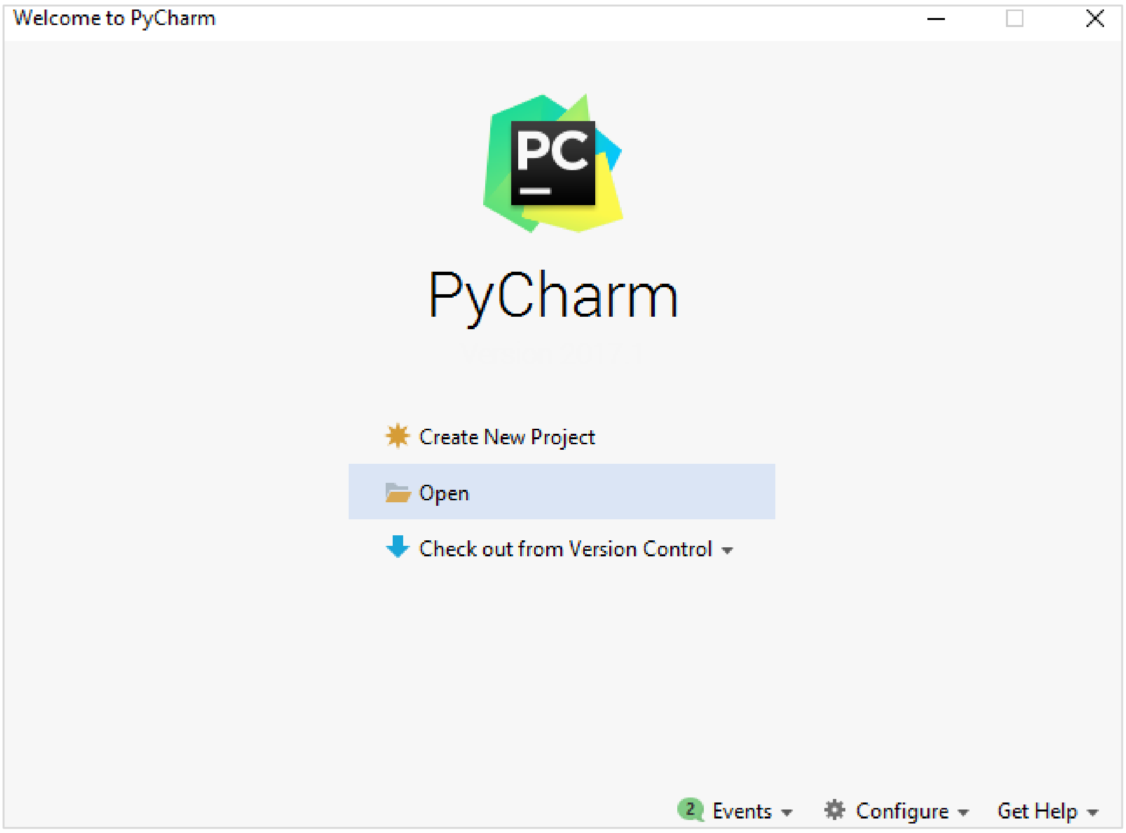
This helps in creating a new project of Python where you can work from the scratch. Note that unlike other IDEs, PyCharm only focusses on working with projects of Python scripting language.

# 3. PyCharm – Understanding Basics

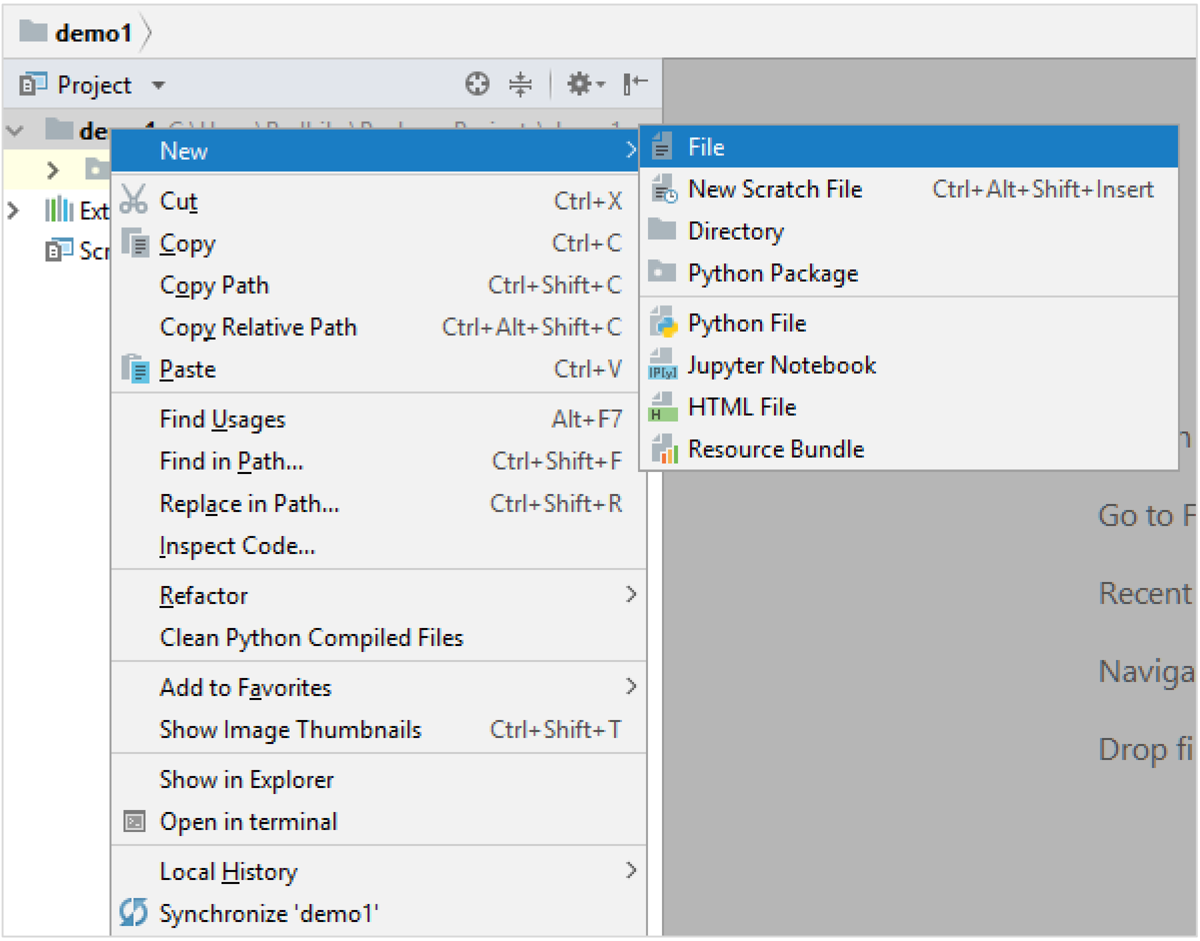
This chapter will discuss the basics of PyCharm and make you feel comfortable to begin working in PyCharm editor.

When you launch PyCharm for the first time, you can see a welcome screen with entry points to IDE such as:

- Creating or opening the project
- Checking out the project from version control
- Viewing the documentation
- Configuring the IDE



Recall that in the last chapter, we created a project named **demo1** and we will be referring to the same project throughout this tutorial. Now we will start creating new files in the same project to understand the basics of PyCharm Editor.

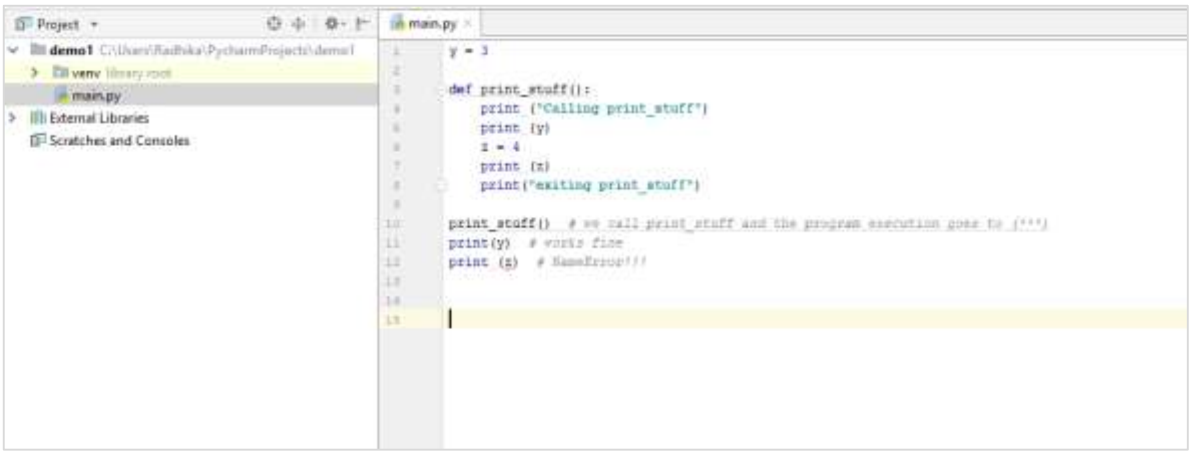


The above snapshot describes the project overview of demo1 and the options to create a new file. Let us create a new file called **main.py**.

The code included in main.py is as follows:

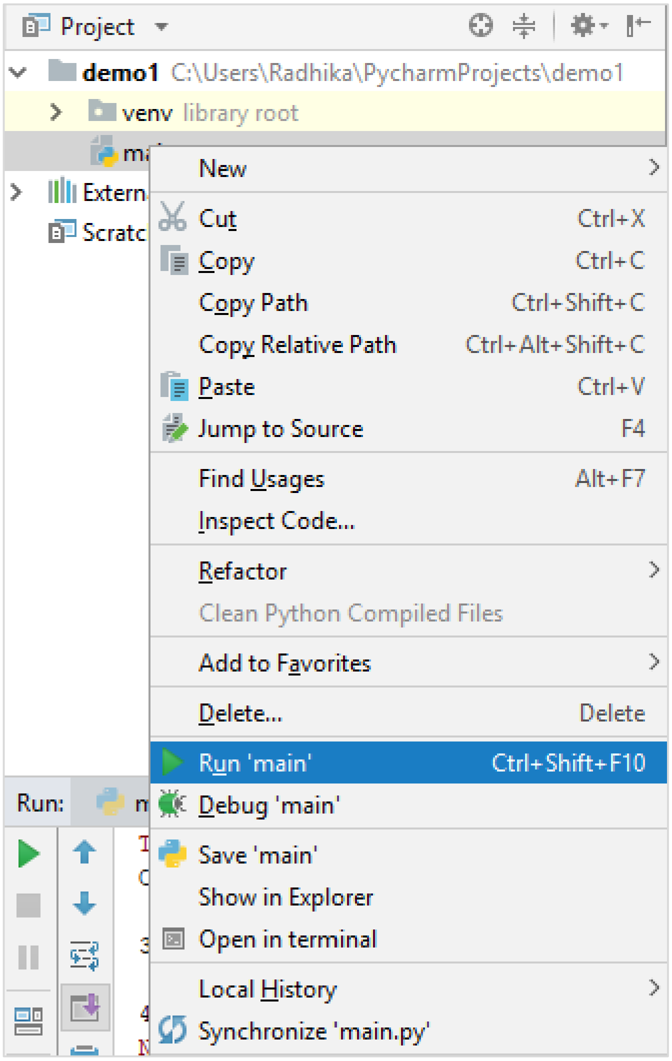
```
y = 3

def print_stuff():
    print ("Calling print_stuff")
    print (y)
    z = 4
    print (z)
    print("exiting print_stuff")

print_stuff()  # we call print_stuff and the program execution goes to (***)
print(y)  # works fine
print (z)  # NameError!!!
```

The code created in the file **main.py** using PyCharm Editor is displayed as shown below:

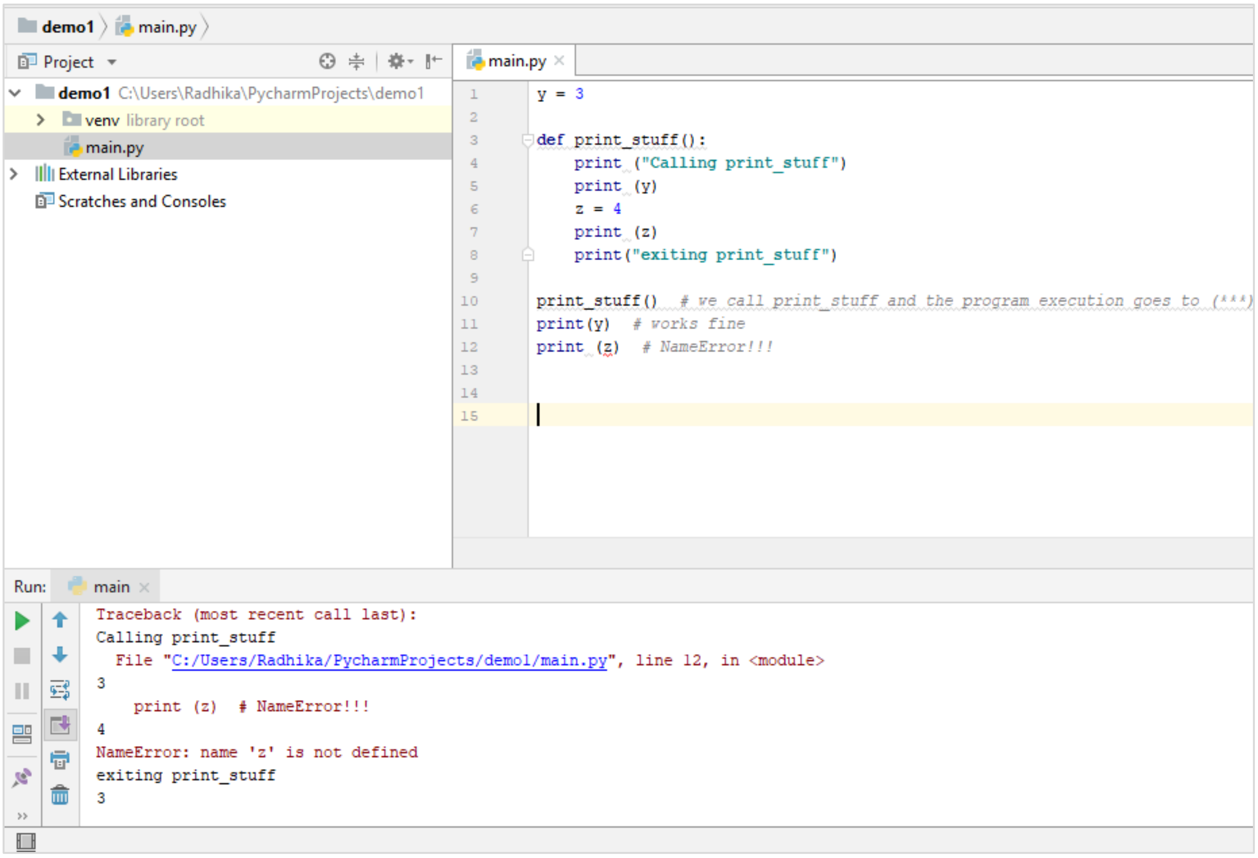


```
y = 3

def print_stuff():
    print ("Calling print_stuff")
    print (y)
    z = 4
    print (z)
    print("exiting print_stuff")

print_stuff()  # we call print_stuff and the program execution goes to (***)
print(y)  # works fine
print (z)  # NameError!!!
```

This code can be run within IDE environment. The basic demonstration of running a program is discussed below:



Note that we have included some errors within the specified code such that console can execute the code and display output as the way it is intended to.



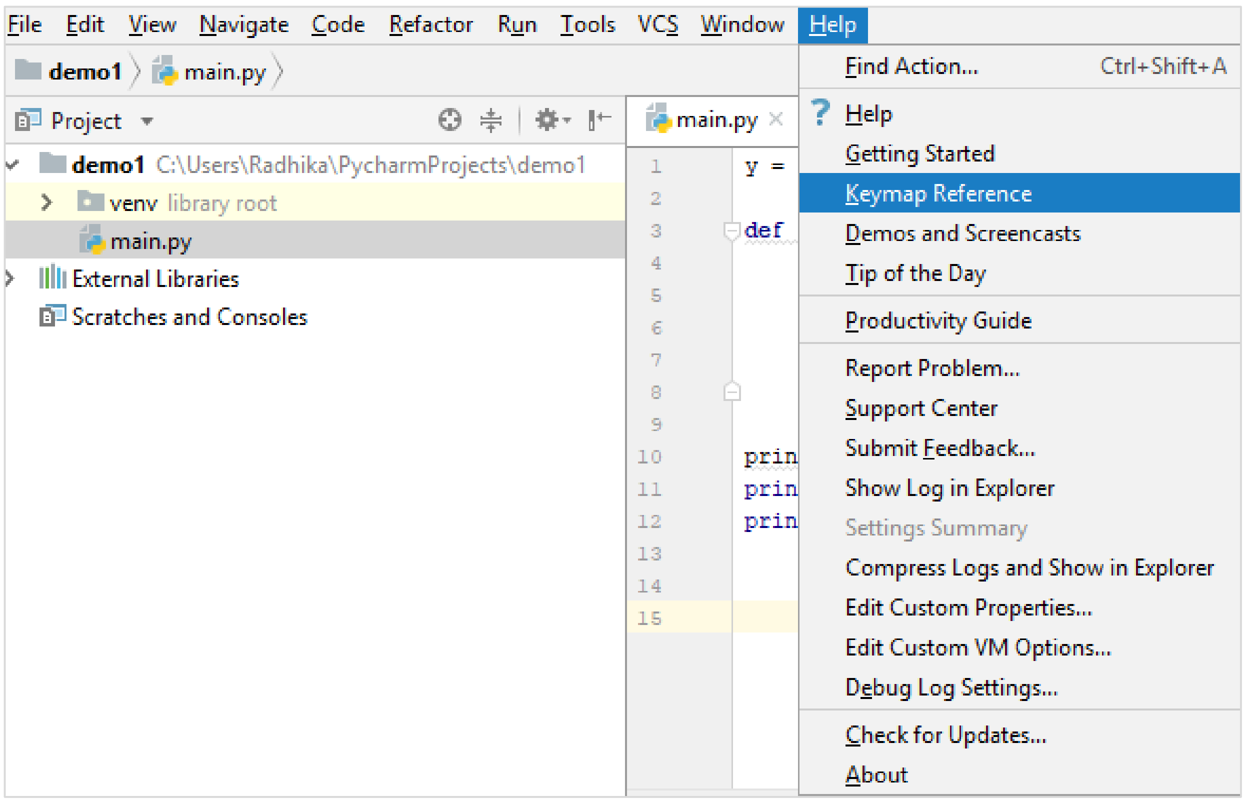
```
y = 3

def print_stuff():
    print ("Calling print_stuff")
    print (y)
    z = 4
    print (z)
    print("exiting print_stuff")

print_stuff()   # we call print_stuff and the program execution goes to (***)
print(y)   # works fine
print (z)   # NameError!!!
```

```
Traceback (most recent call last):
Calling print_stuff
  File "C:/Users/Radhika/PycharmProjects/demo1/main.py", line 12, in <module>
3
    print (z)  # NameError!!!
4
NameError: name 'z' is not defined
exiting print_stuff
3
```

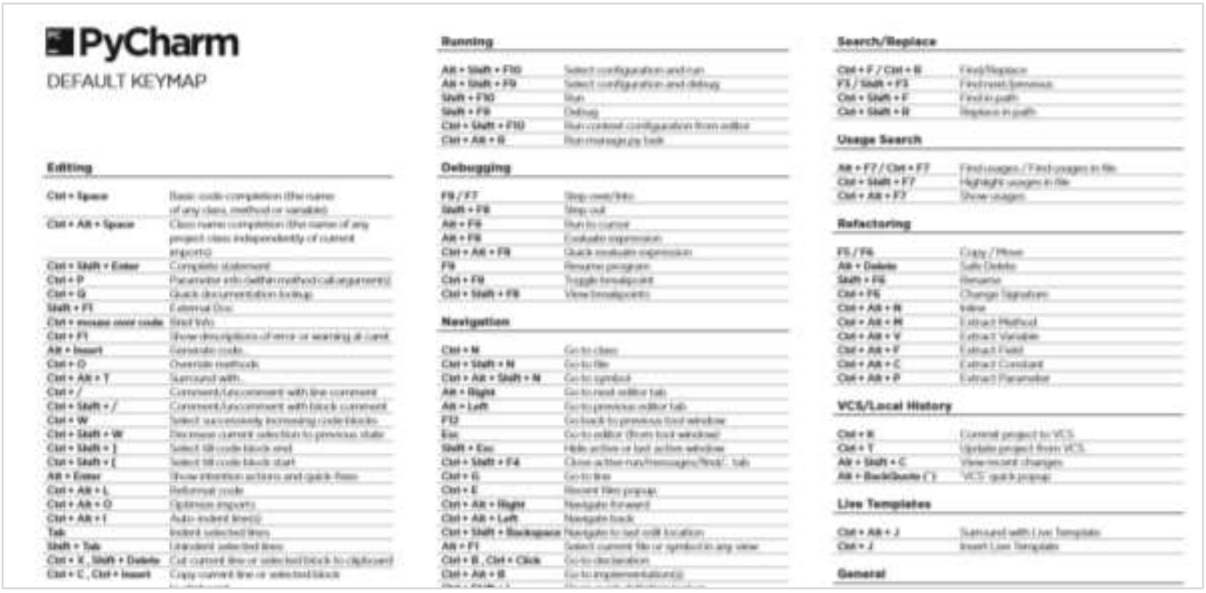
# 4. PyCharm – Keymaps

PyCharm includes various Keymaps to show the most-used commands in the editor. This chapter discusses Keymaps in detail.

You can find the list of Keymaps available in the file menu **Help -> Keymap Reference** as shown in the screenshot given below:



You can find the list of Keymaps and the available shortcuts in PDF format as shown below:



PyCharm
DEFAULT KEYMAP

**Editing**

| | |
|---|---|
| Ctrl + Space | Basic code completion (the name of any class, method or variable) |
| Ctrl + Alt + Space | Class name completion (the name of any project class independently of current imports) |
| Ctrl + Shift + Enter | Complete statement |
| Ctrl + P | Parameter info (within method call arguments) |
| Ctrl + Q | Quick documentation lookup |
| Shift + F1 | External Doc |
| Ctrl + mouse over code | Brief Info |
| Ctrl + F1 | Show descriptions of error or warning at caret |
| Alt + Insert | Generate code... |
| Ctrl + O | Override methods |
| Ctrl + Alt + T | Surround with... |
| Ctrl + / | Comment/uncomment with line comment |
| Ctrl + Shift + / | Comment/uncomment with block comment |
| Ctrl + W | Select successively increasing code blocks |
| Ctrl + Shift + W | Decrease current selection to previous state |
| Ctrl + Shift + ] | Select till code block end |
| Ctrl + Shift + [ | Select till code block start |
| Alt + Enter | Show intention actions and quick-fixes |
| Ctrl + Alt + L | Reformat code |
| Ctrl + Alt + O | Optimize imports |
| Ctrl + Alt + I | Auto-indent line(s) |
| Tab | Indent selected lines |
| Shift + Tab | Unindent selected lines |
| Ctrl + X , Shift + Delete | Cut current line or selected block to clipboard |
| Ctrl + C , Ctrl + Insert | Copy current line or selected block |

**Running**

| | |
|---|---|
| Alt + Shift + F10 | Select configuration and run |
| Alt + Shift + F9 | Select configuration and debug |
| Shift + F10 | Run |
| Shift + F9 | Debug |
| Ctrl + Shift + F10 | Run context configuration from editor |
| Ctrl + Alt + R | Run manage.py task |

**Debugging**

| | |
|---|---|
| F8 / F7 | Step over/into |
| Shift + F8 | Step out |
| Alt + F9 | Run to cursor |
| Alt + F8 | Evaluate expression |
| Ctrl + Alt + F8 | Quick evaluate expression |
| F9 | Resume program |
| Ctrl + F8 | Toggle breakpoint |
| Ctrl + Shift + F8 | View breakpoints |

**Navigation**

| | |
|---|---|
| Ctrl + N | Go to class |
| Ctrl + Shift + N | Go to file |
| Ctrl + Alt + Shift + N | Go to symbol |
| Alt + Right | Go to next editor tab |
| Alt + Left | Go to previous editor tab |
| F12 | Go back to previous tool window |
| Esc | Go to editor (from tool window) |
| Shift + Esc | Hide active or last active window |
| Ctrl + Shift + F4 | Close active run/messages/find/... tab |
| Ctrl + G | Go to line |
| Ctrl + E | Recent files popup |
| Ctrl + Alt + Right | Navigate forward |
| Ctrl + Alt + Left | Navigate back |
| Ctrl + Shift + Backspace | Navigate to last edit location |
| Alt + F1 | Select current file or symbol in any view |
| Ctrl + B , Ctrl + Click | Go to declaration |
| Ctrl + Alt + B | Go to implementation(s) |

**Search/Replace**

| | |
|---|---|
| Ctrl + F / Ctrl + R | Find/Replace |
| F3 / Shift + F3 | Find next/previous |
| Ctrl + Shift + F | Find in path |
| Ctrl + Shift + R | Replace in path |

**Usage Search**

| | |
|---|---|
| Alt + F7 / Ctrl + F7 | Find usages / Find usages in file |
| Ctrl + Shift + F7 | Highlight usages in file |
| Ctrl + Alt + F7 | Show usages |

**Refactoring**

| | |
|---|---|
| F5 / F6 | Copy / Move |
| Alt + Delete | Safe Delete |
| Shift + F6 | Rename |
| Ctrl + F6 | Change Signature |
| Ctrl + Alt + N | Inline |
| Ctrl + Alt + M | Extract Method |
| Ctrl + Alt + V | Extract Variable |
| Ctrl + Alt + F | Extract Field |
| Ctrl + Alt + C | Extract Constant |
| Ctrl + Alt + P | Extract Parameter |

**VCS/Local History**

| | |
|---|---|
| Ctrl + K | Commit project to VCS |
| Ctrl + T | Update project from VCS |
| Alt + Shift + C | View recent changes |
| Alt + BackQuote (`) | 'VCS' quick popup |

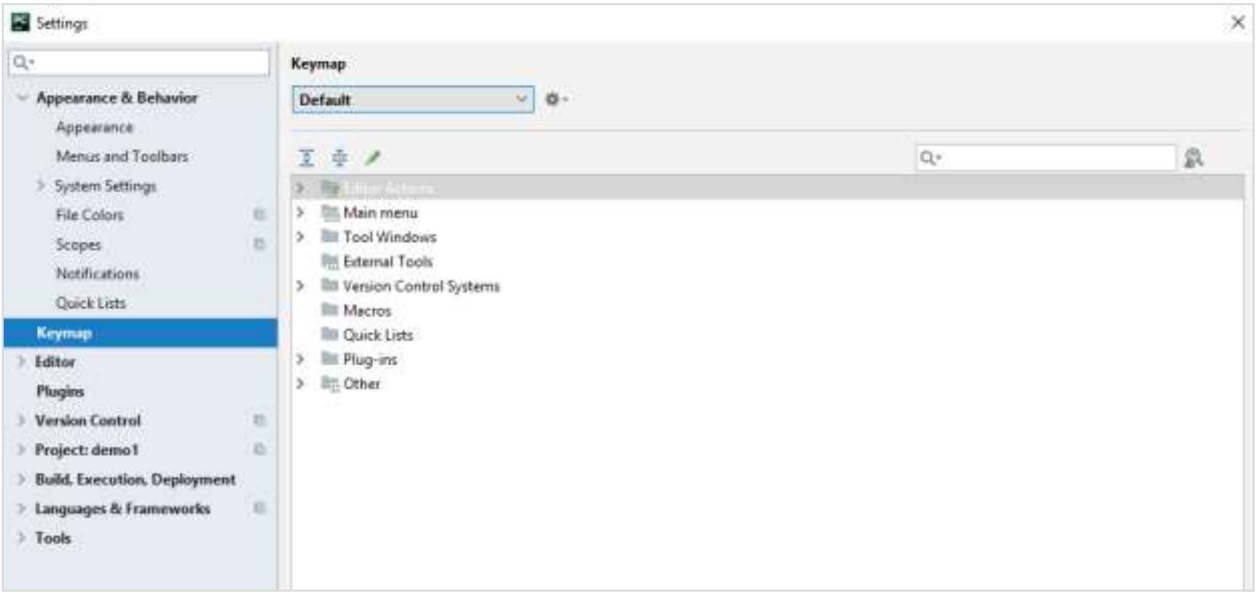
**Live Templates**

| | |
|---|---|
| Ctrl + Alt + J | Surround with Live Template |
| Ctrl + J | Insert Live Template |

**General**

Note: The default Keymap for Windows and Linux operating systems is default, while in Mac OS the default Keymap is OSX 10.5.

You can also view the list of Keymaps available using the **Settings** option in Windows and Linux Operating system (Preferences in Mac OS) as shown in the screenshot given below:



Keymap: Default

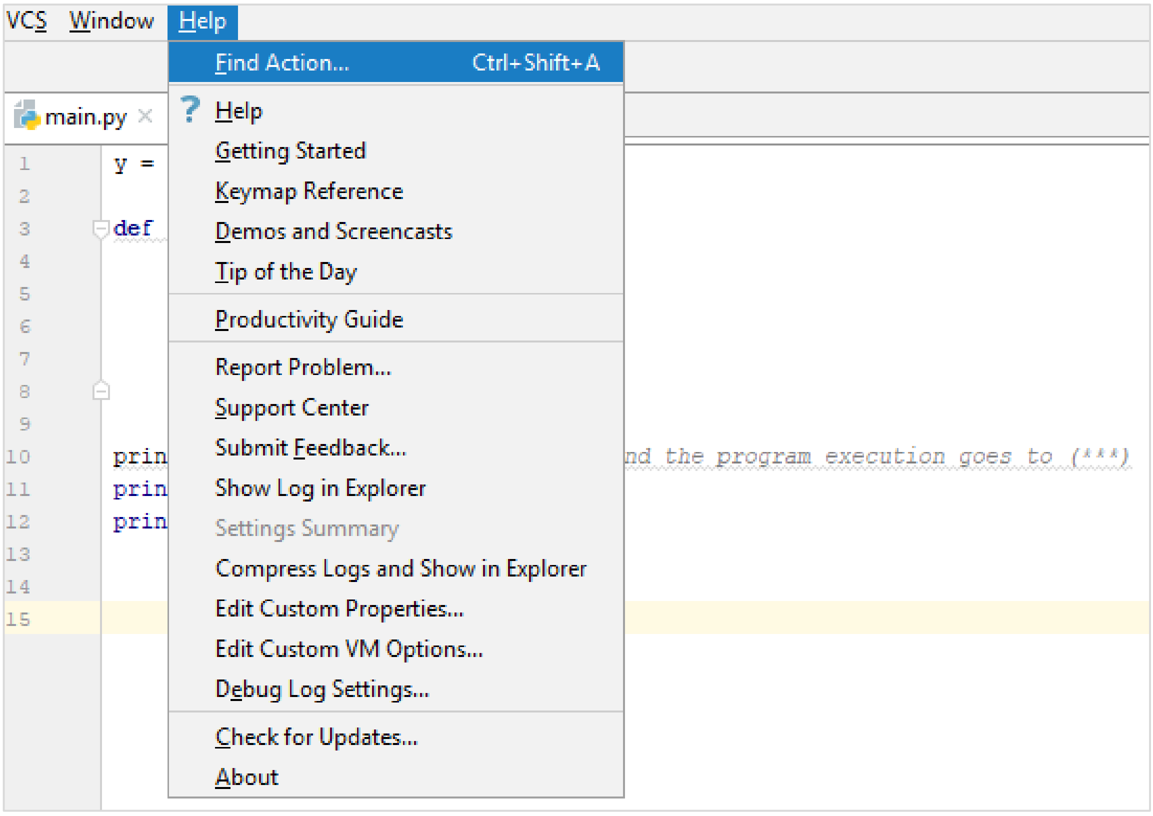
The default Keymap includes various sections for Editor Actions, Main Menu, Tool Windows, External tools, Version Control System, Macros, Quick Lists, Plug-ins and Other options as well.

# 5. PyCharm – Shortcuts

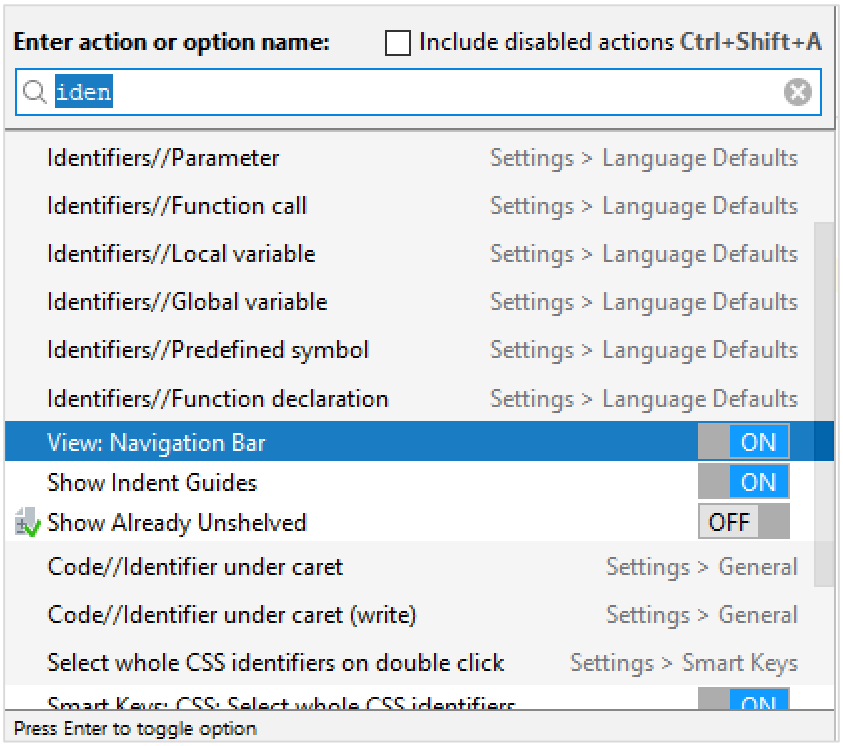
Shortcuts are the combinations of keys being used to perform a set of activities. You can find the list of PyCharm shortcuts in Keymaps guide reference.

## Finding Shortcut

The list of shortcuts is available in the following option **Help -> Find Action** menu where it pops up with a shortcut window.



You can see the shortcut window as shown here:



Enter action or option name: ☐ Include disabled actions Ctrl+Shift+A

iden

| | | |
|---|---|---|
| Identifiers//Parameter | Settings > Language Defaults | |
| Identifiers//Function call | Settings > Language Defaults | |
| Identifiers//Local variable | Settings > Language Defaults | |
| Identifiers//Global variable | Settings > Language Defaults | |
| Identifiers//Predefined symbol | Settings > Language Defaults | |
| Identifiers//Function declaration | Settings > Language Defaults | |
| View: Navigation Bar | | ON |
| Show Indent Guides | | ON |
| Show Already Unshelved | | OFF |
| Code//Identifier under caret | Settings > General | |
| Code//Identifier under caret (write) | Settings > General | |
| Select whole CSS identifiers on double click | Settings > Smart Keys | |
| Smart Keys: CSS: Select whole CSS identifiers | | ON |

Press Enter to toggle option

The shortcut includes a list of Identifiers, shortcuts with functions and option menu bar. For example, View Navigation Bar includes toggle ON and OFF which displays the navigation bar as per the value set (ON and OFF).

# 6. PyCharm – Omni

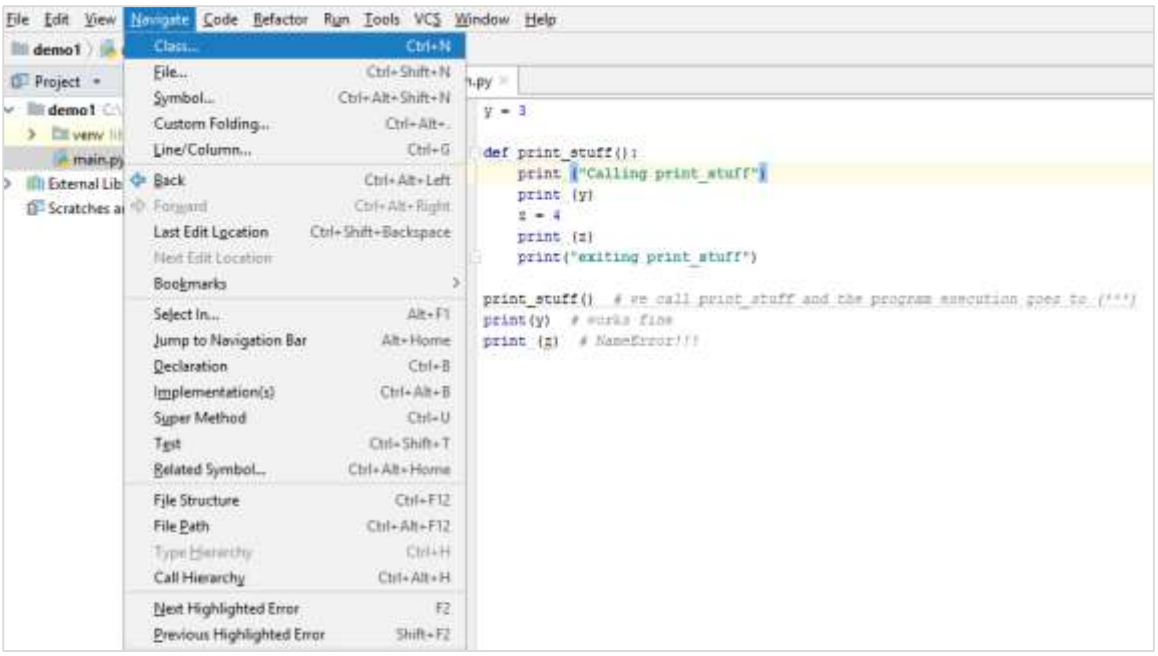
Omni is the section in PyCharm which deals into anywhere from any place. It includes various tools for a user to move from one place to another. It helps in such a scenario that you need to quickly move from one project directory into another. This chapter will familiarize you with the functionalities of Omni.

## Functionalities

The **Navigate** menu describes the functionalities involved in Omni. This section discusses these in detail:

### Class

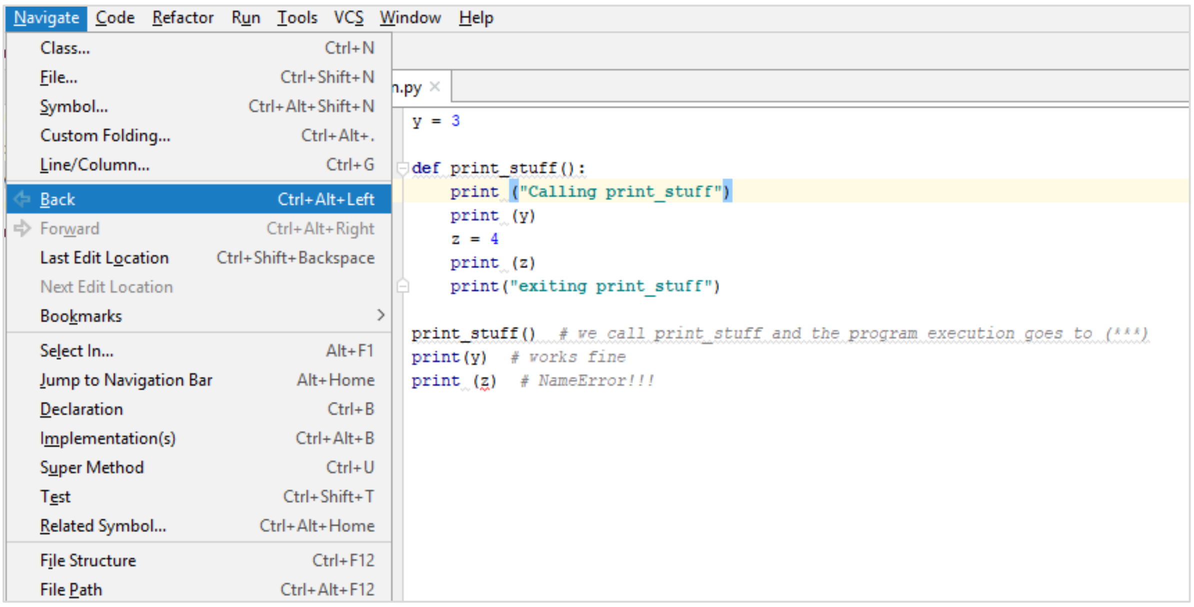
This helps to navigate from one class to another in a mentioned project. This is very helpful to navigate through a list of classes.


File Edit View Navigate Code Refactor Run Tools VCS Window Help
demo1
Project
demo1
venv
main.py
External Lib
Scratches a
Class... Ctrl+N
File... Ctrl+Shift+N
Symbol... Ctrl+Alt+Shift+N
Custom Folding... Ctrl+Alt+.
Line/Column... Ctrl+G
Back Ctrl+Alt+Left
Forward Ctrl+Alt+Right
Last Edit Location Ctrl+Shift+Backspace
Next Edit Location
Bookmarks
Select In... Alt+F1
Jump to Navigation Bar Alt+Home
Declaration Ctrl+B
Implementation(s) Ctrl+Alt+B
Super Method Ctrl+U
Test Ctrl+Shift+T
Related Symbol... Ctrl+Alt+Home
File Structure Ctrl+F12
File Path Ctrl+Alt+F12
Type Hierarchy Ctrl+H
Call Hierarchy Ctrl+Alt+H
Next Highlighted Error F2
Previous Highlighted Error Shift+F2
n.py

```
y = 3

def print_stuff():
    print ("Calling print_stuff")
    print (y)
    z = 4
    print (z)
    print("exiting print_stuff")

print_stuff()  # we call print_stuff and the program execution goes to (***)
print(y)  # works fine
print (z)  # NameError!!!
```

## Back

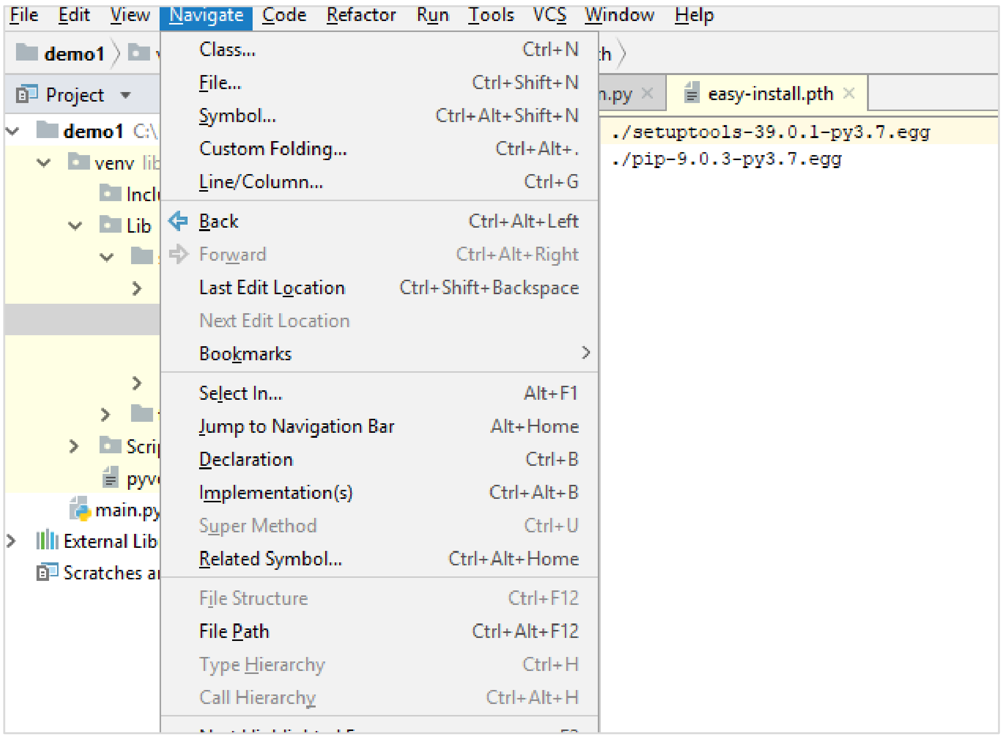
This option helps to move backwards from the existing state. The shortcut key is **Ctrl+Alt+Left.**



```
y = 3

def print_stuff():
    print ("Calling print_stuff")
    print (y)
    z = 4
    print (z)
    print("exiting print_stuff")

print_stuff()  # we call print_stuff and the program execution goes to (***)
print(y)  # works fine
print (z)  # NameError!!!
```

## Forward

It works similar to the **back** option. However, the functionality is completely vice-versa.



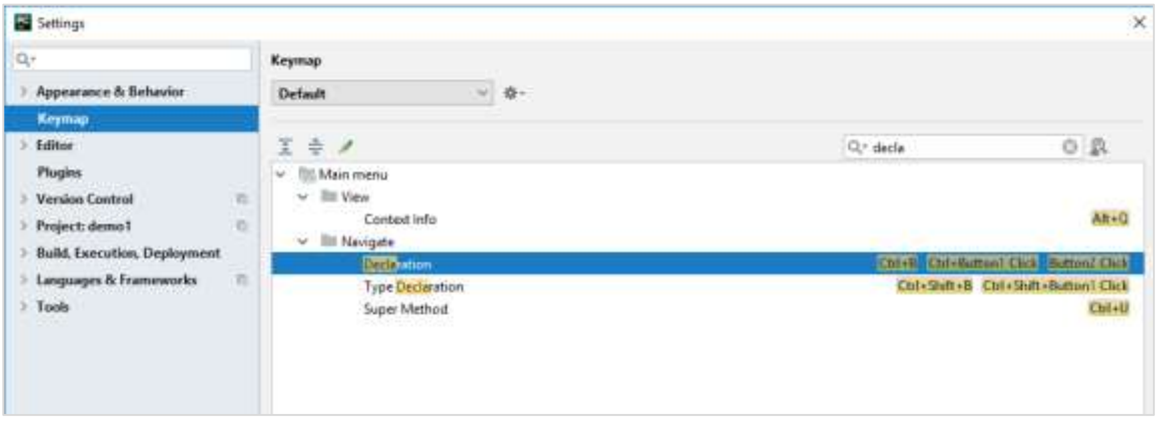
```
./setuptools-39.0.1-py3.7.egg
./pip-9.0.3-py3.7.egg
```

# 7. PyCharm – Macros

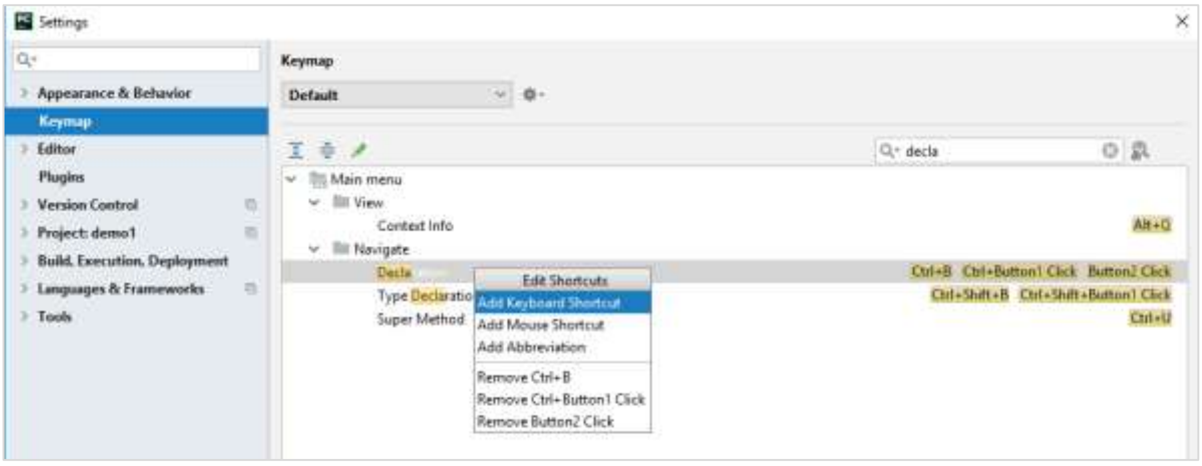
The difference between a macro and Omni is subtle in PyCharm Editor. Omni allows you to go to the exact location of editor or a specified place of code with no particular significance. Macro on the other hand allows the user to navigate through functions and classes or particular class method.

## Navigate Macro

Observe the following screenshot for a better understanding of Navigate macro:



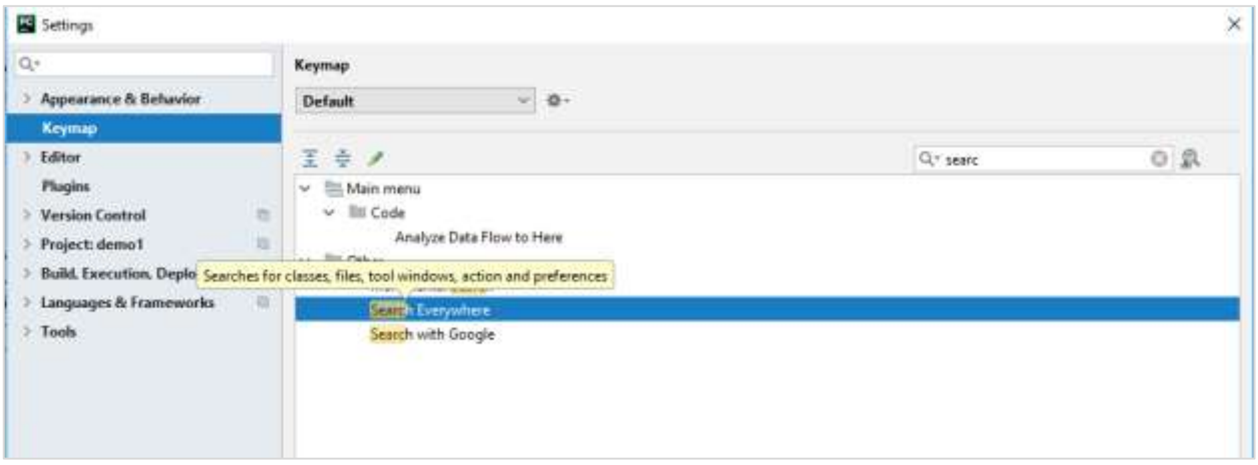
The **Navigate -> Declaration** helps to show declaration, type declaration and to define super methods. Various attributes included in the type declaration are shown below:



However, there is an issue with this macro, if a user tries to go to the declaration of a **.so** object for example, navigating from **datetime** module to **select** module, then each time it will encounter the **stub** file.

## Search Everywhere

It helps to search the classes and associated methods. It includes the option to search with Google as well.



Each of these parts includes a shortcut key combination next to its section name. **Search Everywhere** is a gateway to other search actions available in PyCharm.

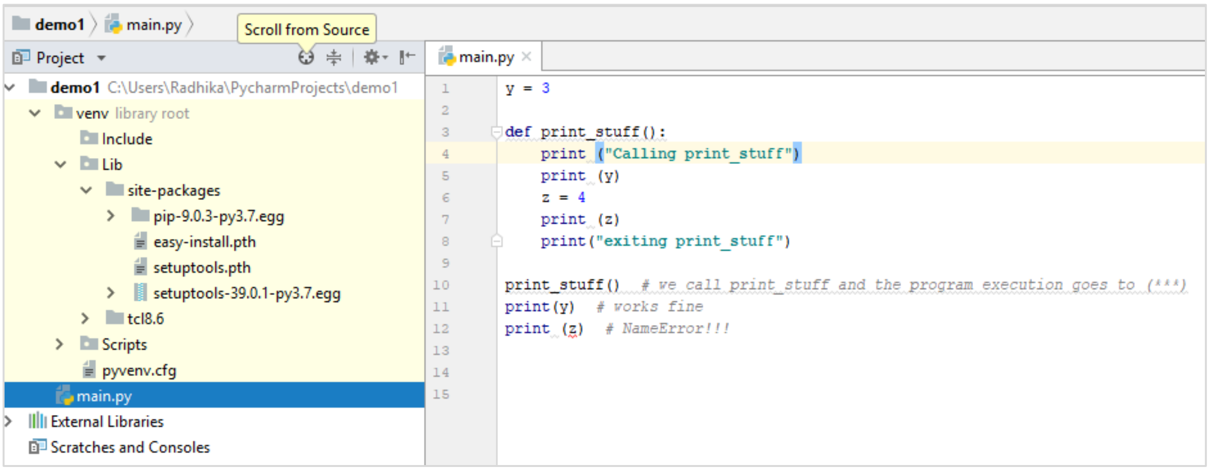
# 8. PyCharm – Micros

Micros deal with getting places within a specified file. These tools end up using most of the development procedure. In this chapter, you will learn Micro in detail.

Consider the example of **Structure Panel** which is being used as representation of micros.

## Scroll from Source

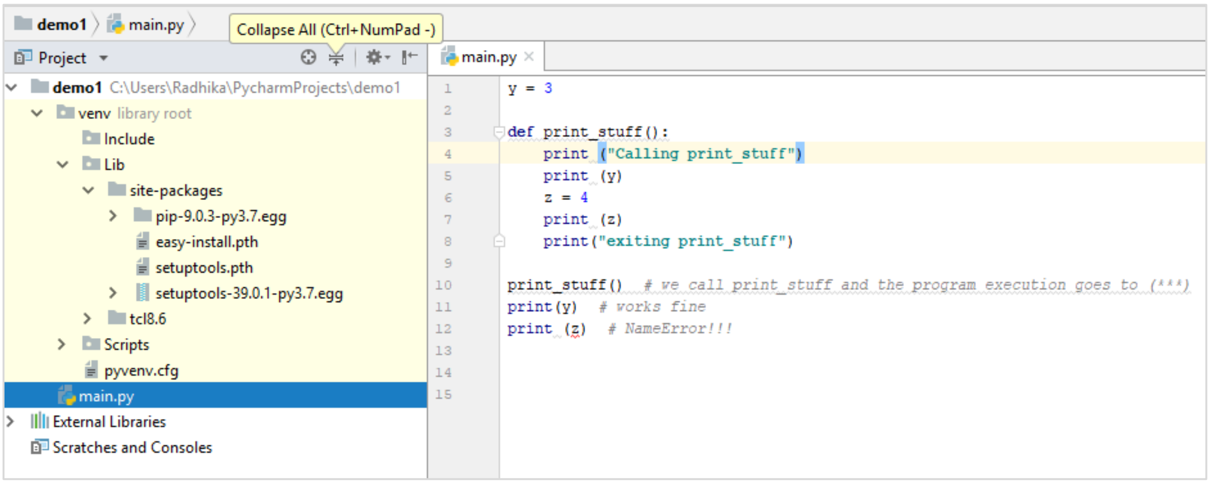
It helps to scroll from the mentioned source like the complete folder location of the specified file.



```
y = 3

def print_stuff():
    print ("Calling print_stuff")
    print (y)
    z = 4
    print (z)
    print("exiting print_stuff")

print_stuff()   # we call print_stuff and the program execution goes to (***)
print(y)   # works fine
print (z)   # NameError!!!
```

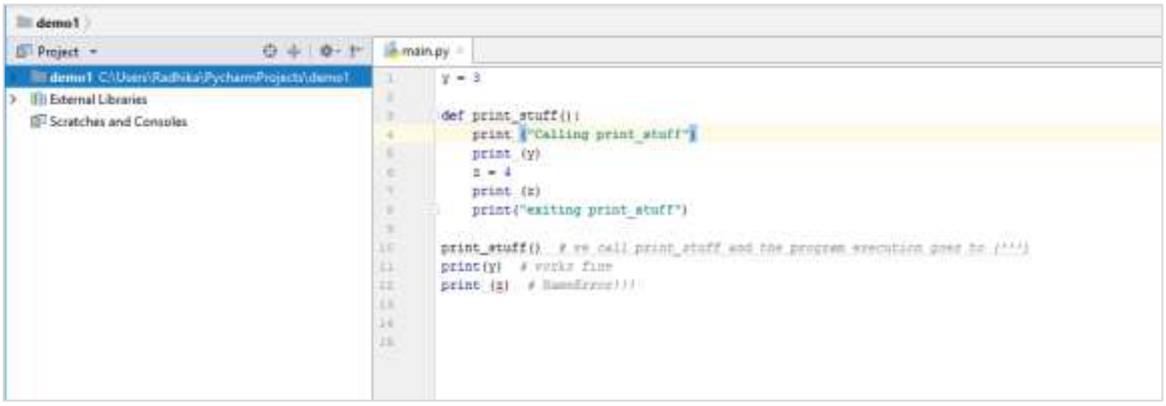
## Collapse All

Consider the screenshot shown below which shows opening the file with specified location. In order to collapse the folder structure, you can use the shortcut key shown in the image.



```
y = 3

def print_stuff():
    print ("Calling print_stuff")
    print (y)
    z = 4
    print (z)
    print("exiting print_stuff")

print_stuff()   # we call print_stuff and the program execution goes to (***)
print(y)   # works fine
print (z)   # NameError!!!
```

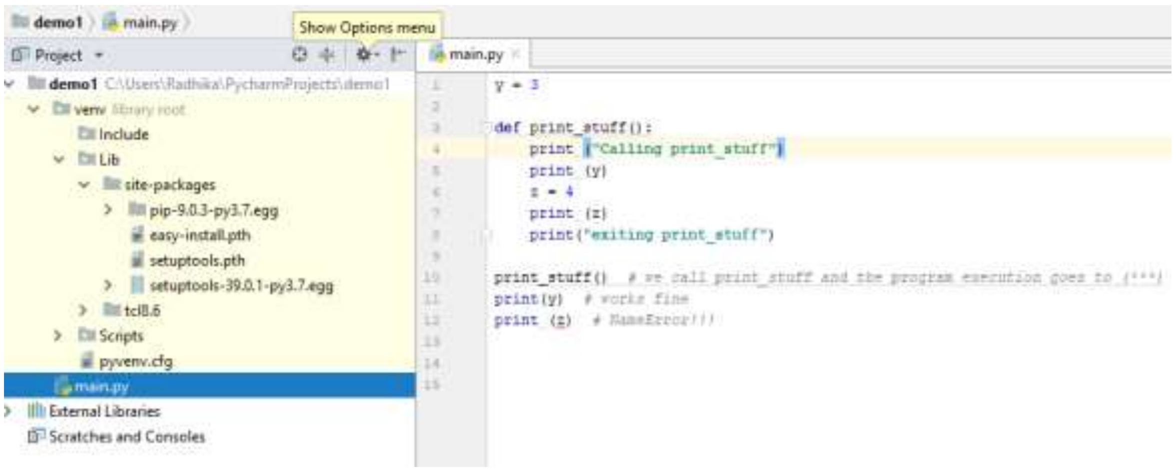
This shortcut key helps in collapsing the folder location of specified code as shown below.



```
y = 3

def print_stuff():
    print ("Calling print_stuff")
    print (y)
    z = 4
    print (z)
    print("exiting print_stuff")

print_stuff()  # we call print_stuff and the program execution goes to (***)
print(y)  # works fine
print (z)  # NameError!!!
```

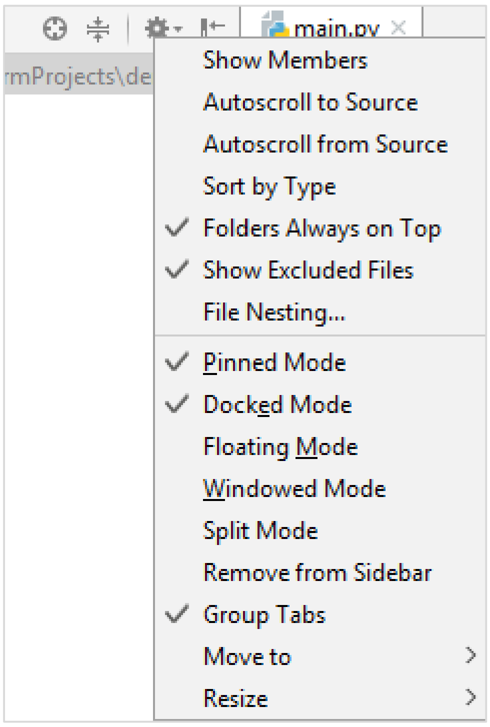
## Show Options menu

The **Show Options** menu of the structure panel of project displays the list of options available for the project created. Observe the screenshot shown below for a better understanding:



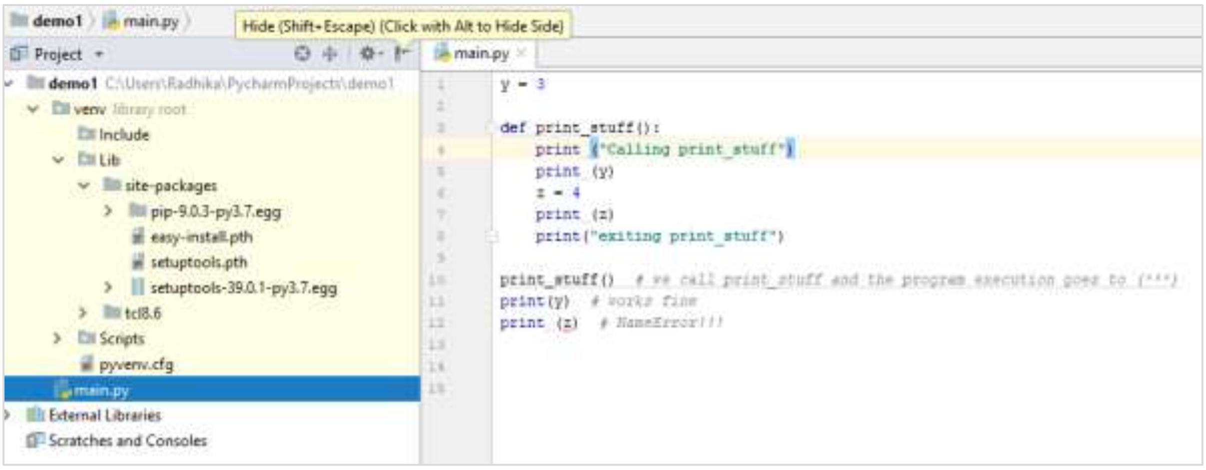
```
y = 3

def print_stuff():
    print ("Calling print_stuff")
    print (y)
    z = 4
    print (z)
    print("exiting print_stuff")

print_stuff()  # we call print_stuff and the program execution goes to (***)
print(y)  # works fine
print (z)  # NameError!!!
```

The list of options is displayed below:

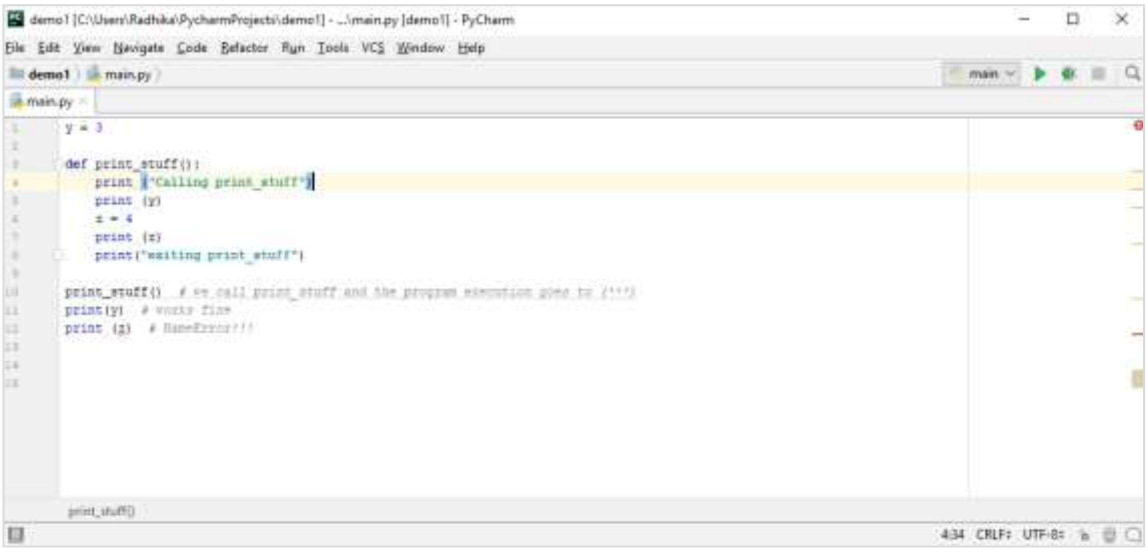


## Hide

This option helps to hide the structure panel of the project window. The user interface of the structure panel after being collapsed is as shown below:

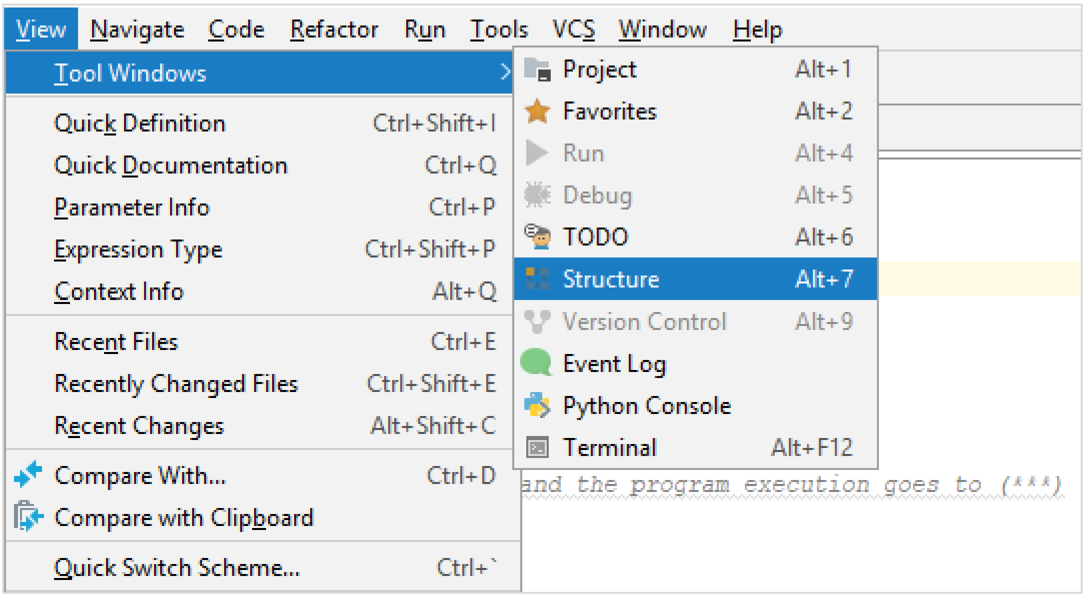


```
y = 3

def print_stuff():
    print ("Calling print_stuff")
    print (y)
    z = 4
    print (z)
    print("exiting print_stuff")

print_stuff()  # we call print_stuff and the program execution goes to (***)
print(y)  # works fine
print (z)  # NameError!!!
```



```
y = 3

def print_stuff():
    print ("Calling print_stuff")
    print (y)
    z = 4
    print (z)
    print("exiting print_stuff")

print_stuff()  # we call print_stuff and the program execution goes to (***)
print(y)  # works fine
print (z)  # NameError!!!
```

You can reopen the structure panel as shown here:



```
and the program execution goes to (***)
```

# 9. PyCharm – Improving and Writing Code

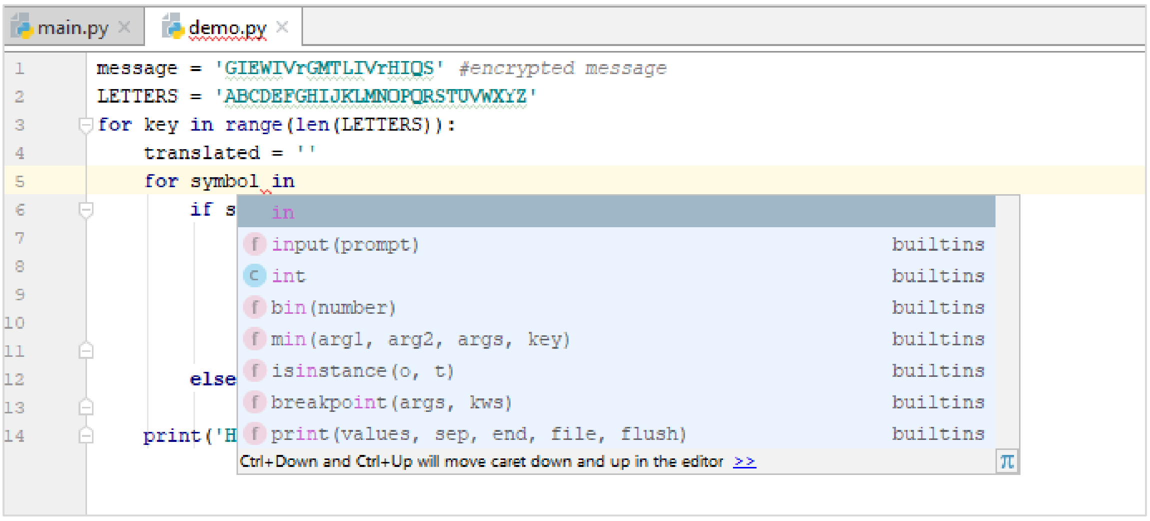
PyCharm includes various standards for writing code with proper indentations valid for Python. This makes it interesting to improve the code standards and writing the complete code in PyCharm editor.

## Improving Code Completion

Code completion in PyCharm is really unique. You can enhance it further using many other features. Note that the editor provides start and end of the code block. Consider a file named **demo.py** with the following code:

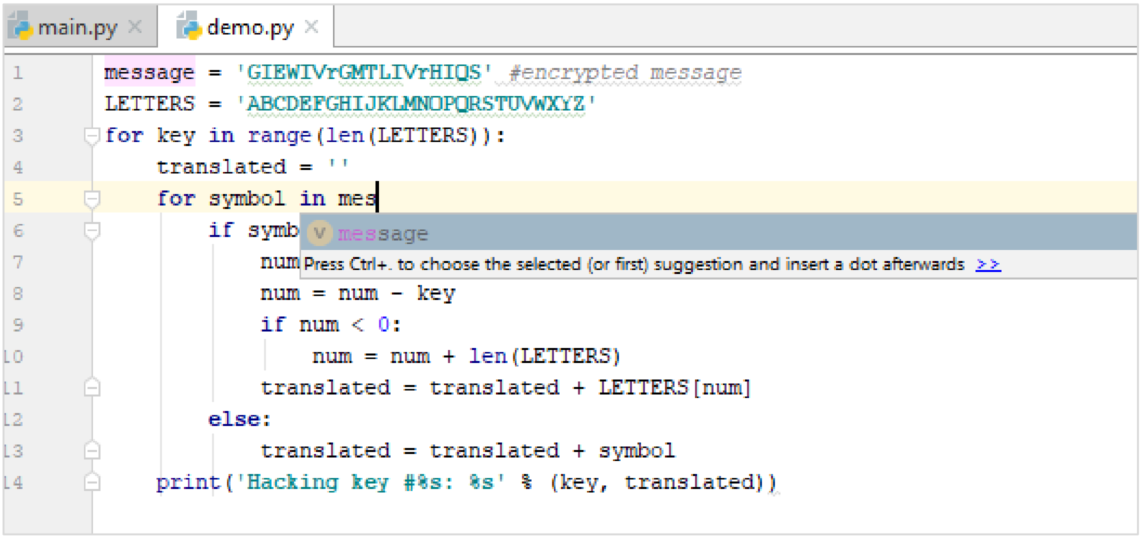
```
message = 'GIEWIVrGMTLIVrHIQS' #encrypted message
LETTERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
for key in range(len(LETTERS)):
    translated = ''
    for symbol in message:
        if symbol in LETTERS:
            num = LETTERS.find(symbol)
            num = num - key
            if num < 0:
                num = num + len(LETTERS)
            translated = translated + LETTERS[num]
        else:
            translated = translated + symbol
    print('Hacking key #%s: %s' % (key, translated))
```

The code is completed using the following construct:



```
message = 'GIEWIVrGMTLIVrHIQS' #encrypted message
LETTERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
for key in range(len(LETTERS)):
    translated = ''
    for symbol in
        if s in
             input(prompt)                                   builtins
             int                                             builtins
             bin(number)                                     builtins
             min(arg1, arg2, args, key)                      builtins
        else isinstance(o, t)                                builtins
             breakpoint(args, kws)                           builtins
    print('H print(values, sep, end, file, flush)            builtins
Ctrl+Down and Ctrl+Up will move caret down and up in the editor >>
```

If you press Ctrl + spacebar while this popup is on the screen, you can see more code completion options:

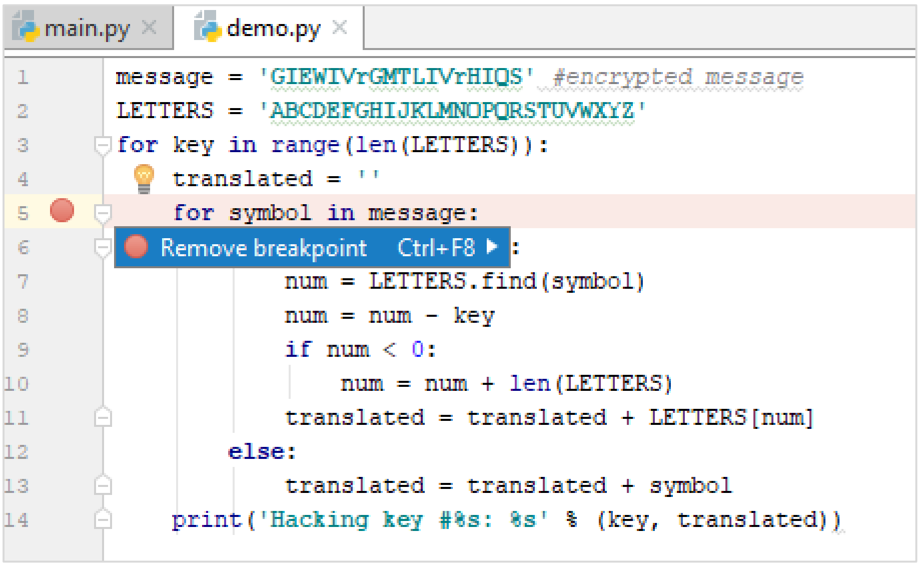


```
message = 'GIEWIVrGMTLIVrHIQS' #encrypted message
LETTERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
for key in range(len(LETTERS)):
    translated = ''
    for symbol in mes
        if symb message
            num Press Ctrl+. to choose the selected (or first) suggestion and insert a dot afterwards >>
            num = num - key
            if num < 0:
                num = num + len(LETTERS)
            translated = translated + LETTERS[num]
        else:
            translated = translated + symbol
    print('Hacking key #%s: %s' % (key, translated))
```

## Intention Actions

PyCharm includes intent specific actions and the shortcut key for the same is **Alt+Enter**. The most important example of intentions at work is using language injection in strings.

The screenshot given below shows the working of intention actions:

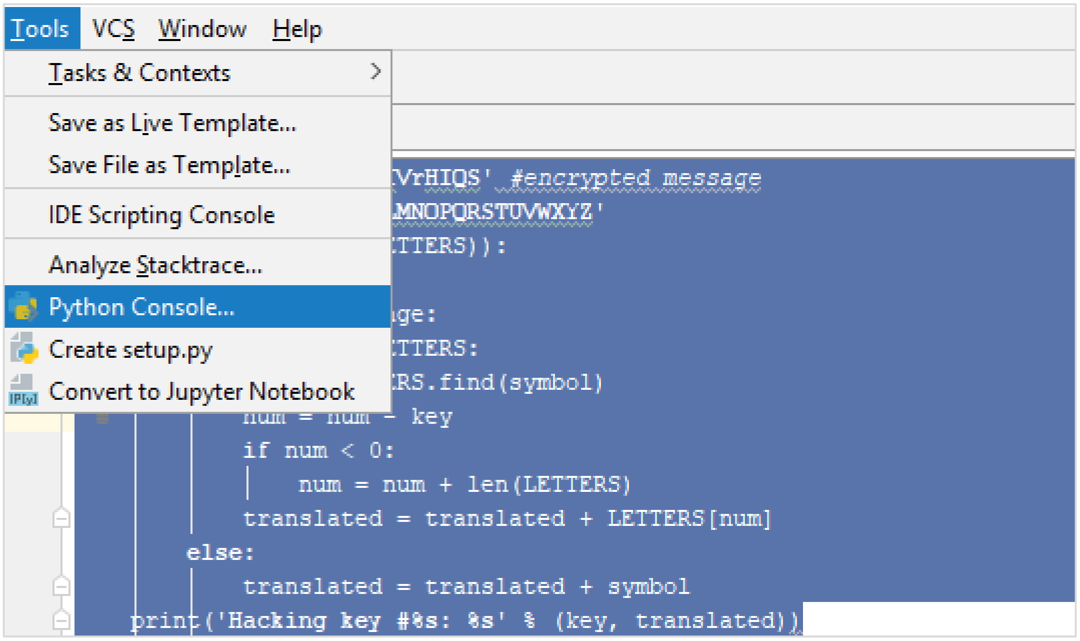


```
message = 'GIEWIVrGMTLIVrHIQS' #encrypted message
LETTERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
for key in range(len(LETTERS)):
    translated = ''
    for symbol in message:
                                :
            num = LETTERS.find(symbol)
            num = num - key
            if num < 0:
                num = num + len(LETTERS)
            translated = translated + LETTERS[num]
        else:
            translated = translated + symbol
    print('Hacking key #%s: %s' % (key, translated))
```

Note that we can insert many different languages of intention actions in PyCharm Editor.

# 10. PyCharm – Console

PyCharm has a full-fledged Python console with full code completion which is available in the option menu **Tools -> Run Python Console**.

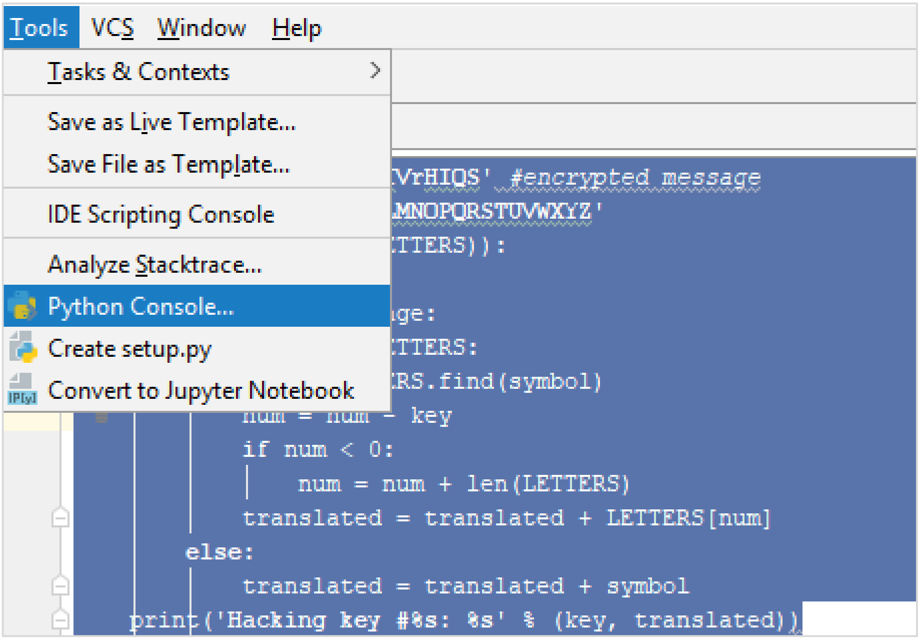


```
VrHIQS' #encrypted message
MNOPQRSTUVWXYZ'
TTERS)):

ge:
TTERS:
RS.find(symbol)
            num = num - key
            if num < 0:
                num = num + len(LETTERS)
            translated = translated + LETTERS[num]
        else:
            translated = translated + symbol
    print('Hacking key #%s: %s' % (key, translated))
```

Consider the code which was mentioned in the previous chapter, as shown below:

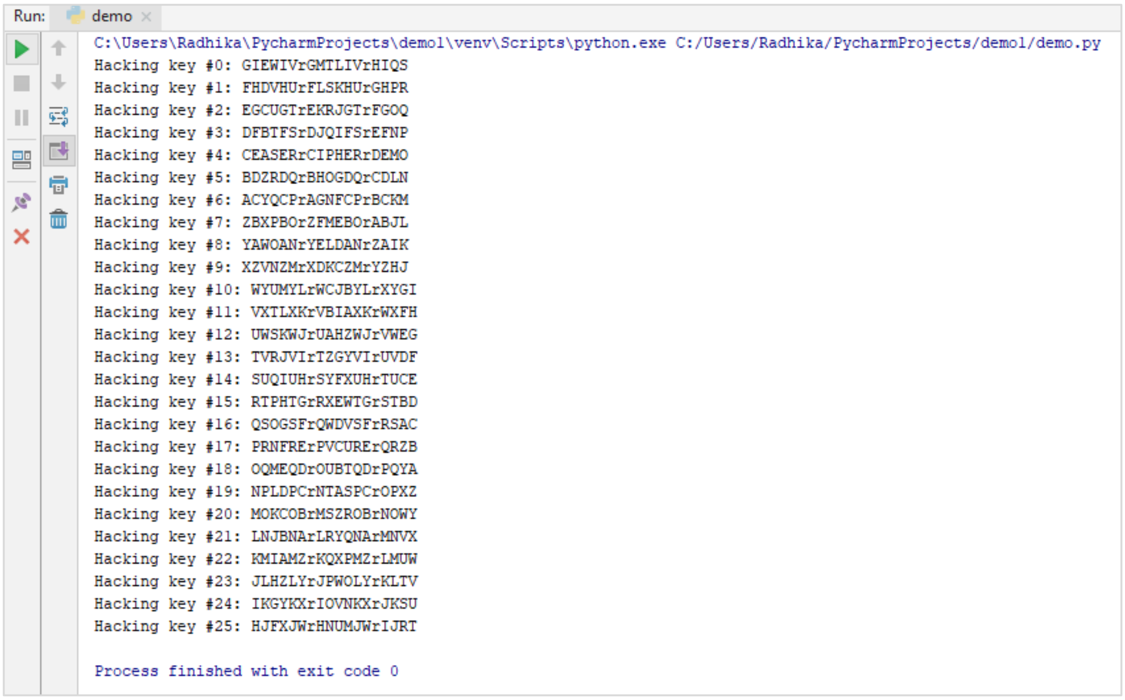
```
message = 'GIEWIVrGMTLIVrHIQS' #encrypted message
LETTERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
for key in range(len(LETTERS)):
    translated = ''
    for symbol in message:
        if symbol in LETTERS:
            num = LETTERS.find(symbol)
            num = num - key
            if num < 0:
                num = num + len(LETTERS)
            translated = translated + LETTERS[num]
        else:
            translated = translated + symbol
    print('Hacking key #%s: %s' % (key, translated))
```

Now, let us run the code with the help of console to execute the script for getting the desired output, as shown below.



```
VrHIQS' #encrypted message
MNOPQRSTUVWXYZ'
TTERS)):

ge:
TTERS:
RS.find(symbol)
        num = num - key
        if num < 0:
            num = num + len(LETTERS)
        translated = translated + LETTERS[num]
    else:
        translated = translated + symbol
print('Hacking key #%s: %s' % (key, translated))
```

You can observe the output as shown below:



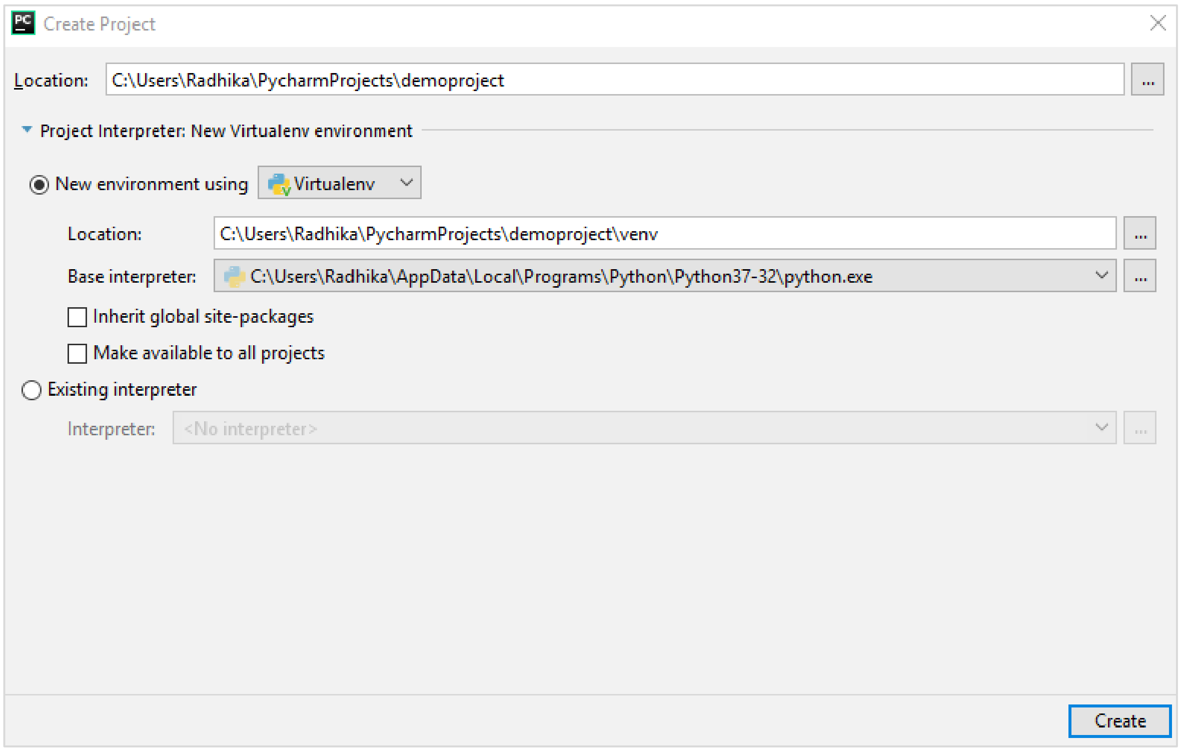
```
C:\Users\Radhika\PycharmProjects\demo1\venv\Scripts\python.exe C:/Users/Radhika/PycharmProjects/demo1/demo.py
Hacking key #0: GIEWIVrGMTLIVrHIQS
Hacking key #1: FHDVHUrFLSKHUrGHPR
Hacking key #2: EGCUGTrEKRJGTrFGOQ
Hacking key #3: DFBTFSrDJQIFSrEFNP
Hacking key #4: CEASERrCIPHERrDEMO
Hacking key #5: BDZRDQrBHOGDQrCDLN
Hacking key #6: ACYQCPrAGNFCPrBCKM
Hacking key #7: ZBXPBOrZFMEBOrABJL
Hacking key #8: YAWOANrYELDANrZAIK
Hacking key #9: XZVNZMrXDKCZMrYZHJ
Hacking key #10: WYUMYLrWCJBYLrXYGI
Hacking key #11: VXTLXKrVBIAXKrWXFH
Hacking key #12: UWSKWJrUAHZWJrVWEG
Hacking key #13: TVRJVIrTZGYVIrUVDF
Hacking key #14: SUQIUHrSYFXUHrTUCE
Hacking key #15: RTPHTGrRXEWTGrSTBD
Hacking key #16: QSOGSFrQWDVSFrRSAC
Hacking key #17: PRNFRErPVCURErQRZB
Hacking key #18: OQMEQDrOUBTQDrPQYA
Hacking key #19: NPLDPCrNTASPCrOPXZ
Hacking key #20: MOKCOBrMSZROBrNOWY
Hacking key #21: LNJBNArLRYQNArMNVX
Hacking key #22: KMIAMZrKQXPMZrLMUW
Hacking key #23: JLHZLYrJPWOLYrKLTV
Hacking key #24: IKGYKXrIOVNKXrJKSU
Hacking key #25: HJFXJWrHNUMJWrIJRT

Process finished with exit code 0
```

# 11. PyCharm – Interpreters

PyCharm includes interpreters to create a new project with new features as the way it is needed. You can create a virtual environment in your system as the way you need it. You can also inherit global site packages in the dialog box. Interpreters are available on Python Package Index (PyPI) and can be easily installed and accessed using **pip install**.

## Creation of Interpreter

For creating an interpreter, it is always recommended to create a new project where desired configurations are managed. Look at the following screenshot for a better understanding:



Location: C:\Users\Radhika\PycharmProjects\demoproject

Project Interpreter: New Virtualenv environment

[✓ New environment using] Virtualenv

Location: C:\Users\Radhika\PycharmProjects\demoproject\venv

Base interpreter: C:\Users\Radhika\AppData\Local\Programs\Python\Python37-32\python.exe

☐ Inherit global site-packages

☐ Make available to all projects

○ Existing interpreter

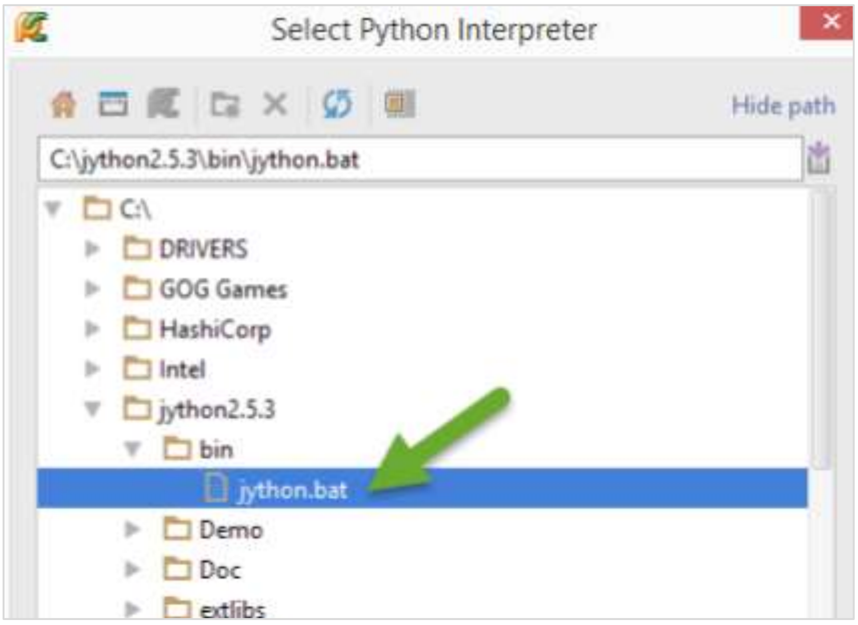
Interpreter: <No interpreter>

These parameters include:

- Location: This describes the parameter where virtual environment is created focusing on the location on system.

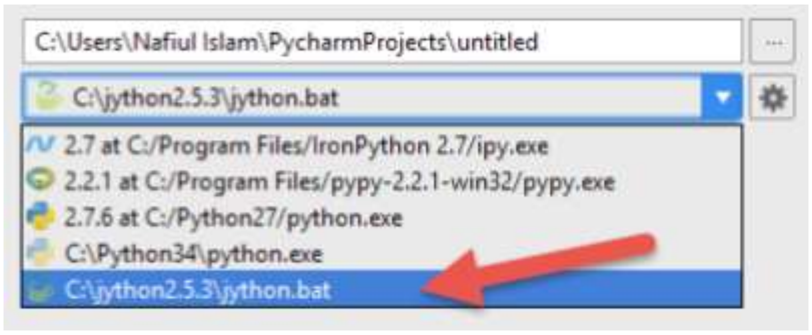
- Basic interpreter: It defines the attributes of interpreter.

The dialog box also refers to the parameter where an existing virtual interpreter will be taken as an attribute. Once the user adds a new local interpreter, PyCharm will ask the user for the binary of interpreter. In most cases, it is always considered to be a **.exe** file. In case of Jython, it will be always a **.bat** file.



C:\jython2.5.3\bin\jython.bat

```
C:\
  DRIVERS
  GOG Games
  HashiCorp
  Intel
  jython2.5.3
    bin
      jython.bat
    Demo
    Doc
    extlibs
```

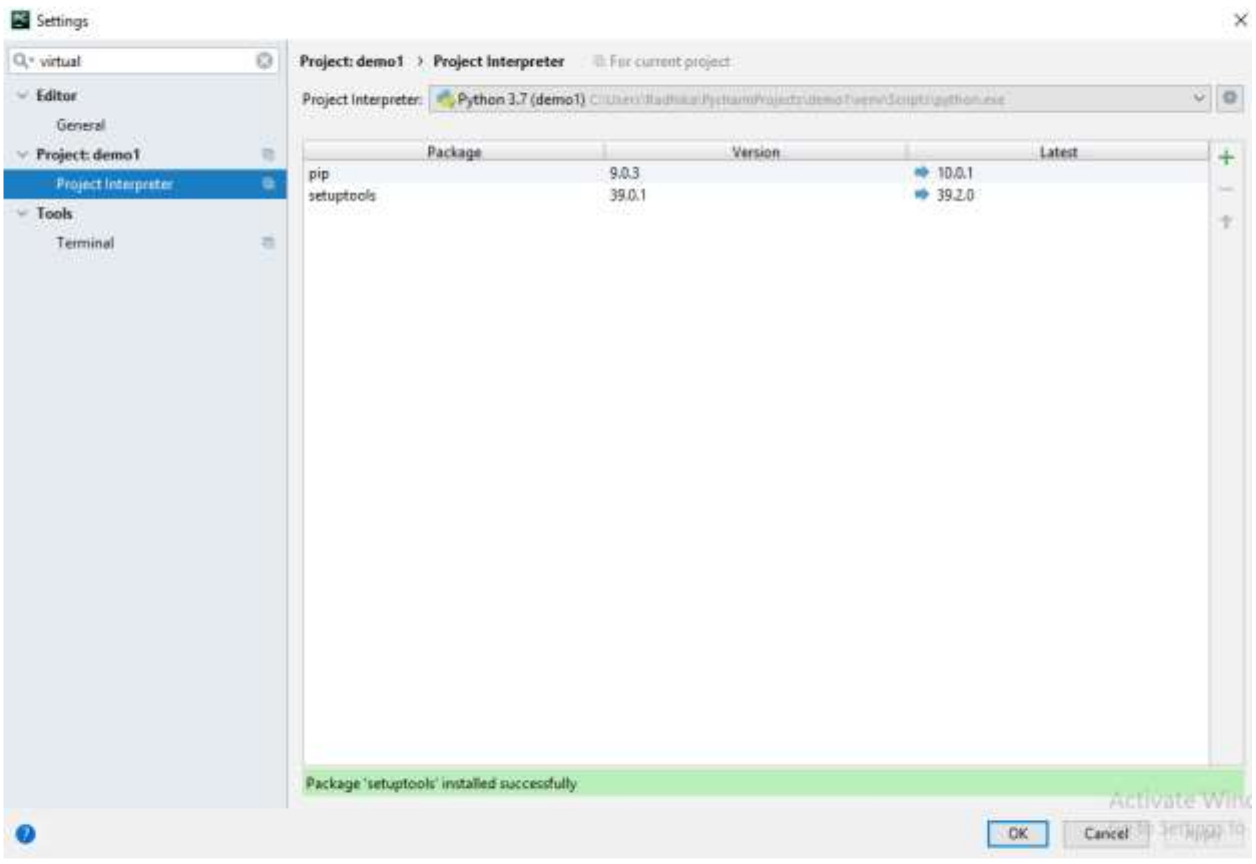


C:\Users\Nafiul Islam\PycharmProjects\untitled

C:\jython2.5.3\jython.bat

```
2.7 at C:/Program Files/IronPython 2.7/ipy.exe
2.2.1 at C:/Program Files/pypy-2.2.1-win32/pypy.exe
2.7.6 at C:/Python27/python.exe
C:\Python34\python.exe
C:\jython2.5.3\jython.bat
```

The details of Project Interpreter and the basic configuration of the existing project **demo1** can be seen as shown below:



Project Interpreter: Python 3.7 (demo1)

| Package | Version | Latest |
|---|---|---|
| pip | 9.0.3 | 10.0.1 |
| setuptools | 39.0.1 | 39.2.0 |

Package 'setuptools' installed successfully

Remember that the interpreter also includes the basic packages which are mandatory for smooth functioning of working of project.

# 12. PyCharm – Debugging and Breakpoints

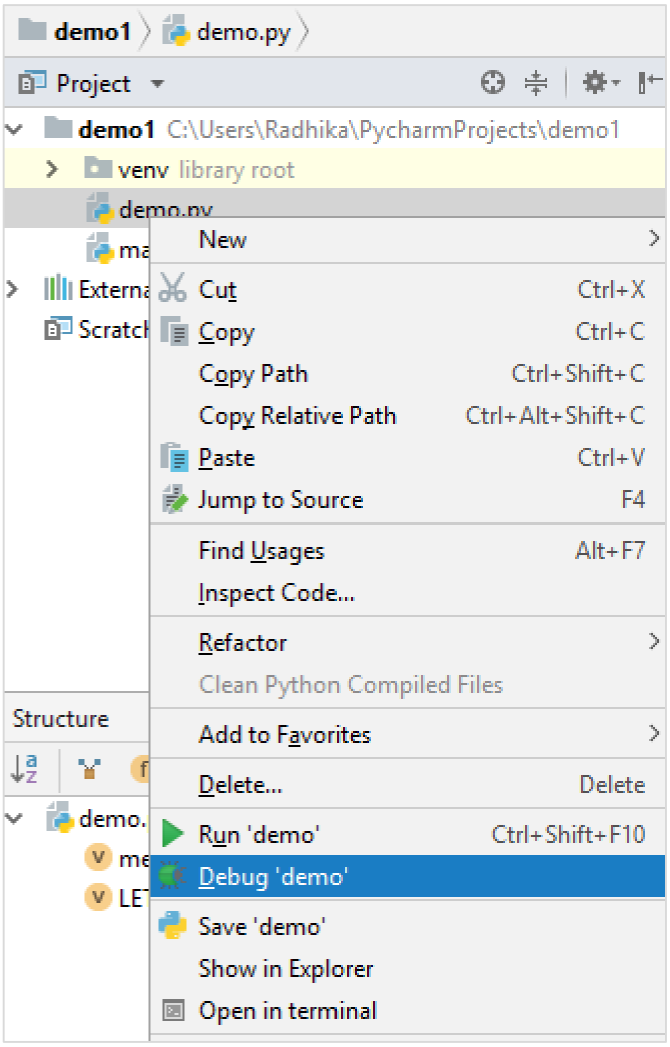
Running a python code comprises of two modes: running a script and debugging the script. This chapter focusses on debugging the Python script using PyCharm.

## Steps Involved

The steps for debugging the Python project are as explained below:

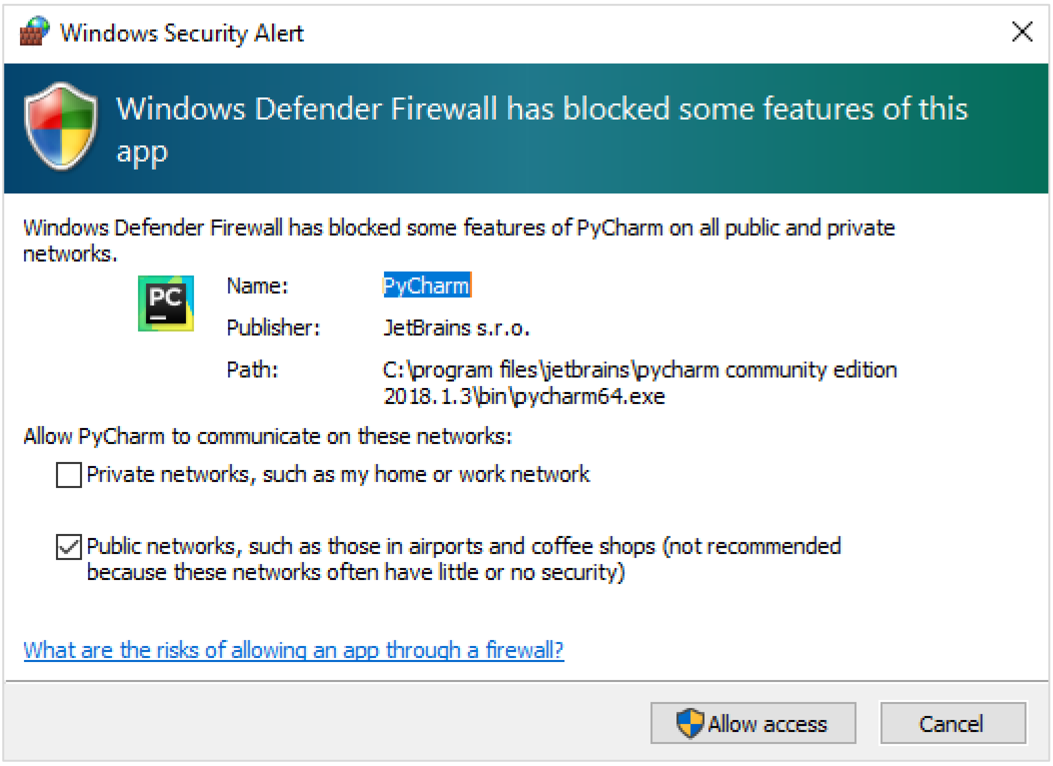
### Step 1

Start with debugging the Python project as shown in the screenshot below:



## Step 2

Now, Windows firewall asks permission for debugging the Python project as the procedure involves line by line compilation.



Windows Security Alert

Windows Defender Firewall has blocked some features of this app

Windows Defender Firewall has blocked some features of PyCharm on all public and private networks.

| | |
|---|---|
| Name: | PyCharm |
| Publisher: | JetBrains s.r.o. |
| Path: | C:\program files\jetbrains\pycharm community edition 2018.1.3\bin\pycharm64.exe |

Allow PyCharm to communicate on these networks:

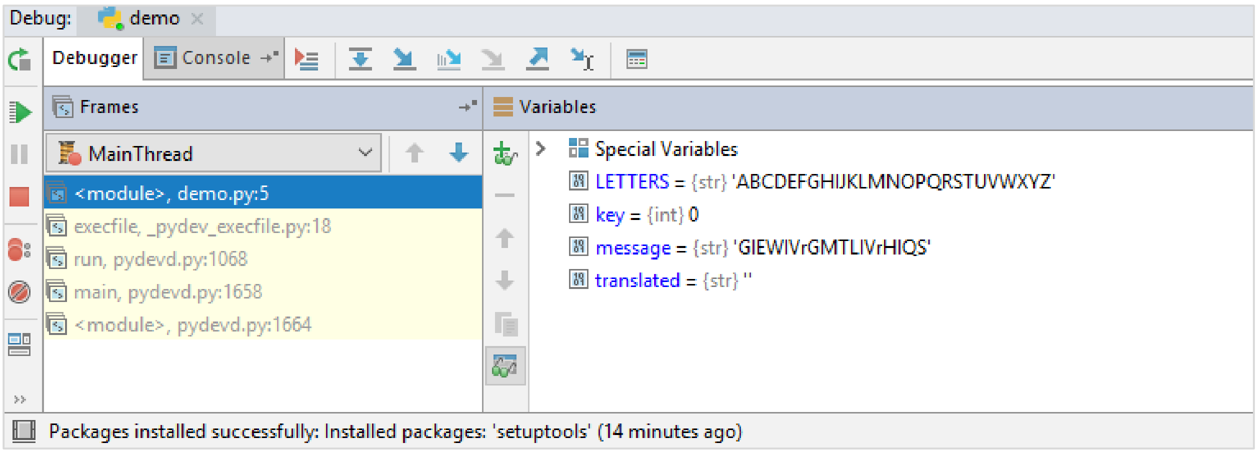
☐ Private networks, such as my home or work network

☑ Public networks, such as those in airports and coffee shops (not recommended because these networks often have little or no security)

What are the risks of allowing an app through a firewall?

## Step 3

The debugging console is created in PyCharm editor as shown below which executes the output line by line.

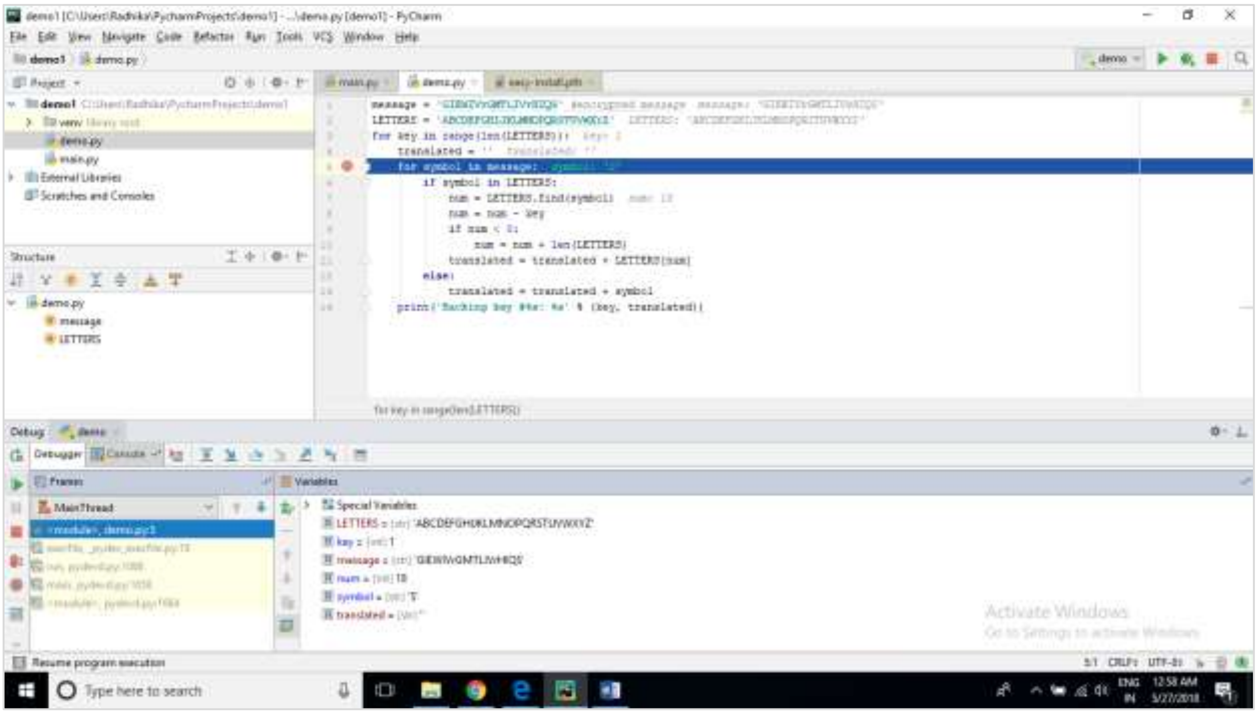


```
<module>, demo.py:5
execfile, _pydev_execfile.py:18
run, pydevd.py:1068
main, pydevd.py:1658
<module>, pydevd.py:1664
```

```
> Special Variables
  LETTERS = {str} 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
  key = {int} 0
  message = {str} 'GIEWIVrGMTLIVrHIQS'
  translated = {str} ''
```

Packages installed successfully: Installed packages: 'setuptools' (14 minutes ago)

The run button moves from one line to another to execute the output as the way we want.

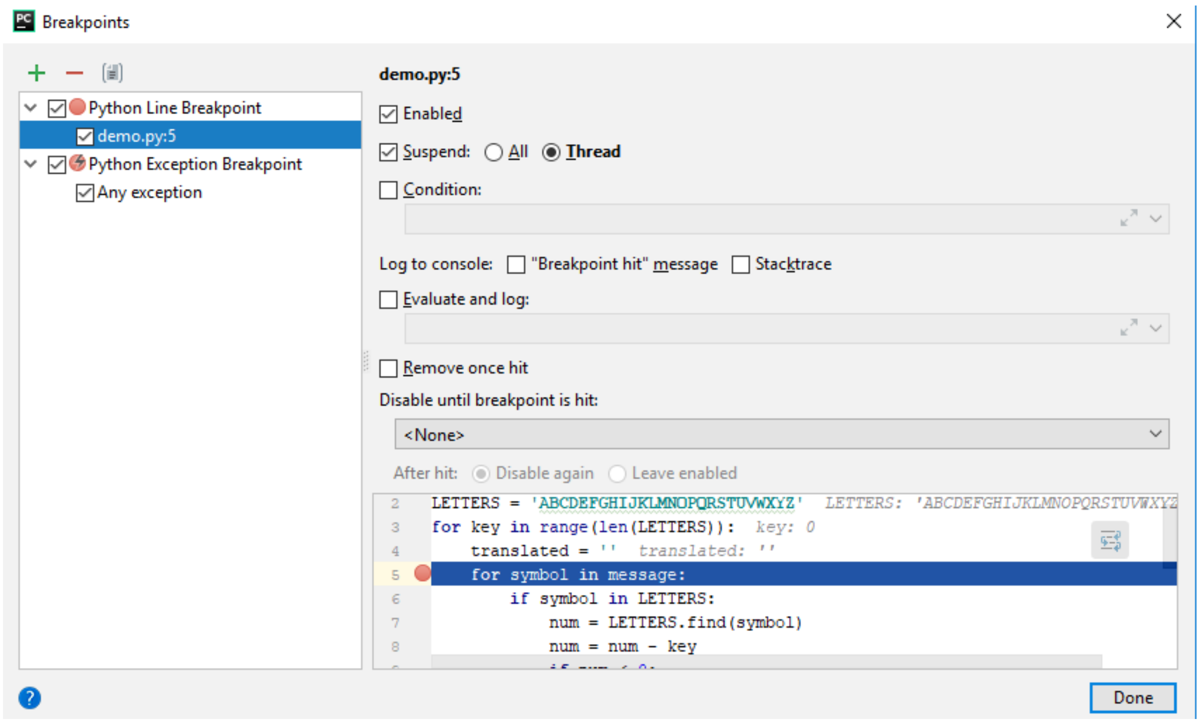


```
message = 'GIEWIVrGMTLIVrHIQS' #encrypted message  message: 'GIEWIVrGMTLIVrHIQS'
LETTERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'  LETTERS: 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
for key in range(len(LETTERS)):  key: 1
    translated = ''  translated: ''
    for symbol in message:  symbol: 'T'
        if symbol in LETTERS:
            num = LETTERS.find(symbol)  num: 19
            num = num - key
            if num < 0:
                num = num + len(LETTERS)
            translated = translated + LETTERS[num]
        else:
            translated = translated + symbol
    print('Hacking key #%s: %s' % (key, translated))
```

```
LETTERS = {str} 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
key = {int} 1
message = {str} 'GIEWIVrGMTLIVrHIQS'
num = {int} 18
symbol = {str} 'T'
translated = {str} ''
```

## Understanding Breakpoints

While debugging a particular script, it is intentional to create a breakpoint. Breakpoints are intentional stopping place or the place where the code is paused in order to identify the output at specific stage.



Breakpoints

- [✓] Python Line Breakpoint
  - [✓] demo.py:5
- [✓] Python Exception Breakpoint
  - [✓] Any exception

demo.py:5

[✓] Enabled

[✓] Suspend: ○ All [✓ ◉ Thread]

☐ Condition:

Log to console: ☐ "Breakpoint hit" message ☐ Stacktrace

☐ Evaluate and log:

☐ Remove once hit

Disable until breakpoint is hit: <None>

After hit: ◉ Disable again ○ Leave enabled

```
LETTERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'  LETTERS: 'ABCDEFGHIJKLMNOPQRSTUVWXYZ
for key in range(len(LETTERS)):  key: 0
    translated = ''  translated: ''
    for symbol in message:
        if symbol in LETTERS:
            num = LETTERS.find(symbol)
            num = num - key
```

Done

In PyCharm, breakpoints are visible using a separate dialog in the specified editor. It includes various attributes to evaluate the breakpoints defined and tracing log for the same with a main motive to achieve better programming practice.

# 13. PyCharm – Integration of Version Control

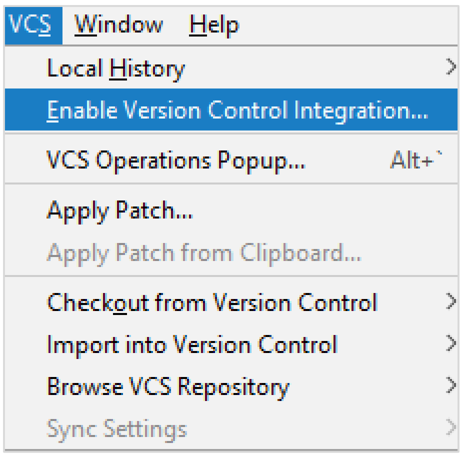
PyCharm supports various subversion control systems. This feature helps in improving the code base managing various versions together. This chapter talks about this concept in detail.

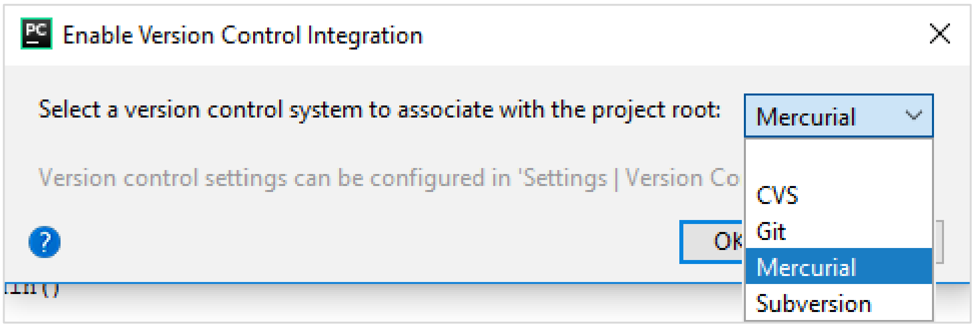
## Steps Involved

You will have to go through the following steps for initializing and managing version control system:

### Initializing a Subversion Control System

To start the version control system in a systematic way, it is important to initialize it. Various options are available in PyCharm for different version control systems.


VCS
Window
Help
Local History
Enable Version Control Integration...
VCS Operations Popup... Alt+`
Apply Patch...
Apply Patch from Clipboard...
Checkout from Version Control
Import into Version Control
Browse VCS Repository
Sync Settings

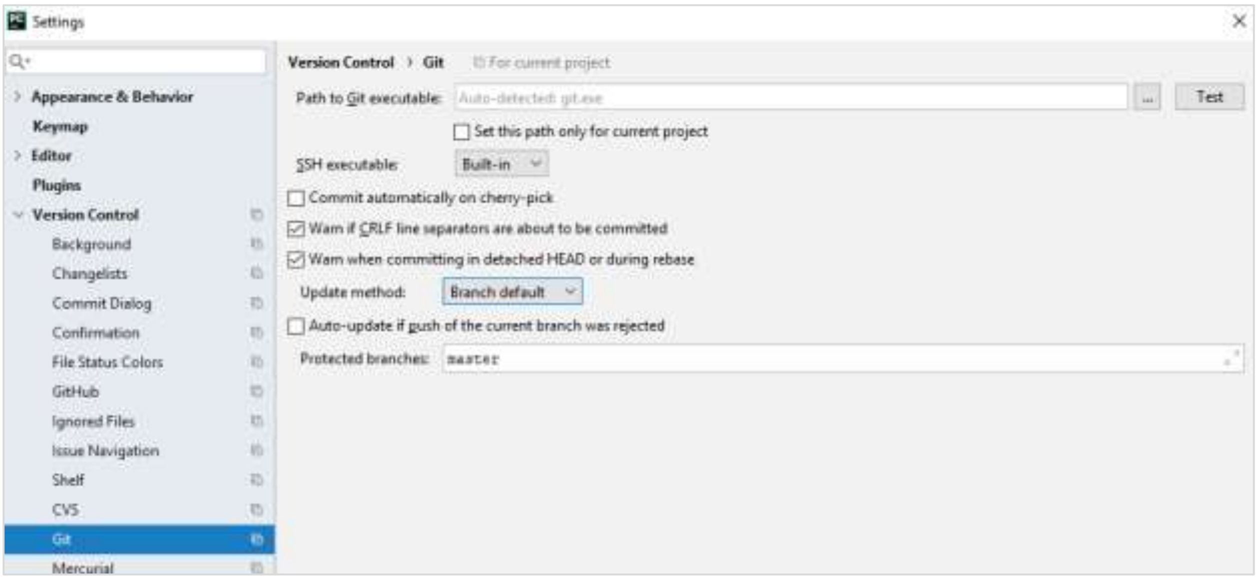

Enable Version Control Integration

Select a version control system to associate with the project root: Mercurial

Version control settings can be configured in 'Settings | Version Co

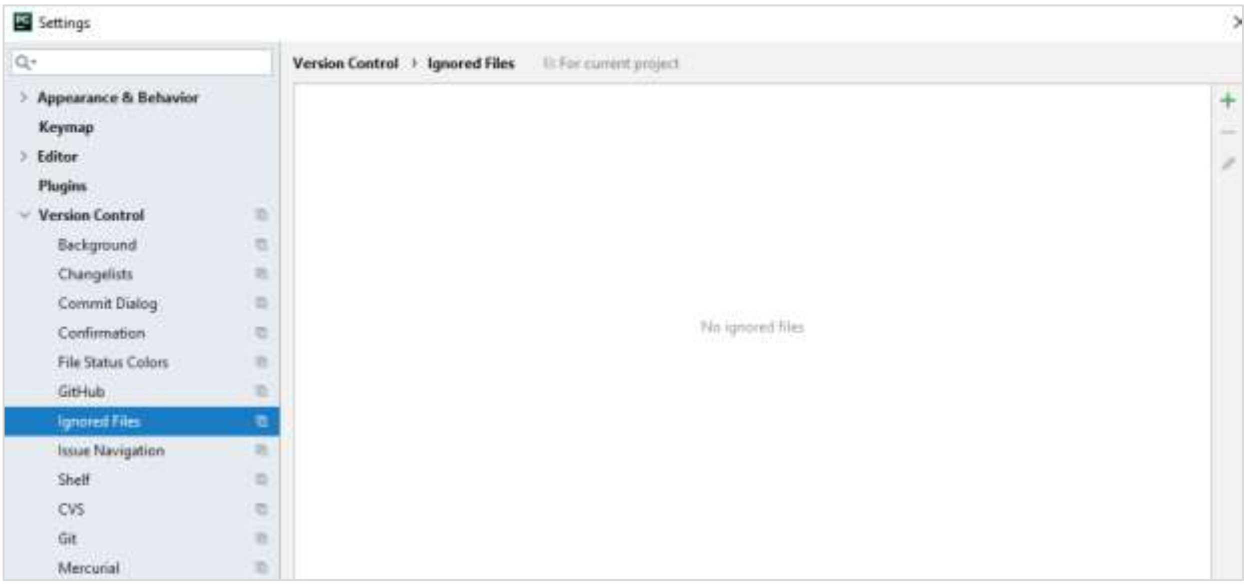
### Ignoring File

In any project of PyCharm where we set up the default project and the virtual environment on it, we should also create its management with version control system. For example, Git includes **.gitignore** files which are ignored during commit operation, however, includes some of its configurations. Now, go to the Settings menu and check for the following:



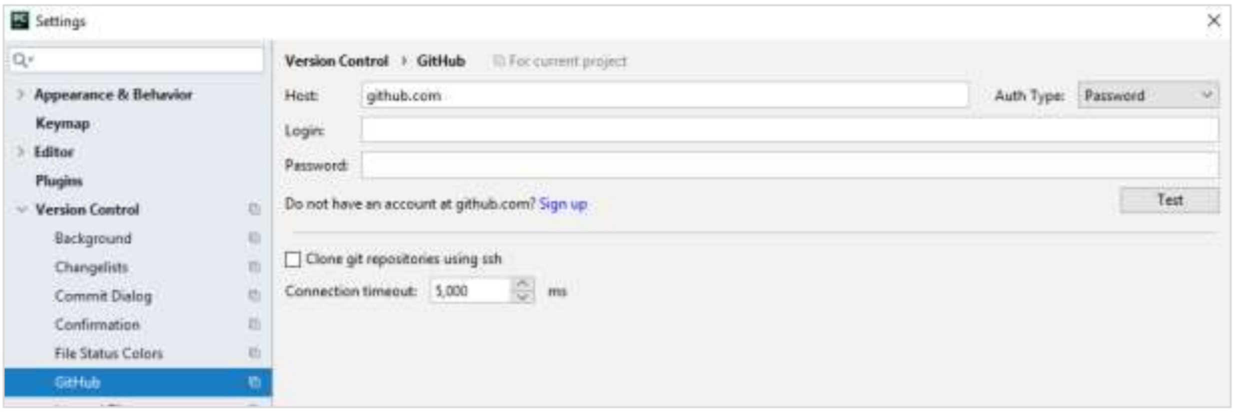
Path to Git executable: Auto-detected: git.exe
☐ Set this path only for current project
SSH executable: Built-in
☐ Commit automatically on cherry-pick
☑ Warn if CRLF line separators are about to be committed
☑ Warn when committing in detached HEAD or during rebase
Update method: Branch default
☐ Auto-update if push of the current branch was rejected
Protected branches: master

It includes various configurations for checking the path of Git executable and verifying if any files are ignored.



## Configuration of GitHub

PyCharm includes settings to include configuration of GitHub repository, where a user can include username, password and other credentials, if any.



Settings

Version Control › GitHub   For current project

Host: github.com   Auth Type: Password

Login:

Password:

Do not have an account at github.com? Sign up   Test

☐ Clone git repositories using ssh

Connection timeout: 5,000 ms

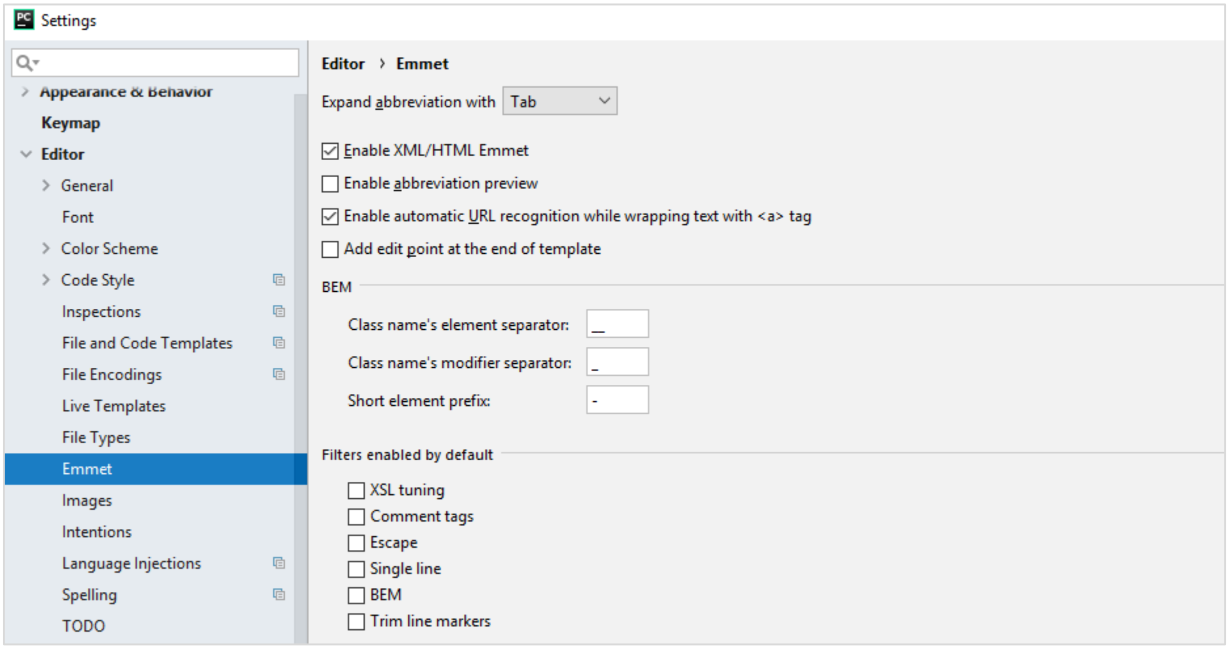
Once you are done with the settings mentioned, you can directly add and commit the local changes to Git repository.

# 14. PyCharm – HTML and CSS Integration

HTML and CSS are well supported in PyCharm Editor. PyCharm Editor includes a special shorthand and provides tag completion for HTML.

## Emmet

Emmet is the shorthand used in PyCharm editor. It includes various features such as abbreviation preview, automatic URL recognition and edit points, for HTML and CSS files. The user interface of the settings section is shown in the screenshot given below:



Editor › Emmet

Expand abbreviation with: Tab

- [x] Enable XML/HTML Emmet
- [ ] Enable abbreviation preview
- [x] Enable automatic URL recognition while wrapping text with <a> tag
- [ ] Add edit point at the end of template

BEM

Class name's element separator: __

Class name's modifier separator: _

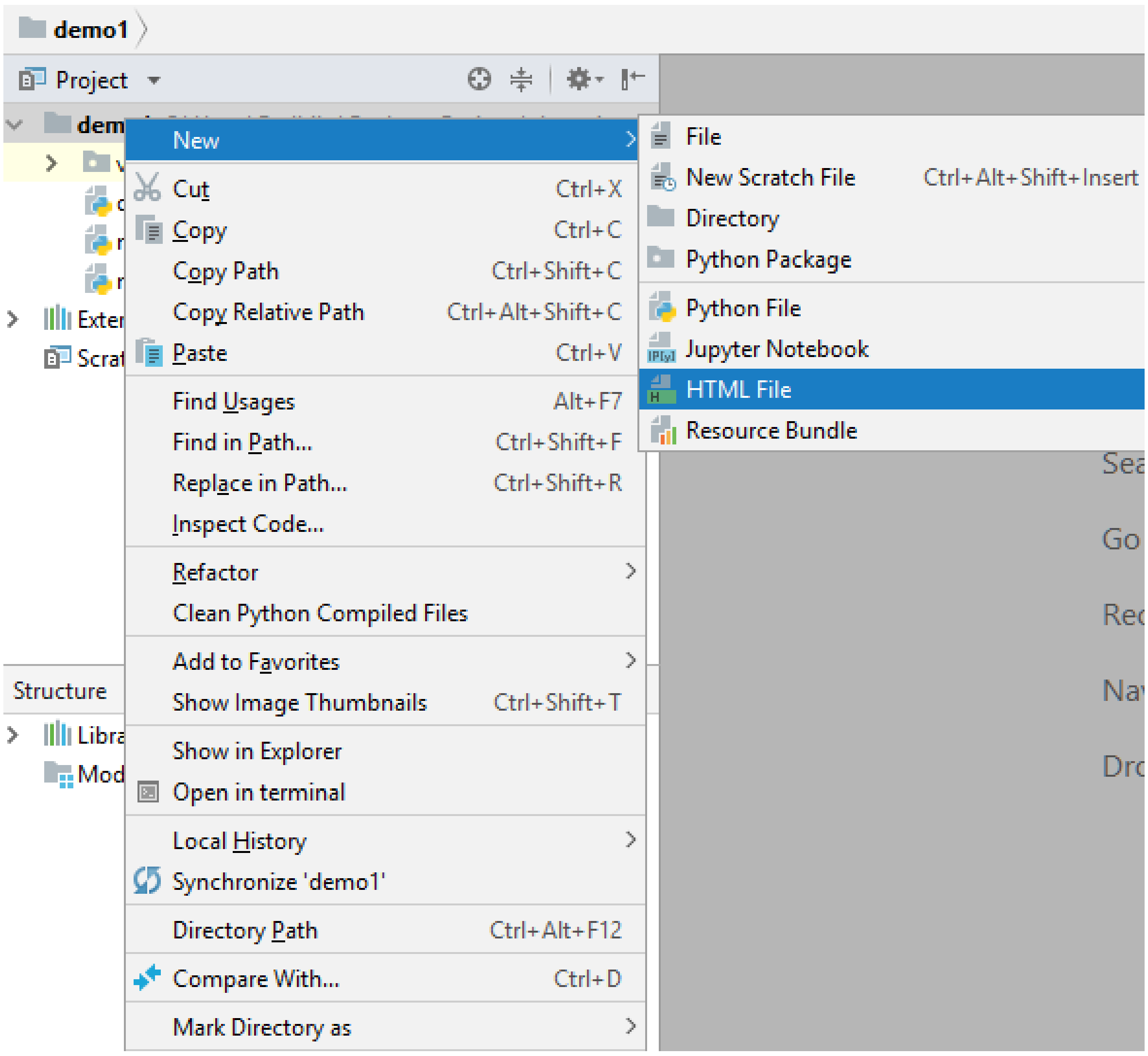
Short element prefix: -

Filters enabled by default

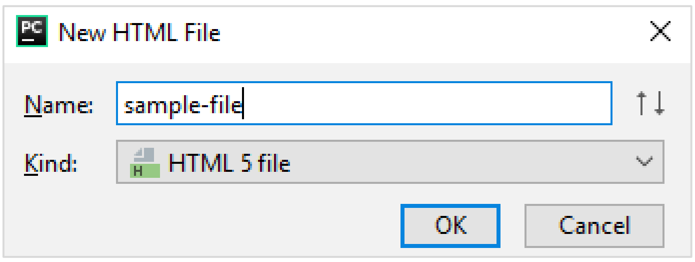
- [ ] XSL tuning
- [ ] Comment tags
- [ ] Escape
- [ ] Single line
- [ ] BEM
- [ ] Trim line markers

## Creating HTML and CSS files

PyCharm includes an inbuilt feature for creating HTML and CSS files. The basic steps for creating new HTML and CSS files are as follows:



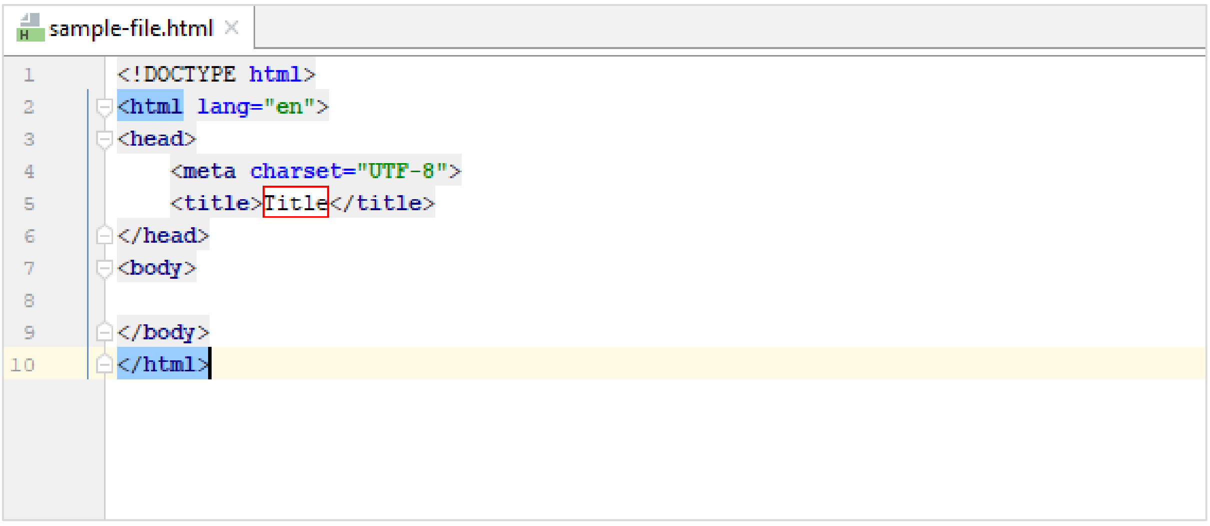
Now, mention the name of file while creating HTML files in the project as shown below:



Name: sample-file

Kind: HTML 5 file

This creates the **sample-file.html** file as shown below:

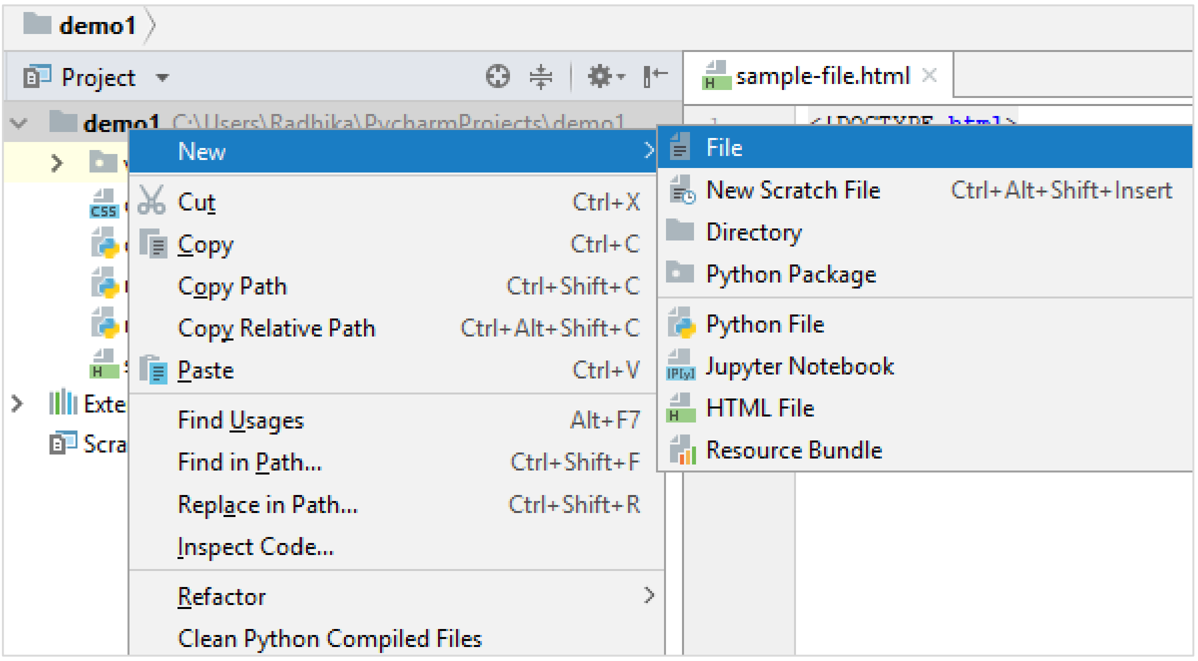


```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>

</body>
</html>
```

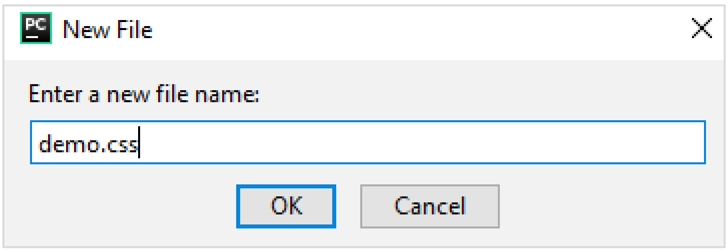
## Creating CSS file

The steps for creating a CSS file are shown here:

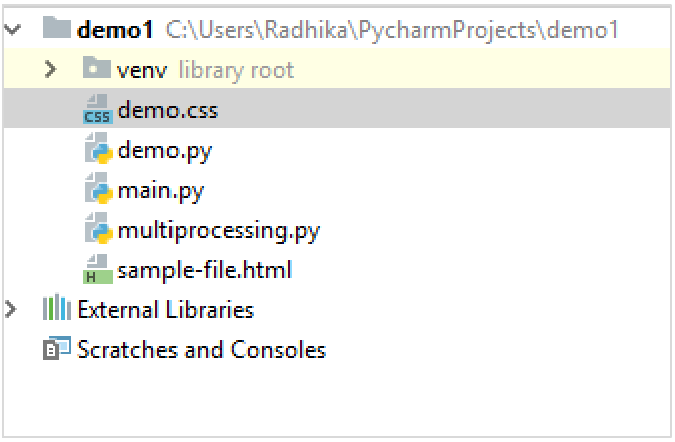
From the **New** menu, select the **File** option as shown below:



Specify the name of CSS during its creation as shown here:



You can see the complete project structure with various files mentioned in different color combinations as shown below:

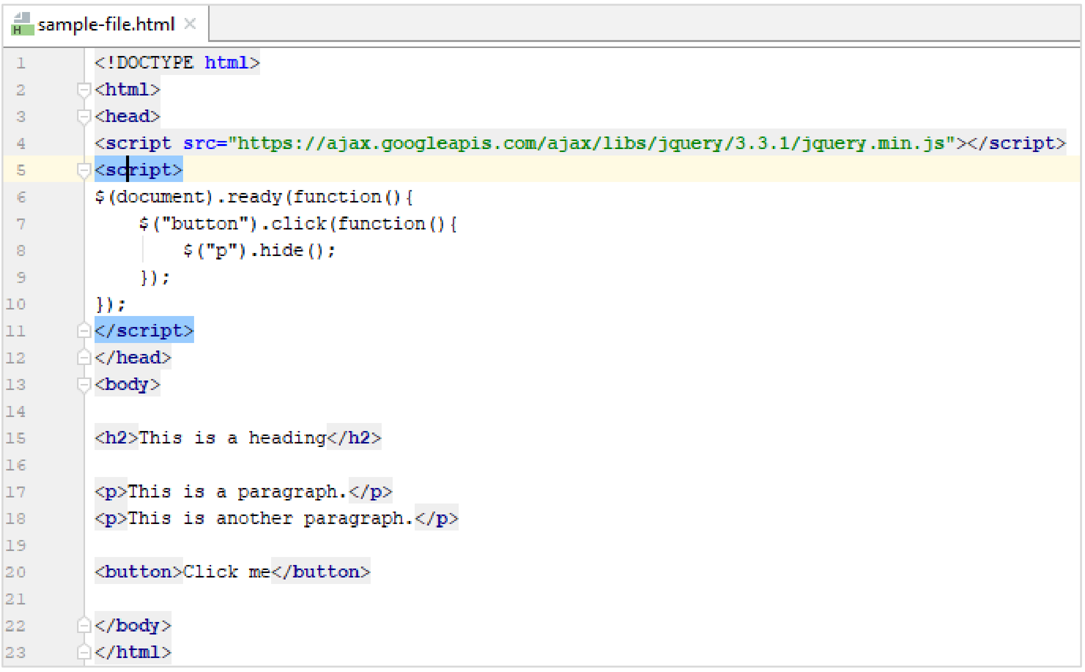


demo1 C:\Users\Radhika\PycharmProjects\demo1
venv library root
demo.css
demo.py
main.py
multiprocessing.py
sample-file.html
External Libraries
Scratches and Consoles

# 15. PyCharm – JavaScript Support

In this chapter, we will focus on main features in using JavaScript in PyCharm editor. When a user implements JavaScript library through URL, PyCharm intends to download a local copy so it can be used for completion and code analysis.

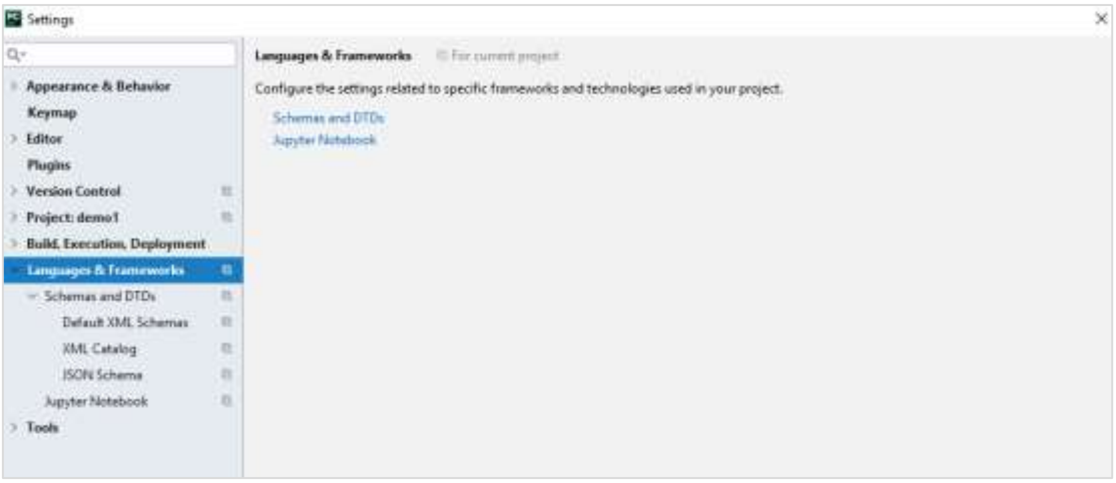
Consider the sample code of our HTML file as shown below, which we created in the previous chapter:



sample-file.html

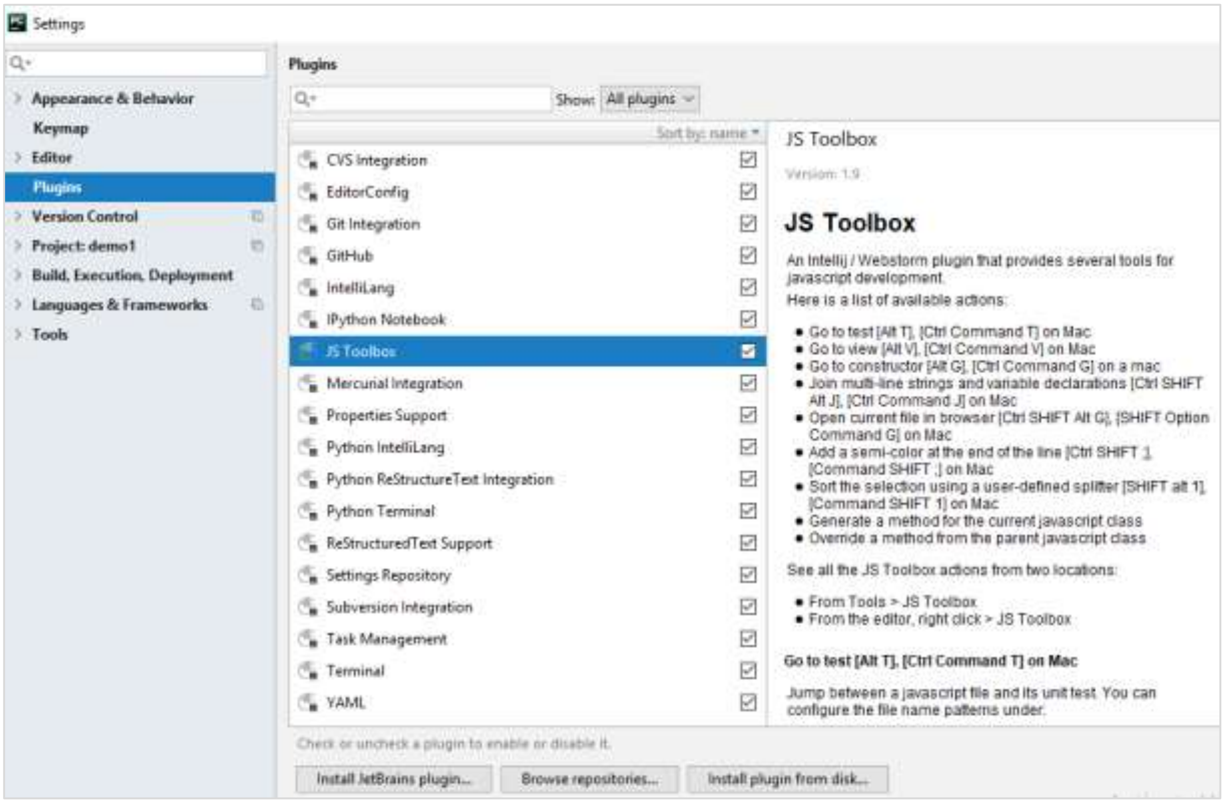
```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        $("p").hide();
    });
});
</script>
</head>
<body>

<h2>This is a heading</h2>

<p>This is a paragraph.</p>
<p>This is another paragraph.</p>

<button>Click me</button>

</body>
</html>
```

For each HTML file or JavaScript file, you can check the external libraries loaded through **Settings** configuration of PyCharm Editor. Observe the screenshot shown below for a better understanding:



Configure the settings related to specific frameworks and technologies used in your project.

Note that you cannot see any library unless you download and implement it. PyCharm also includes JavaScript support of various libraries through a toolbox called **JS Toolbox**. The following screenshot shows this.



JS Toolbox

Version: 1.9

**JS Toolbox**

An Intellij / Webstorm plugin that provides several tools for javascript development.

Here is a list of available actions:

- Go to test [Alt T], [Ctrl Command T] on Mac
- Go to view [Alt V], [Ctrl Command V] on Mac
- Go to constructor [Alt G], [Ctrl Command G] on a mac
- Join multi-line strings and variable declarations [Ctrl SHIFT Alt J], [Ctrl Command J] on Mac
- Open current file in browser [Ctrl SHIFT Alt G], [SHIFT Option Command G] on Mac
- Add a semi-color at the end of the line [Ctrl SHIFT ;], [Command SHIFT ;] on Mac
- Sort the selection using a user-defined splitter [SHIFT alt 1], [Command SHIFT 1] on Mac
- Generate a method for the current javascript class
- Override a method from the parent javascript class

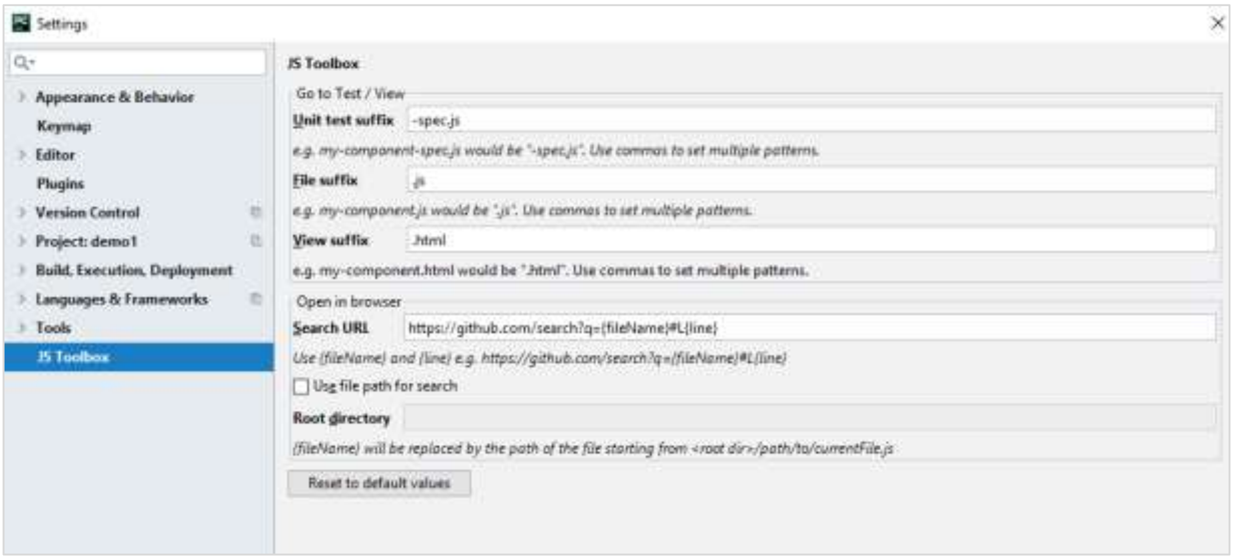
See all the JS Toolbox actions from two locations:

- From Tools > JS Toolbox
- From the editor, right click > JS Toolbox

**Go to test [Alt T], [Ctrl Command T] on Mac**

Jump between a javascript file and its unit test. You can configure the file name patterns under.

It also includes various attributes which are necessary for the JavaScript file configuration. The list of attributes and configurations is shown below:



JS Toolbox

Go to Test / View

Unit test suffix: -spec.js

*e.g. my-component-spec.js would be "-spec.js". Use commas to set multiple patterns.*

File suffix: .js

*e.g. my-component.js would be ".js". Use commas to set multiple patterns.*

View suffix: .html

e.g. my-component.html would be ".html". Use commas to set multiple patterns.

Open in browser

Search URL: https://github.com/search?q={fileName}#L{line}

*Use {fileName} and {line} e.g. https://github.com/search?q={fileName}#L{line}*

☐ Use file path for search

Root directory:

*{fileName} will be replaced by the path of the file starting from <root dir>/path/to/currentFile.js*

Reset to default values

Observe that it includes various parameters such as **Unit test suffix**, **File suffix**, **View suffix**, **Search URL** and the specific **Root directory**.

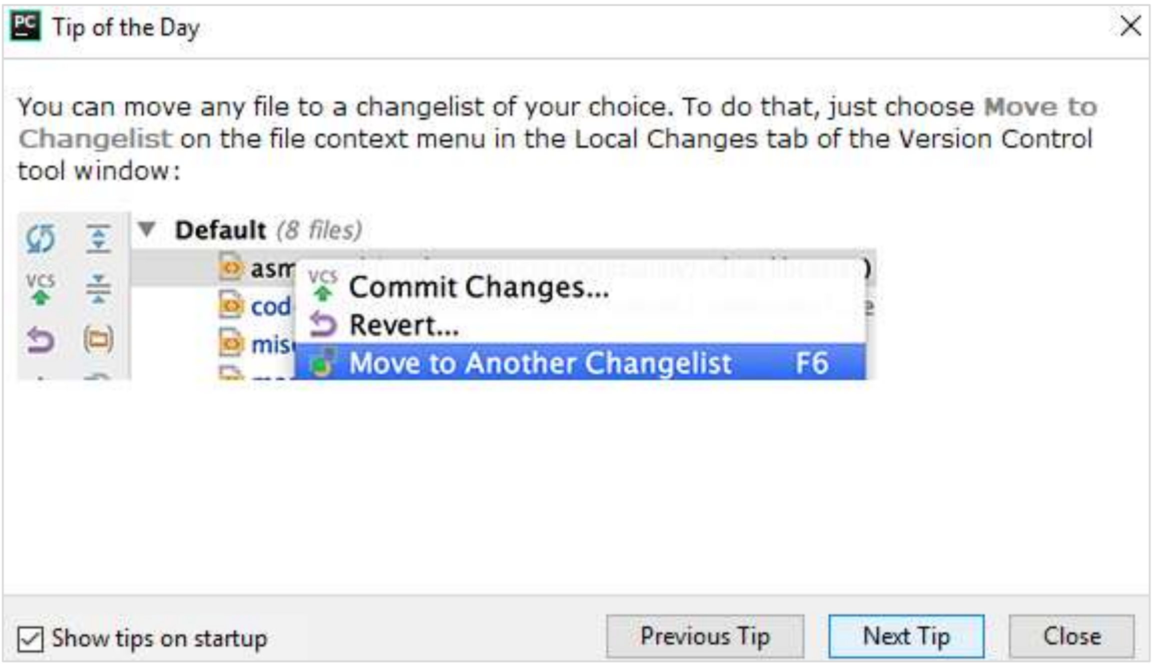
# 16. PyCharm – Tips

PyCharm includes various tips during startup that help its user to understand its functionalities and operations. It also includes some shortcuts which are mandatory to understand.

In this chapter, you will see some of the important PyCharm tips.

## Changing the File to a Specific Changelist

This tip shows how to change the file to a specific changelist as per the user's choice. This helps in managing repositories as per version control system settings. Observe the following screenshot for a better understanding:

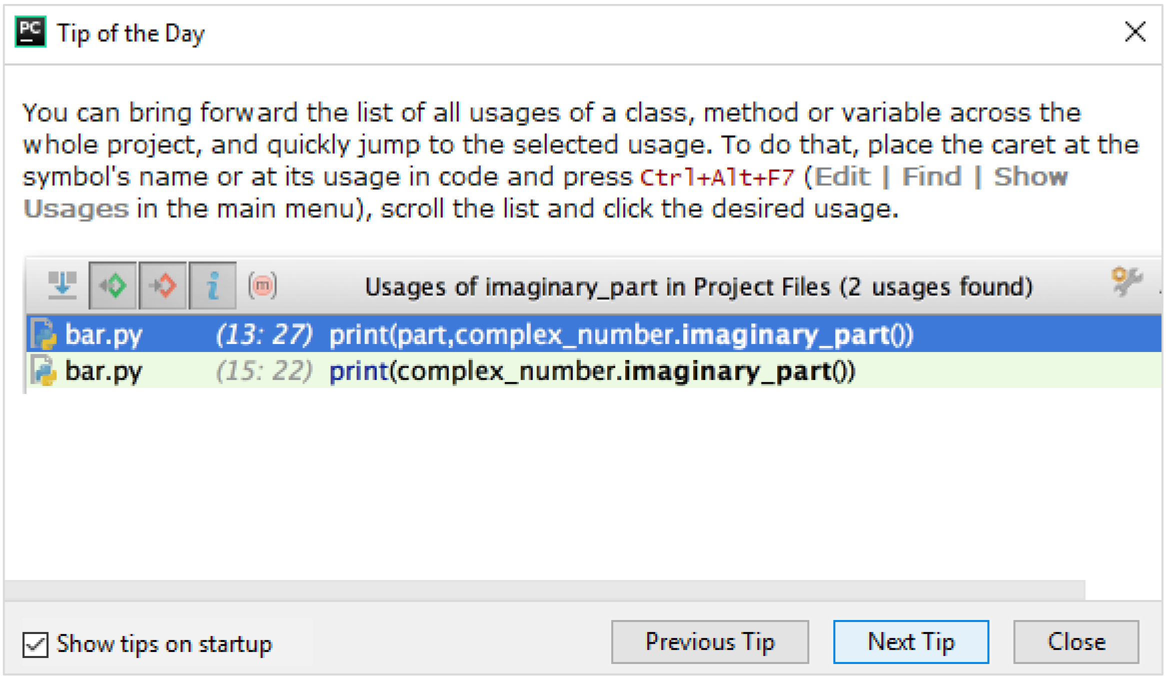


Tip of the Day

You can move any file to a changelist of your choice. To do that, just choose **Move to Changelist** on the file context menu in the Local Changes tab of the Version Control tool window:

## Display the List of all Usages in a Class

This function displays the list of all usages included in a specific class, method or variable across the project. It quickly enables the user to jump to specific area. Observe the following screenshot for a better understanding:



Tip of the Day

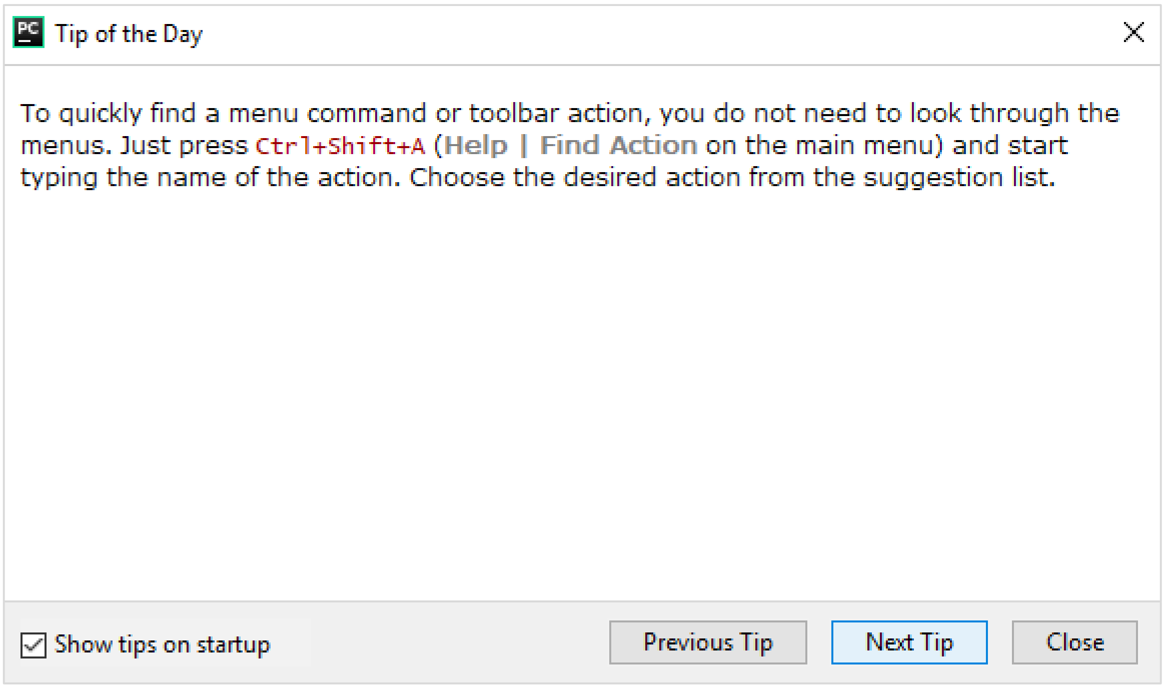
You can bring forward the list of all usages of a class, method or variable across the whole project, and quickly jump to the selected usage. To do that, place the caret at the symbol's name or at its usage in code and press Ctrl+Alt+F7 (Edit | Find | Show Usages in the main menu), scroll the list and click the desired usage.

Usages of imaginary_part in Project Files (2 usages found)

```
bar.py    (13: 27)  print(part,complex_number.imaginary_part())
bar.py    (15: 22)  print(complex_number.imaginary_part())
```

## To find Menu Command for an Action

This tip helps to find menu command for a specific action and the shortcut key for the same is **Ctrl+Shift+A**. A user can select desired action from the mentioned suggestion list.

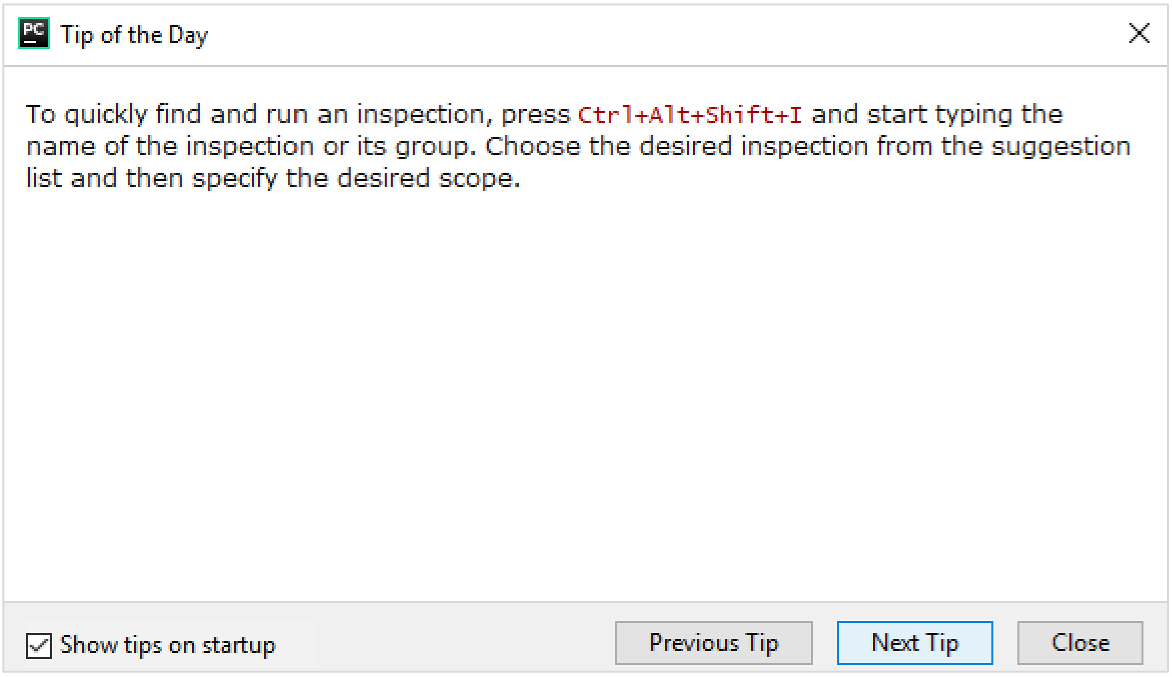


Tip of the Day

To quickly find a menu command or toolbar action, you do not need to look through the menus. Just press Ctrl+Shift+A (Help | Find Action on the main menu) and start typing the name of the action. Choose the desired action from the suggestion list.

## Running Inspection through a Code

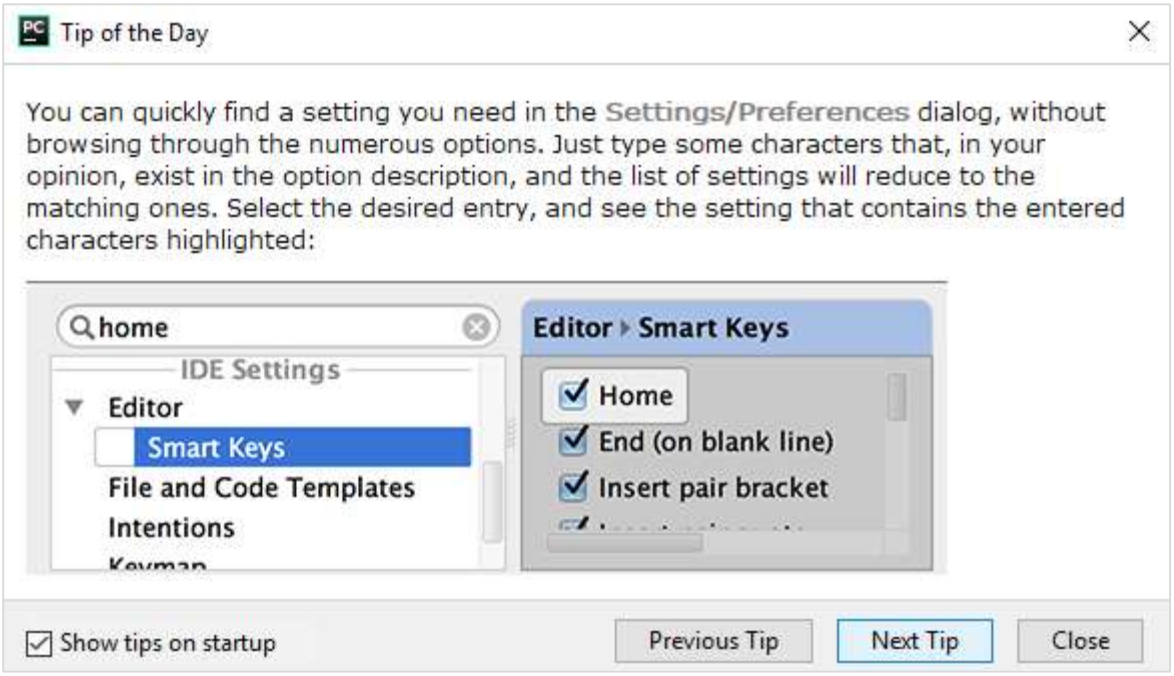
This tip helps in running a specific inspection through the code. The shortcut key combination for the same is **Ctrl+Alt+Shift+I**.


Tip of the Day

To quickly find and run an inspection, press `Ctrl+Alt+Shift+I` and start typing the name of the inspection or its group. Choose the desired inspection from the suggestion list and then specify the desired scope.

## Specify the List of Settings

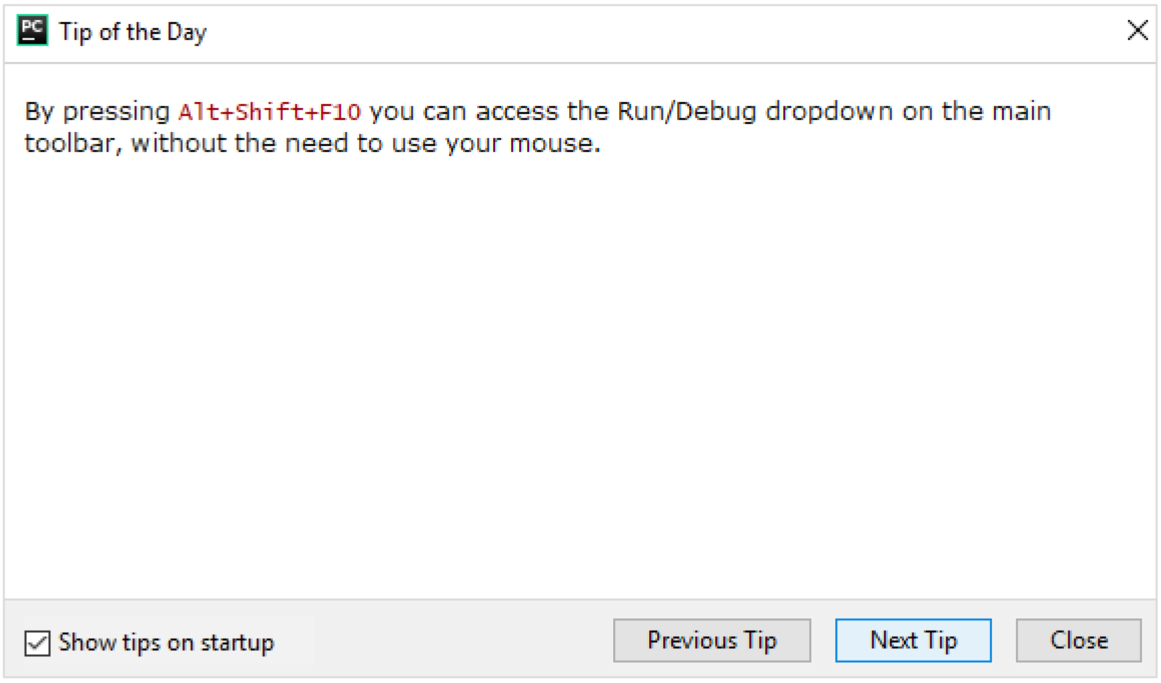
This tip is used to specify the list of desired settings. It includes smart keys for specific editor. The smart keys are shortcut keys for some operations.


Tip of the Day

You can quickly find a setting you need in the Settings/Preferences dialog, without browsing through the numerous options. Just type some characters that, in your opinion, exist in the option description, and the list of settings will reduce to the matching ones. Select the desired entry, and see the setting that contains the entered characters highlighted:

## Run / Debug the Script Files

This tip is very useful for running or debugging the script files which you can access through main toolbar. The shortcut key combination for same is **Alt+Shift+F10**.



By pressing Alt+Shift+F10 you can access the Run/Debug dropdown on the main toolbar, without the need to use your mouse.

# 17. PyCharm – Database Tools

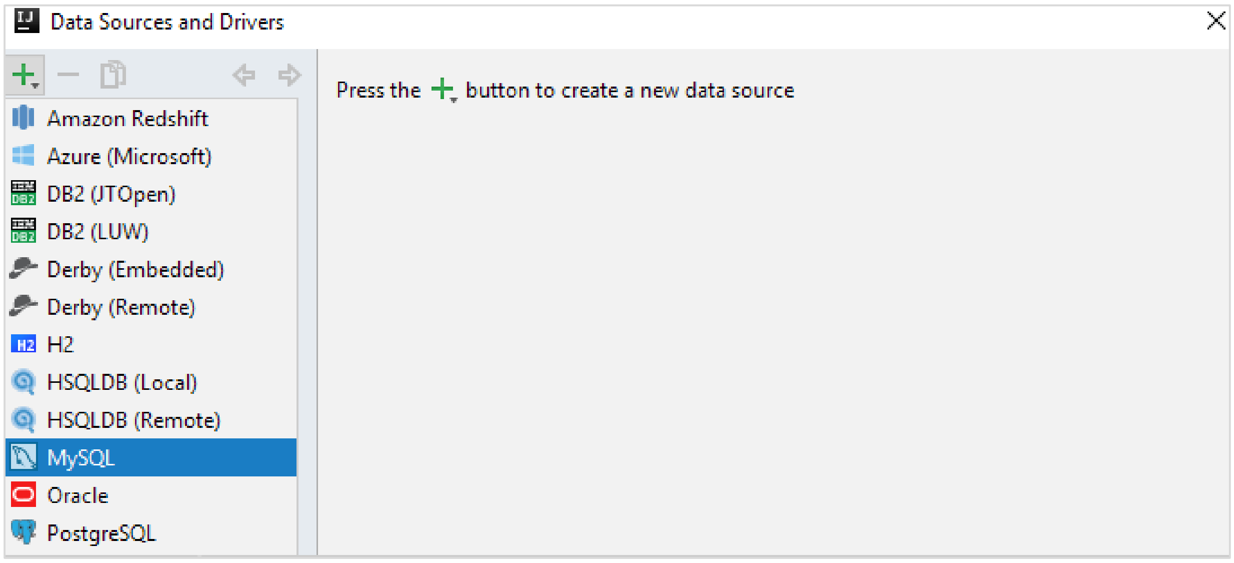
PyCharm supports interface support with various types of databases. Once a user grants access to the created database, it provides schema diagram of the database with SQL writing tools which provide code completion. In this chapter, we will focus on MySQL database connectivity which will involve following steps.

## Adding a Data Source

It is important to keep a note that PyCharm supports a wide variety of database connectivity.

### Step 1

Open the database tool window **View -> Tool Windows -> Database** and open the dialog called **Data Sources and Dialog**.

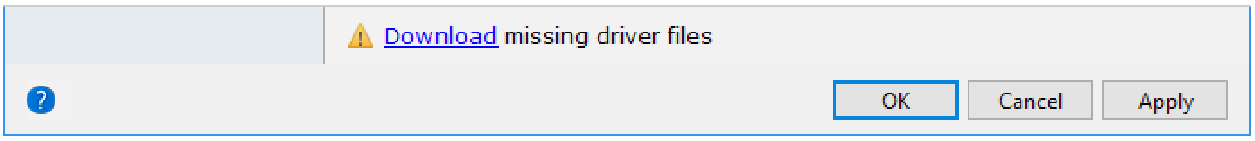


Press the + button to create a new data source

Now, select **MySQL** database for adding a new data source.

### Step 2

User should download the missing driver files to get proper connectivity with **MySQL database**.



Download missing driver files

## Step 3

Now, specify the configuration settings for connectivity to be achieved.

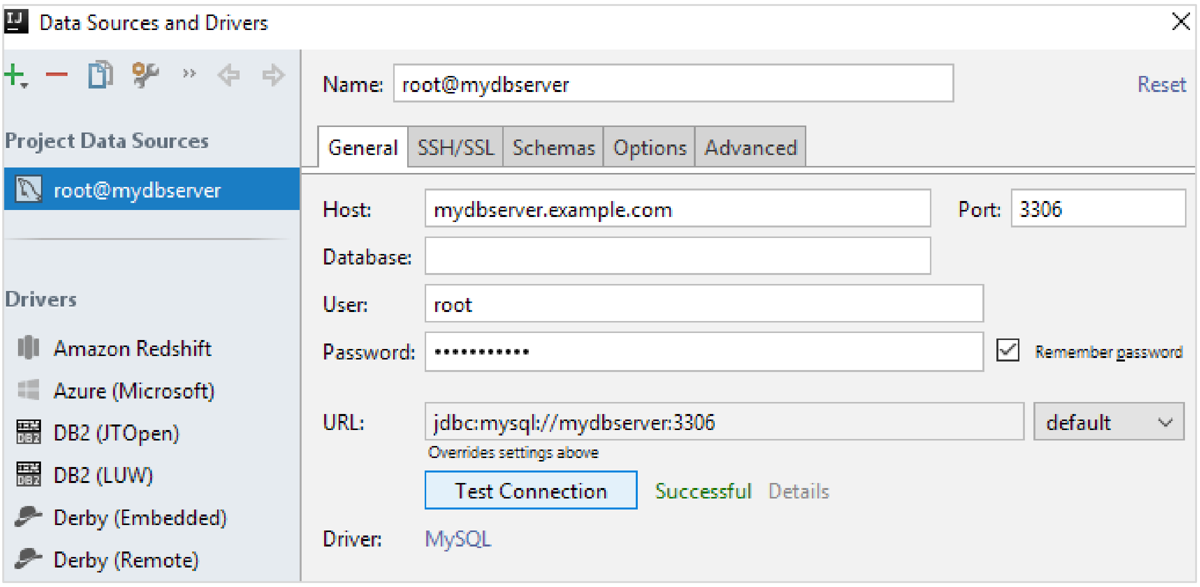
**Host:** If you database server is on a different computer, replace localhost with the IP address of the server host, e.g. 172.20.240.163.

**Port:** The default MySQL server port is 3306. If your server uses a different port, specify that port.

**User and Password:** These are the required credentials.

## Step 4

Always make sure that database connectivity is successful through **Test Connection** feature.



Name: root@mydbserver

Host: mydbserver.example.com Port: 3306

Database:

User: root

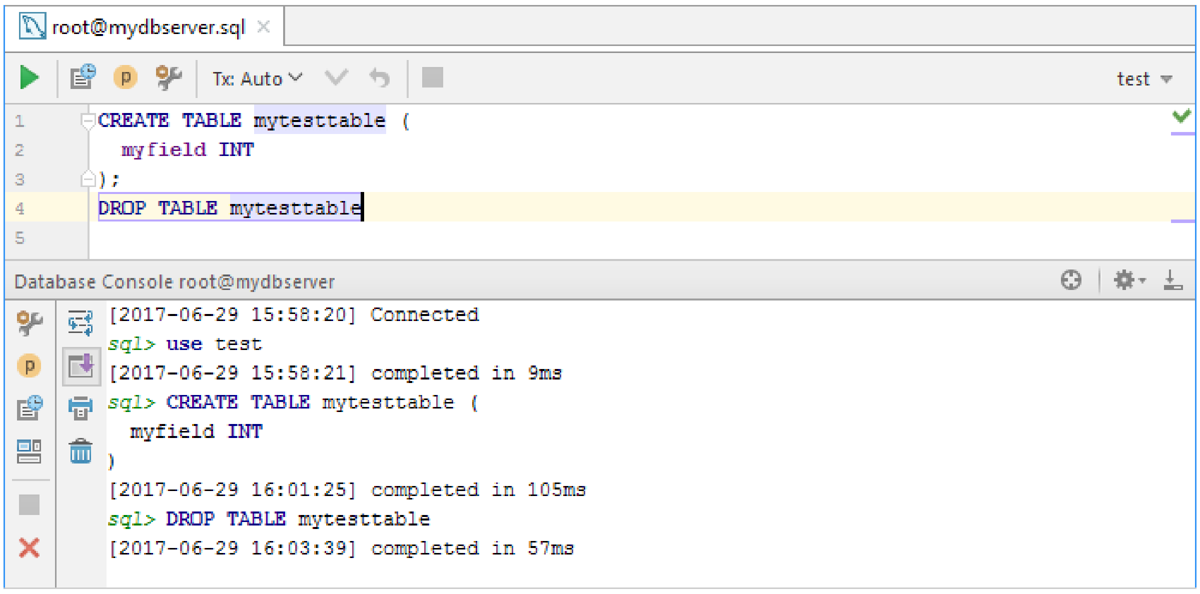
Password: ••••••••••• ☑ Remember password

URL: jdbc:mysql://mydbserver:3306 default

Test Connection Successful

Driver: MySQL

Testing the connection also involves creating test table through query and executing them. Once the execution is successful, you can drop the database.



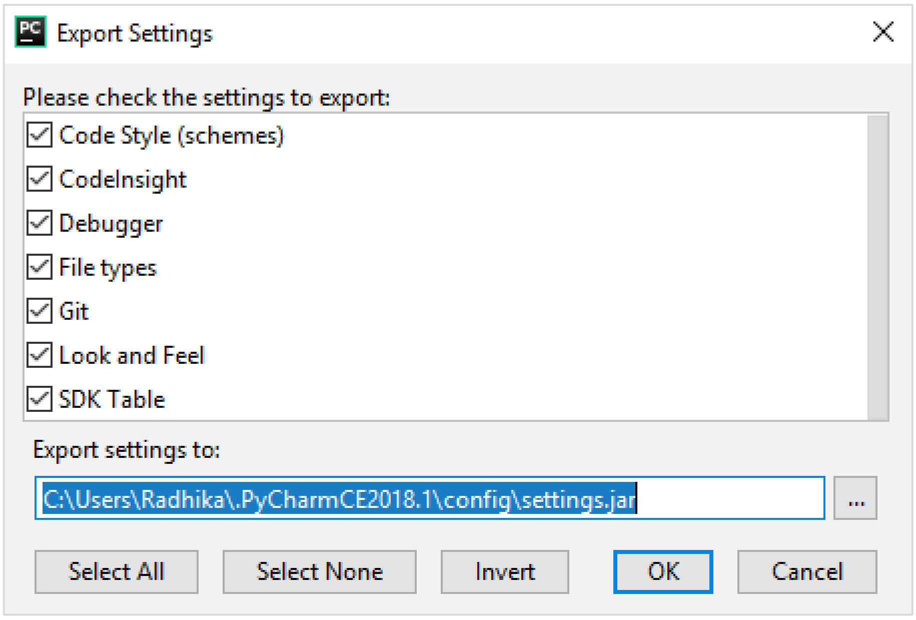
```
CREATE TABLE mytesttable (
  myfield INT
);
DROP TABLE mytesttable
```

```
[2017-06-29 15:58:20] Connected
sql> use test
[2017-06-29 15:58:21] completed in 9ms
sql> CREATE TABLE mytesttable (
  myfield INT
)
[2017-06-29 16:01:25] completed in 105ms
sql> DROP TABLE mytesttable
[2017-06-29 16:03:39] completed in 57ms
```

# 18. PyCharm – Exporting Data

PyCharm IDE includes various features for converting the existing code file into HTML format or CSV format. In this chapter, you will learn exporting data using PyCharm IDE.

The export settings of PyCharm editor are shown in the figure given below:



Export Settings

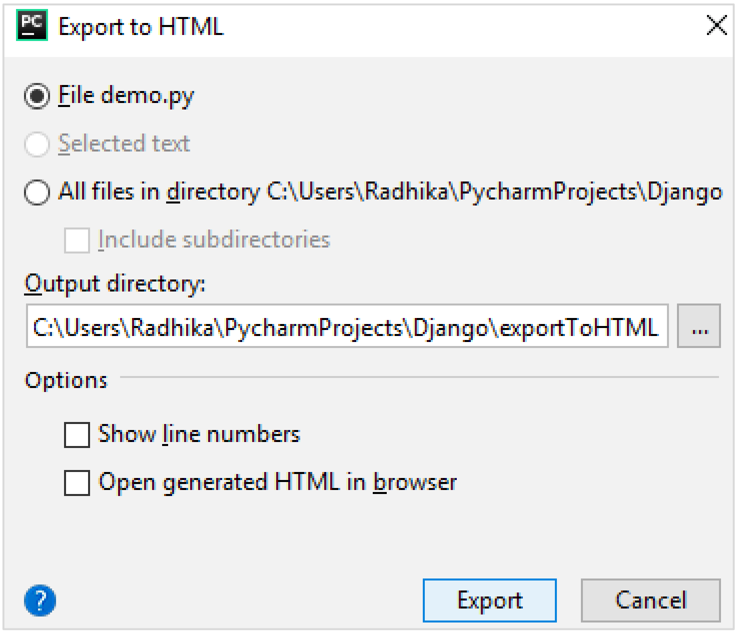
Please check the settings to export:

- [✓] Code Style (schemes)
- [✓] CodeInsight
- [✓] Debugger
- [✓] File types
- [✓] Git
- [✓] Look and Feel
- [✓] SDK Table

Export settings to: C:\Users\Radhika\.PyCharmCE2018.1\config\settings.jar

## Export to HTML feature

This feature helps in exporting the specific file in HTML format. This is done to improve the security purposes of the given module. The following screenshot gives a better understanding:



Export to HTML

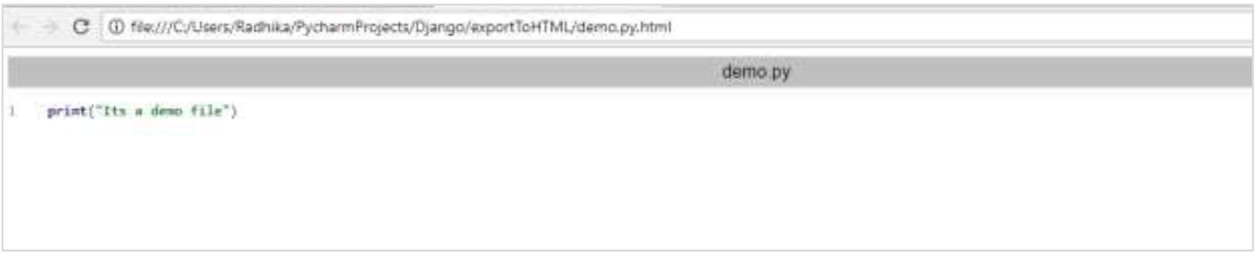
- ● File demo.py
- ○ Selected text
- ○ All files in directory C:\Users\Radhika\PycharmProjects\Django
  - ☐ Include subdirectories

Output directory: C:\Users\Radhika\PycharmProjects\Django\exportToHTML

Options

- ☐ Show line numbers
- ☐ Open generated HTML in browser

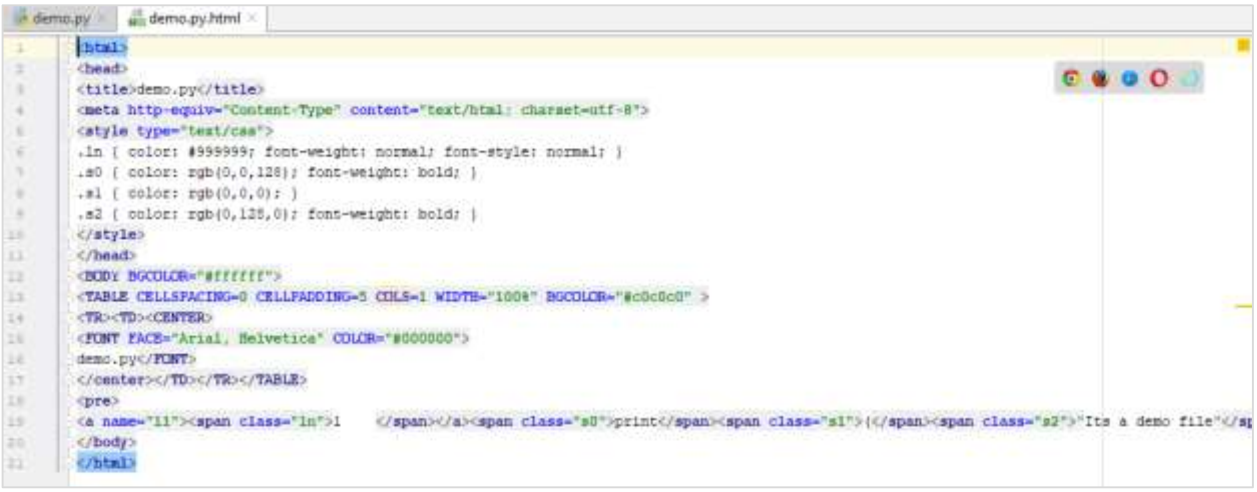
Once the export operation is successful, the generated HTML file will display in browser output as shown below:


file:///C:/Users/Radhika/PycharmProjects/Django/exportToHTML/demo.py.html

demo.py

```
1   print("Its a demo file")
```

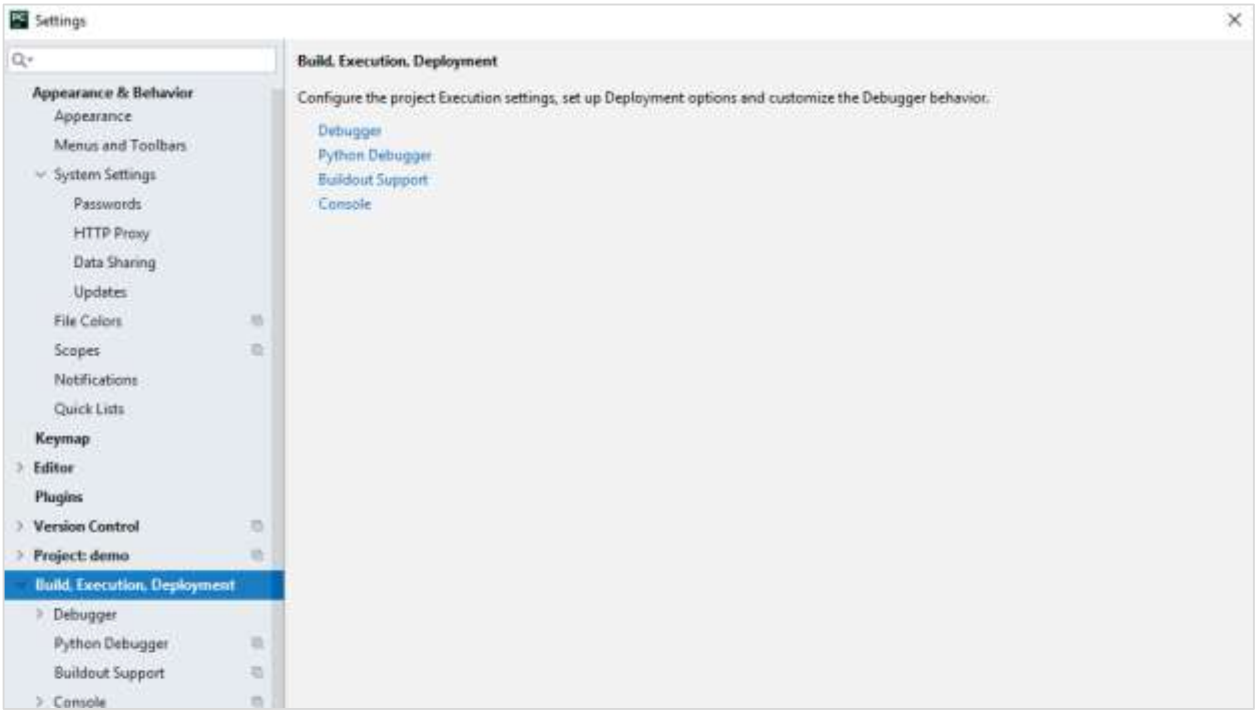
Now, if you check the HTML code generated after the export operation, you can observe that line numbers are also included to achieve this operation.



```
<html>
<head>
<title>demo.py</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<style type="text/css">
.ln { color: #999999; font-weight: normal; font-style: normal; }
.s0 { color: rgb(0,0,128); font-weight: bold; }
.s1 { color: rgb(0,0,0); }
.s2 { color: rgb(0,128,0); font-weight: bold; }
</style>
</head>
<BODY BGCOLOR="#ffffff">
<TABLE CELLSPACING=0 CELLPADDING=5 COLS=1 WIDTH="100%" BGCOLOR="#c0c0c0" >
<TR><TD><CENTER>
<FONT FACE="Arial, Helvetica" COLOR="#000000">
demo.py</FONT>
</center></TD></TR></TABLE>
<pre>
<a name="l1"><span class="ln">1    </span></a><span class="s0">print</span><span class="s1">(</span><span class="s2">"Its a demo file"</sp
</body>
</html>
```

# 19. PyCharm – Web Frameworks

This chapter focusses on web frameworks and its deployment. PyCharm has a simple functionality to deploy code and files. To deploy code with PyCharm, we need to add a web server with Menu Option **Settings -> Build, Execution-> Deployment**.

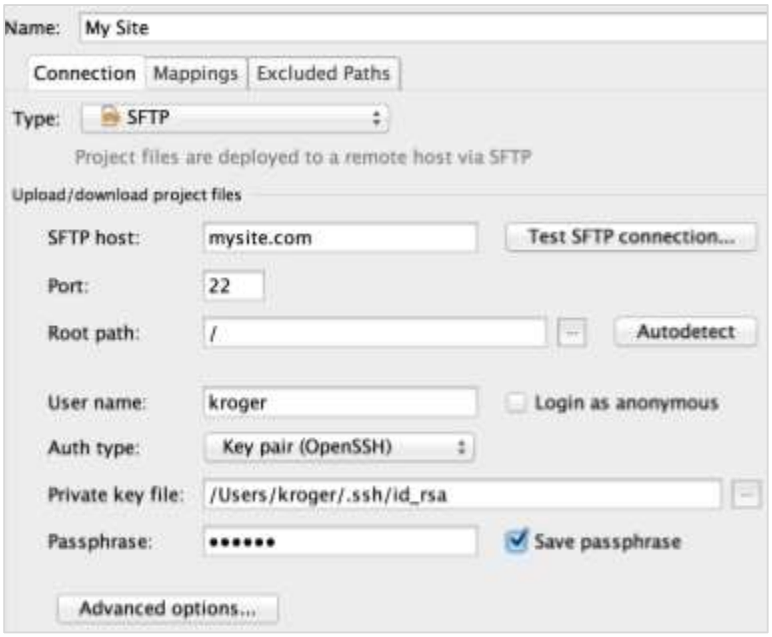


Build, Execution, Deployment

Configure the project Execution settings, set up Deployment options and customize the Debugger behavior.

Debugger
Python Debugger
Buildout Support
Console

Now, include all the settings with various configurations required for deployment of the project.



Name: My Site

Type: SFTP

Project files are deployed to a remote host via SFTP

Upload/download project files

SFTP host: mysite.com

Port: 22

Root path: /

User name: kroger

Login as anonymous

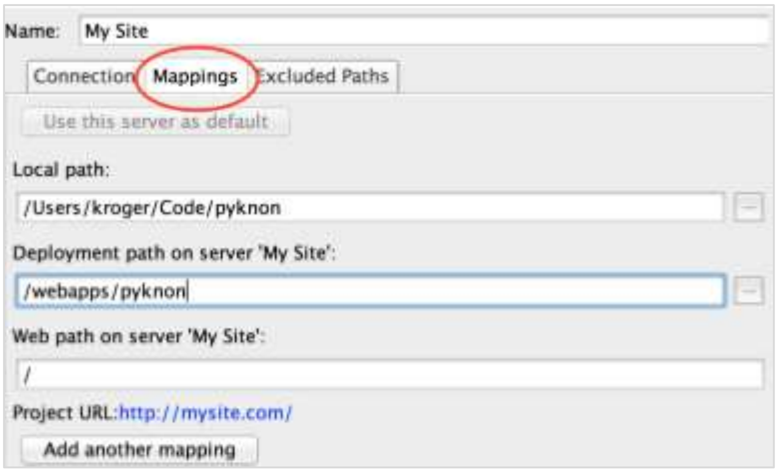
Auth type: Key pair (OpenSSH)

Private key file: /Users/kroger/.ssh/id_rsa

Passphrase: ••••••

[✓] Save passphrase

In the **Mappings** tab, a user can specify where the local code is and where it should be copied to remotely.



Name: My Site

Connection | Mappings | Excluded Paths

Use this server as default

Local path: /Users/kroger/Code/pyknon

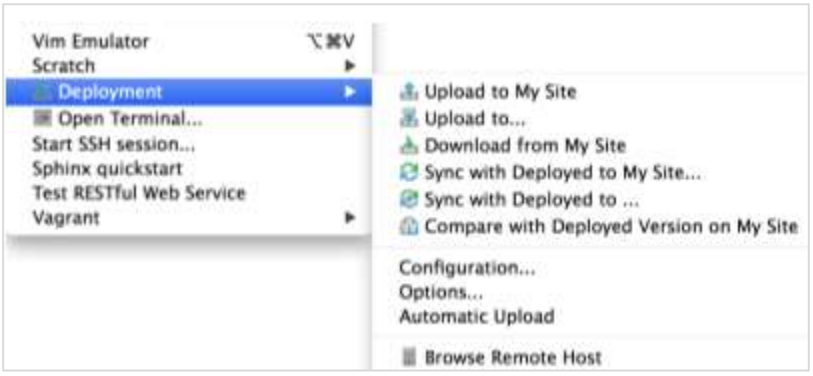
Deployment path on server 'My Site': /webapps/pyknon

Web path on server 'My Site': /

Project URL: http://mysite.com/

Add another mapping

The code can be deployed using **Tools -> Deployment** option under the Tools menu bar. Deployment in PyCharm is very granular: a user can deploy one single file or the whole source code.

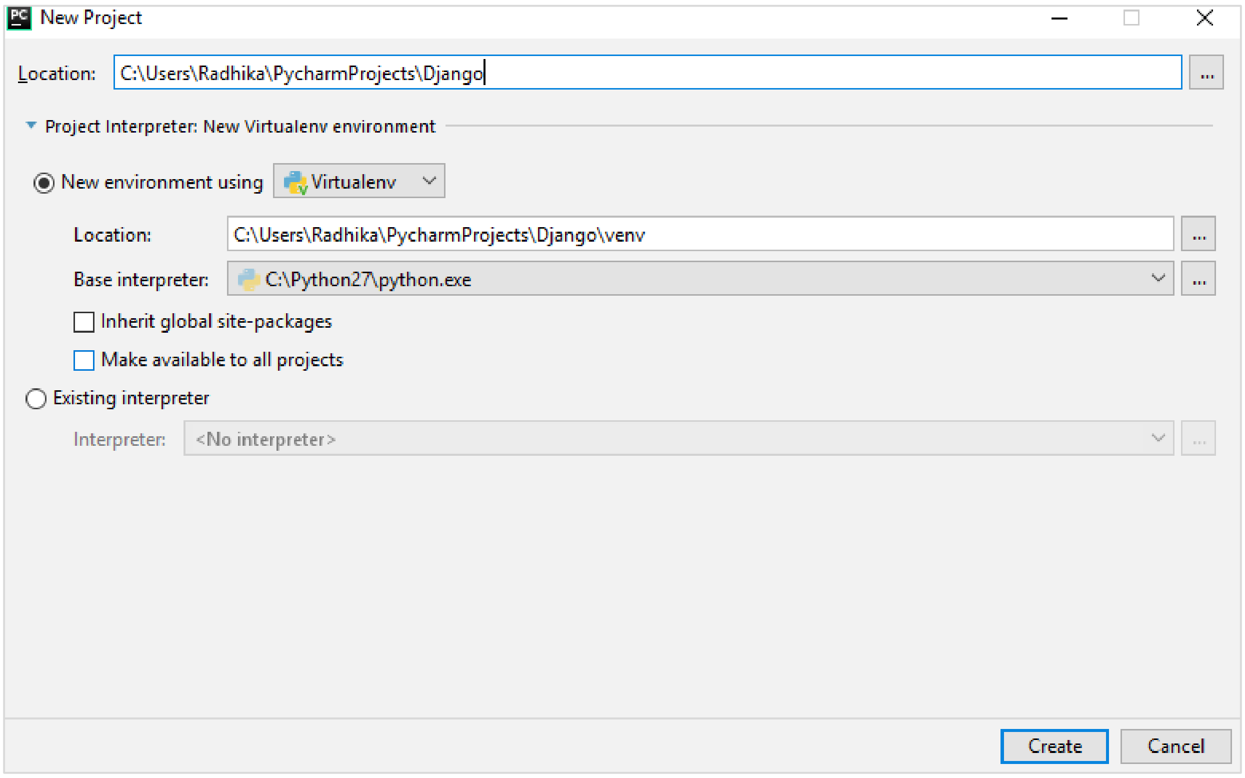


PyCharm also includes various actions to compare remote and local versions. The editor is more reliable to use automatic deployments and a version control system to compare local and remote versions.

# 20. PyCharm – Django

One of the features of PyCharm is that it includes a support for Django. With the ability of including JavaScript features within PyCharm, it can be considered as the best IDE for Django.

The basic steps for creating a Django project in PyCharm IDE are given below:



Location: C:\Users\Radhika\PycharmProjects\Django

Project Interpreter: New Virtualenv environment

[✓ New environment using] Virtualenv

Location: C:\Users\Radhika\PycharmProjects\Django\venv

Base interpreter: C:\Python27\python.exe

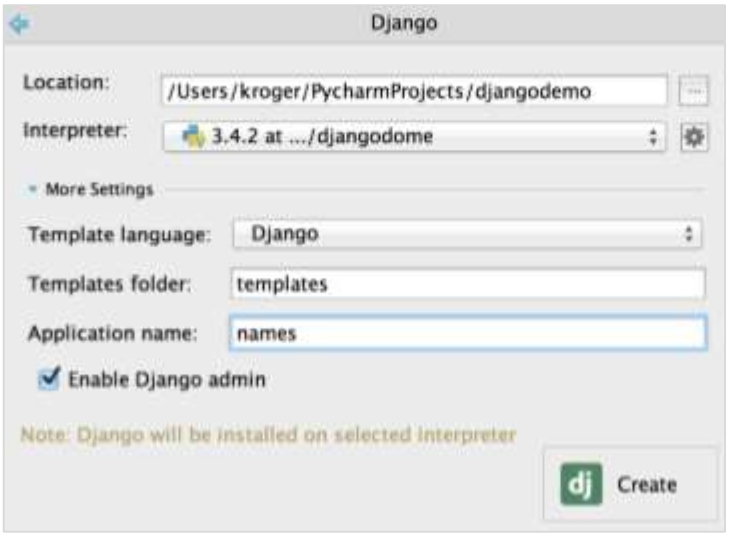
Inherit global site-packages

Make available to all projects

Existing interpreter

Interpreter: <No interpreter>

If the **EnableDjangoadmin** option is enabled, PyCharm will setup the admin site for you.



Location: /Users/kroger/PycharmProjects/djangodemo

Interpreter: 3.4.2 at .../djangodome

More Settings

Template language: Django

Templates folder: templates

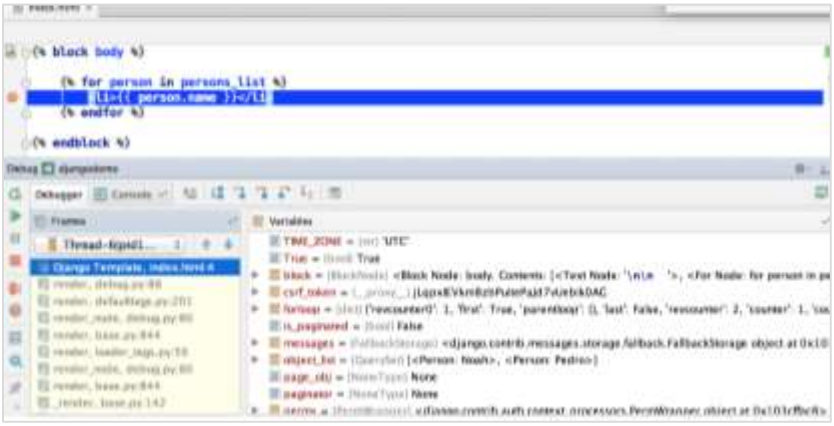
Application name: names

[✓] Enable Django admin

Note: Django will be installed on selected interpreter

## Template Debugging

Debugging works with Django and Jinja templates. We can inspect variables, step through code, and do what we expect in a debugger.



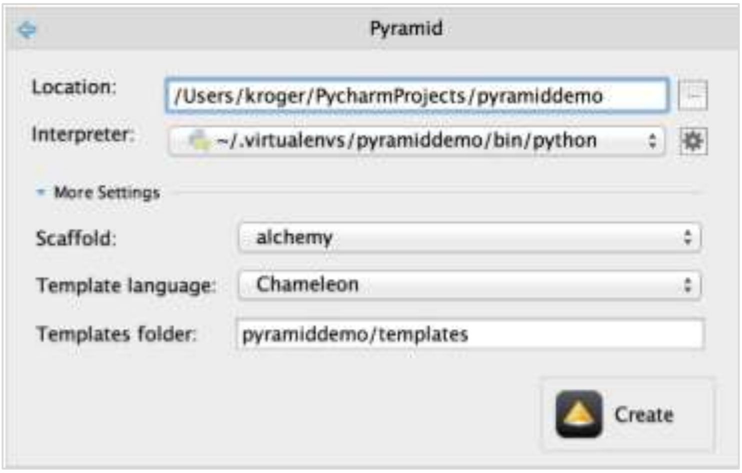
```
{% block body %}

    {% for person in persons_list %}
        <li>{{ person.name }}</li>
    {% endfor %}

{% endblock %}
```

```
TIME_ZONE = {str} 'UTC'
True = {bool} True
block = {BlockNode} <Block Node: body. Contents: [<Text Node: '\n\n   '>, <For Node: for person in pe
csrf_token = {__proxy__} jLqpxEVkmBzbPuleRaJd7vUebkDAC
forloop = {dict} {'revcounter0': 1, 'first': True, 'parentloop': {}, 'last': False, 'revcounter': 2, 'counter': 1, 'co
is_paginated = {bool} False
messages = {FallbackStorage} <django.contrib.messages.storage.fallback.FallbackStorage object at 0x10
object_list = {QuerySet} [<Person: Noah>, <Person: Pedro>]
page_obj = {NoneType} None
paginator = {NoneType} None
perms = {PermWrapper} <django.contrib.auth.context_processors.PermWrapper object at 0x101cfbcf8>
```

# 21. PyCharm – Pyramid

You can create a project of **Pyramid Framework** in PyCharm editor by using its Welcome Window.

A user can set the project's interpreter and Python location, choosing scaffold, and a template language by default. The scaffold in Pyramid framework uses URL dispatch to map URLs and to view code and SQLAlchemy for persistence properties.


Pyramid

Location: /Users/kroger/PycharmProjects/pyramiddemo

Interpreter: ~/.virtualenvs/pyramiddemo/bin/python

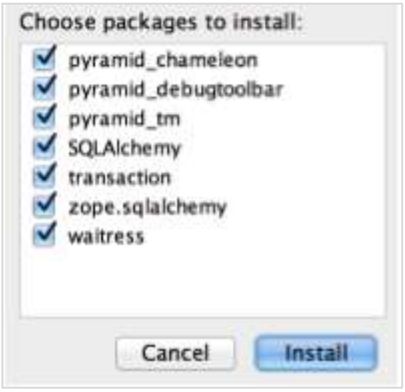
More Settings

Scaffold: alchemy

Template language: Chameleon

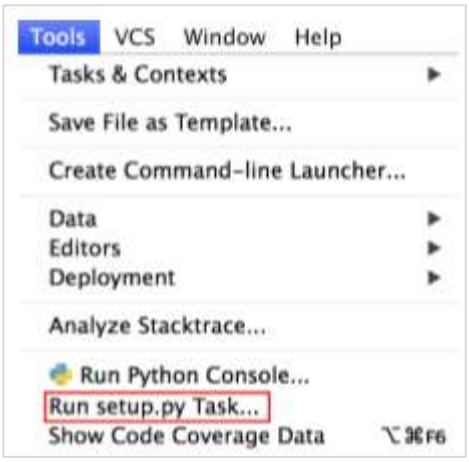
Templates folder: pyramiddemo/templates

PyCharm editor will ask user with list of required packages in the **setup.py** file and prompts the user to download the required packages.

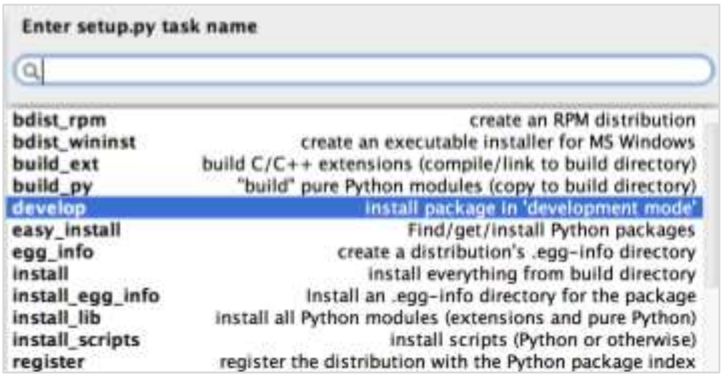

Choose packages to install:

- ✓ pyramid_chameleon
- ✓ pyramid_debugtoolbar
- ✓ pyramid_tm
- ✓ SQLAlchemy
- ✓ transaction
- ✓ zope.sqlalchemy
- ✓ waitress

Install the project in development mode (refer to official documentation of Pyramid for more details). The user should run python setup.py through the menu **Tools -> Run setup.py** option.



A user should select **develop task** while running a .py file as mentioned in the below window:

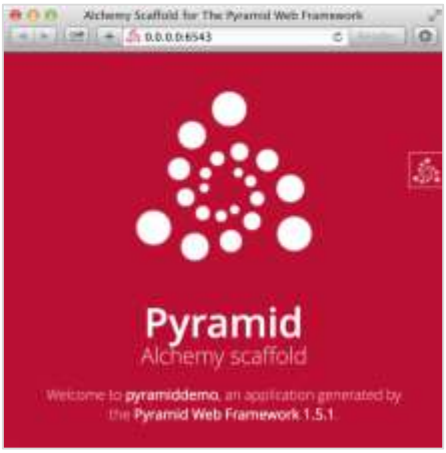

Enter setup.py task name

| | |
|---|---|
| bdist_rpm | create an RPM distribution |
| bdist_wininst | create an executable installer for MS Windows |
| build_ext | build C/C++ extensions (compile/link to build directory) |
| build_py | "build" pure Python modules (copy to build directory) |
| develop | install package in 'development mode' |
| easy_install | Find/get/install Python packages |
| egg_info | create a distribution's .egg-info directory |
| install | install everything from build directory |
| install_egg_info | Install an .egg-info directory for the package |
| install_lib | install all Python modules (extensions and pure Python) |
| install_scripts | install scripts (Python or otherwise) |
| register | register the distribution with the Python package index |

It is important to initialize the database using console script named **initialize <project_name>** using the following command:

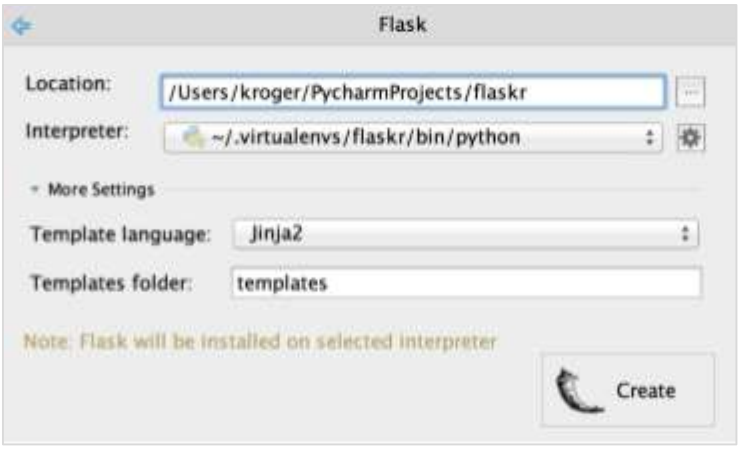
```
initialize_pyramiddemo_db development.ini
```

The user can start the server by running the project which will display the result as shown below:


Alchemy Scaffold for The Pyramid Web Framework
0.0.0.0:6543
Pyramid
Alchemy scaffold
Welcome to pyramiddemo, an application generated by the Pyramid Web Framework 1.5.1

# 22. PyCharm – Flask

PyCharm supports Flask framework development. You can easily create a new Flask project by creating new project through welcome screen. You can set the project's location and the virtual environment and choose a template language and where the templates will be located.



Flask

Location: /Users/kroger/PycharmProjects/flaskr

Interpreter: ~/.virtualenvs/flaskr/bin/python

More Settings

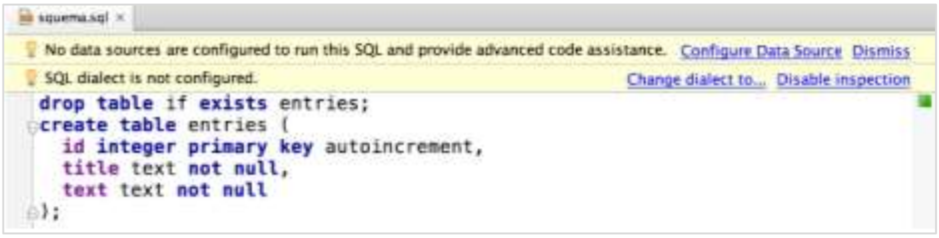
Template language: Jinja2

Templates folder: templates

Note: Flask will be installed on selected interpreter

You can run a project by using **Run -> Run '<project-name>'**.

You can also add a new data source with this framework. Let us create a file called **squema.sql** and add SQL code to create some tables. PyCharm editor will recognize the files and asks you to configure a data source and set up to the database dialect.

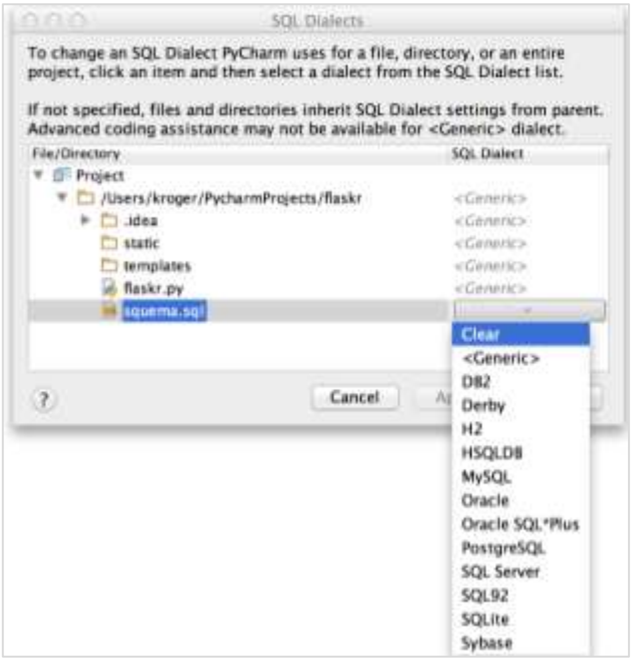


No data sources are configured to run this SQL and provide advanced code assistance. Configure Data Source Dismiss

SQL dialect is not configured. Change dialect to... Disable inspection

```
drop table if exists entries;
create table entries (
  id integer primary key autoincrement,
  title text not null,
  text text not null
);
```

PyCharm will let you choose the desired dialect that you want to use. You can change the properties of the SQL: **Settings -> Language and Frameworks -> SQL Dialects**



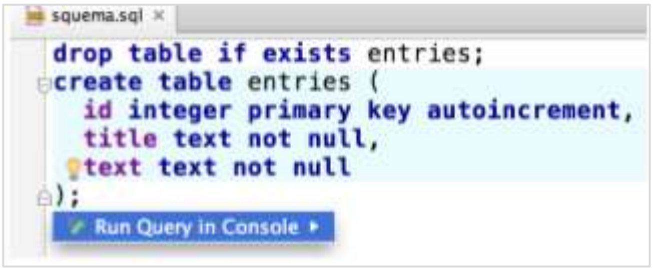
SQL Dialects

To change an SQL Dialect PyCharm uses for a file, directory, or an entire project, click an item and then select a dialect from the SQL Dialect list.

If not specified, files and directories inherit SQL Dialect settings from parent. Advanced coding assistance may not be available for <Generic> dialect.

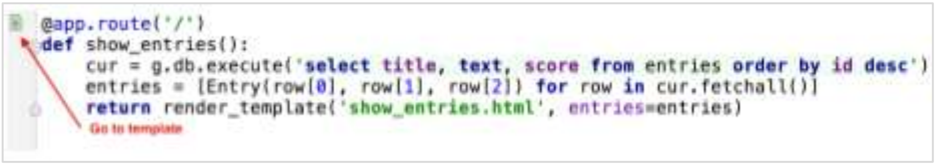
| File/Directory | SQL Dialect |
|---|---|
| Project | |
| /Users/kroger/PycharmProjects/flaskr | <Generic> |
| .idea | <Generic> |
| static | <Generic> |
| templates | <Generic> |
| flaskr.py | <Generic> |
| squema.sql | |

For flask editor, the easiest way to run the SQL query is to click somewhere in the query and click on the inspection window and click “Run Query into console ”.



```
squema.sql
drop table if exists entries;
create table entries (
  id integer primary key autoincrement,
  title text not null,
  text text not null
);
```

The user interface of the **Flask** framework is displayed as below:



```
@app.route('/')
def show_entries():
    cur = g.db.execute('select title, text, score from entries order by id desc')
    entries = [Entry(row[0], row[1], row[2]) for row in cur.fetchall()]
    return render_template('show_entries.html', entries=entries)
```