**Python Tutorial for Beginners: Learn Python Programming in 7 Days**

## What is Python?

Python is an object-oriented programming language created by Guido Rossum in 1989. It is ideally designed for rapid prototyping of complex applications. It has interfaces to many OS system calls and libraries and is extensible to C or C++. Many large companies use the Python programming language include NASA, Google, YouTube, BitTorrent, etc.

Python programming is widely used in Artificial Intelligence, Natural Language Generation, Neural Networks and other advanced fields of Computer Science. Python had deep focus on code readability & this class will teach you python from basics.

## Python Basics

|  |  |
| --- | --- |
| [**Tutorial**](https://www.guru99.com/how-to-install-python.html) | How to Install Python on Windows with Pycharm IDE |
| [**Tutorial**](https://www.guru99.com/creating-your-first-python-program.html) | Hello World: Creat your First Python Program |
| [**Tutorial**](https://www.guru99.com/print-python-examples.html) | How to Print in Python with Examples |
| [**Tutorial**](https://www.guru99.com/learn-python-main-function-with-examples-understand-main.html) | Python Main Function with Examples: Understand \_\_main\_\_ |
| [**Tutorial**](https://www.guru99.com/variables-in-python.html) | Python Variables: Declare, Concatenate, Global & Local |

### Python Data Structure

|  |  |
| --- | --- |
| [**Tutorial**](https://www.guru99.com/learning-python-strings-replace-join-split-reverse.html) | Python Strings: Replace, Join, Split, Reverse, Uppercase & Lowercase |
| [**Tutorial**](https://www.guru99.com/python-tuples-tutorial-comparing-deleting-slicing-keys-unpacking.html) | Python TUPLE - Pack, Unpack, Compare, Slicing, Delete, Key |
| [**Tutorial**](https://www.guru99.com/python-dictionary-beginners-tutorial.html) | Python Dictionary(Dict): Update, Cmp, Len, Sort, Copy, Items, str Example |
| [**Tutorial**](https://www.guru99.com/python-operators-complete-tutorial.html) | Python Operators: Arithmetic, Logical, Comparison, Assignment, Bitwise & Precedence |
| [**Tutorial**](https://www.guru99.com/functions-in-python.html) | Python Functions Examples: Call, Indentation, Arguments & Return Values |

### Python Fundamentals

|  |  |
| --- | --- |
| [**Tutorial**](https://www.guru99.com/if-loop-python-conditional-structures.html) | Python IF, ELSE, ELIF, Nested IF & Switch Case Statement |
| [**Tutorial**](https://www.guru99.com/python-loops-while-for-break-continue-enumerate.html) | Python For & While Loops: Enumerate, Break, Continue Statement |
| [**Tutorial**](https://www.guru99.com/python-class-objects-object-oriented-programming-oop-s.html) | Python OOPs: Class, Object, Inheritance and Constructor with Example |

### Must Know Stuff!

|  |  |
| --- | --- |
| [**Tutorial**](https://www.guru99.com/python-regular-expressions-complete-tutorial.html) | Python Regex Tutorial: re.match(),re.search(), re.findall(), re.split() |
| [**Tutorial**](https://www.guru99.com/date-time-and-datetime-classes-in-python.html) | Python DateTime, TimeDelta, Strftime(Format) with Examples |
| [**Tutorial**](https://www.guru99.com/calendar-in-python.html) | Python CALENDAR Tutorial with Example |
| [**Tutorial**](https://www.guru99.com/python-list-comprehension-sort-examples.html) | Python List: Comprehension, Apend, Sort, Length, Reverse EXAMPLES |
| [**Tutorial**](https://www.guru99.com/reading-and-writing-files-in-python.html) | Python File Handling: Create, Open, Append, Read, Write |
| [**Tutorial**](https://www.guru99.com/python-check-if-file-exists.html) | Python Check If File or Directory Exists |
| [**Tutorial**](https://www.guru99.com/python-copy-file.html) | Python COPY File using shutil.copy(), shutil.copystat() |
| [**Tutorial**](https://www.guru99.com/python-rename-file.html) | Python Rename File and Directory using os.rename() |
| [**Tutorial**](https://www.guru99.com/python-zip-file.html) | Python ZIP file with Example |
| [**Tutorial**](https://www.guru99.com/accessing-internet-data-with-python.html) | Python Internet Access using Urllib.Request and urlopen() |
| [**Tutorial**](https://www.guru99.com/manipulating-xml-with-python.html) | Python XML Parser Tutorial: Read xml file example(Minidom, ElementTree) |
| [**Tutorial**](https://www.guru99.com/python-vs-ruby-vs-php-vs-tcl-vs-perl-vs-java.html) | Python vs [JAVA](https://www.guru99.com/java-tutorial.html) vs [PHP](https://www.guru99.com/php-tutorials.html) vs [PERL](https://www.guru99.com/perl-tutorials.html) vs Ruby vs JavaScript vs C++ vs TCL |
| [**Tutorial**](https://www.guru99.com/pyqt-tutorial.html) | PyQt Tutorial: Python GUI Designer |
| [**Tutorial**](https://www.guru99.com/python-multithreading-gil-example.html) | Multithreading in Python with Global Interpreter Lock (GIL) Example |
| [**Tutorial**](https://www.guru99.com/python-lambda-function.html) | Python Lambda Functions with EXAMPLES |
| [**Tutorial**](https://www.guru99.com/python-interview-questions-answers.html) | Top 40 Python Interview Questions & Answers |
| [**Tutorial**](https://www.guru99.com/python-vs-php.html) | Python Vs PHP: What's the Difference? |
| [**Tutorial**](https://www.guru99.com/find-average-list-python.html) | Find AVERAGE of a List in Python with Example |
| [**Tutorial**](https://www.guru99.com/abs-in-python.html) | Python abs() Function: Absolute Value Examples |
| [**Tutorial**](https://www.guru99.com/round-function-python.html) | Python round() function with EXAMPLES |
| [**Tutorial**](https://www.guru99.com/timeit-python-examples.html) | Python Timeit() with Examples |
| [**Tutorial**](https://www.guru99.com/python-string-length-len.html) | Python string length | len() method Example |

### Python Data Science

|  |  |
| --- | --- |
| [**Tutorial**](https://www.guru99.com/scipy-tutorial.html) | Python SciPy Tutorial: Learn with Example |
| [**Tutorial**](https://www.guru99.com/python-csv.html) | Reading and Writing CSV Files in Python using CSV Module & Pandas |
| [**Tutorial**](https://www.guru99.com/python-json.html) | Python JSON: Encode(dump), Decode(load) json Data & File (Example) |
| [**Tutorial**](https://www.guru99.com/python-mysql-example.html) | Python with MySQL: Connect, Create Database, Table, Insert [Examples] |
| [**Tutorial**](https://www.guru99.com/python-unit-testing-guide.html) | PyUnit Tutorial: Python Unit Testing Framework (with Example) |
| [**Tutorial**](https://www.guru99.com/python-2-vs-python-3.html) | Python 2 vs Python 3: Key Differences |
| [**Tutorial**](https://www.guru99.com/facebook-login-using-python.html) | Facebook Login using Python: FB Login Example |
| [**Tutorial**](https://www.guru99.com/pytest-tutorial.html) | PyTest Tutorial: What is, Install, Fixture, Assertions |
| [**Tutorial**](https://www.guru99.com/django-tutorial.html) | Django Tutorials for Beginners |
| [**Tutorial**](https://www.guru99.com/online-python-courses.html) | 20 Best Online Python Courses |
| [**Tutorial**](https://www.guru99.com/web-scraping-tools.html) | 16 Best Web Scraping Tools for Data Extraction |
| [**Tutorial**](https://www.guru99.com/python-ide-code-editor.html) | BEST Python IDEs |
| [**Check**](https://www.guru99.com/python-tutorial-pdf.html) | Python Tutorial for Beginners PDF |

## Python Programming Characteristics

* It provides rich data types and easier to read syntax than any other programming languages
* It is a platform independent scripted language with full access to operating system API's
* Compared to other programming languages, it allows more run-time flexibility
* It includes the basic text manipulation facilities of Perl and Awk
* A module in Python may have one or more classes and free functions
* Libraries in Pythons are cross-platform compatible with Linux, Macintosh, and Windows
* For building large applications, Python can be compiled to byte-code
* Python supports functional and structured programming as well as OOP
* It supports interactive mode that allows interacting[Testing](https://www.guru99.com/software-testing.html)and debugging of snippets of code
* In Python, since there is no compilation step, editing, debugging and testing is fast.



Python Programming

## Python is commonly used by programmers to:

* Program video games
* Build Artificial Intelligence algorithms
* Program various scientific programs such as statistical models

In these Python tutorials, we will cover Python 2 and Python 3 Examples.

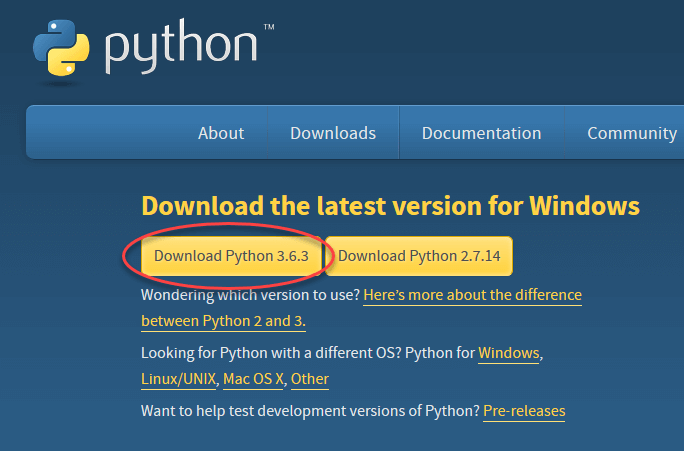
# How to Install Python on Windows [Pycharm IDE]

PyCharm is a cross-platform editor developed by JetBrains. Pycharm provides all the tools you need for productive Python development.

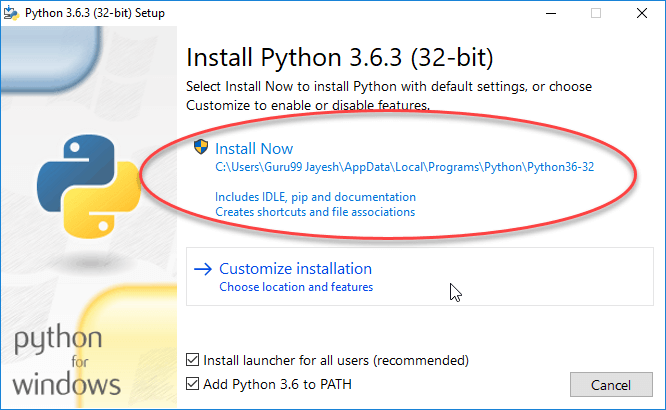
Below are the detailed steps for installing Python and PyCharm

## Installing Python

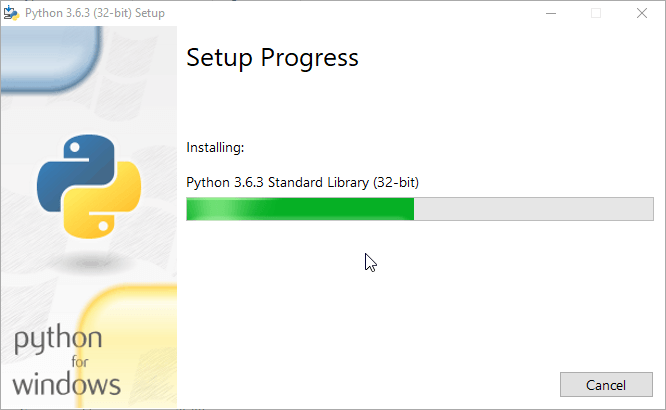
**Step 1)** To download and install Python visit the official website of Python http://www.python.org/downloads/ and choose your version. We have chosen Python version 3.6.3



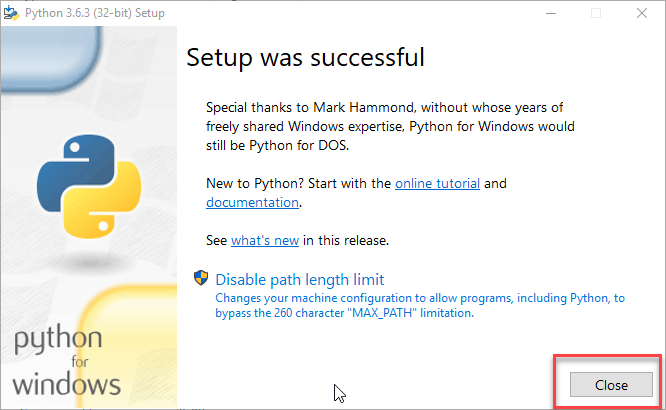
**Step 2)** Once the download is complete, run the exe for install Python. Now click on Install Now.



**Step 3)** You can see Python installing at this point.

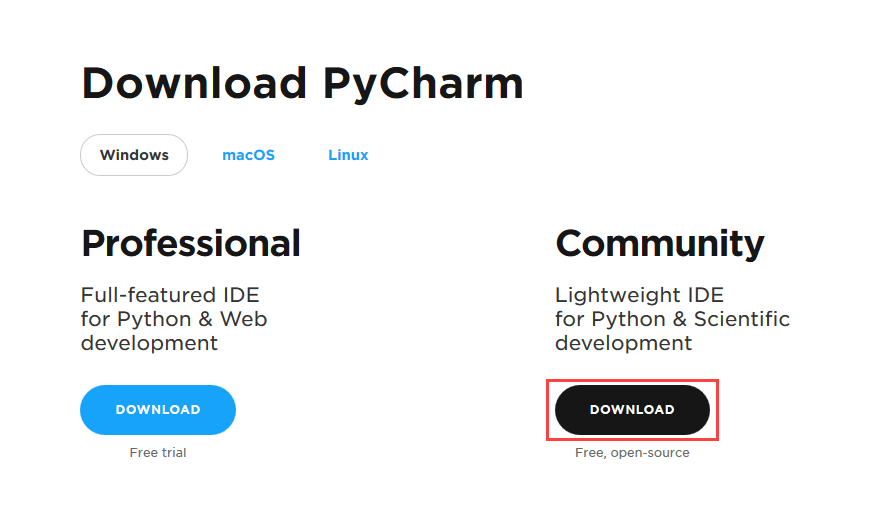


**Step 4)** When it finishes, you can see a screen that says the Setup was successful. Now click on "Close".

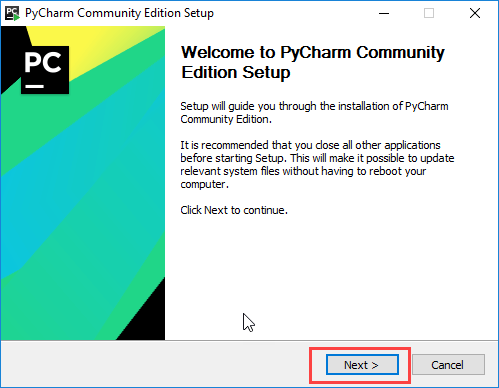


## Installing Pycharm

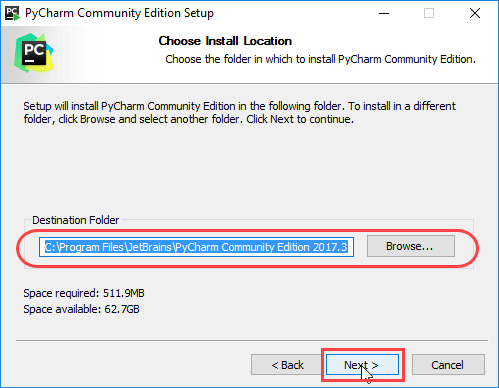
**Step 1)** To download PyCharm visit the website https://www.jetbrains.com/pycharm/download/ and Click the "DOWNLOAD" link under the Community Section.



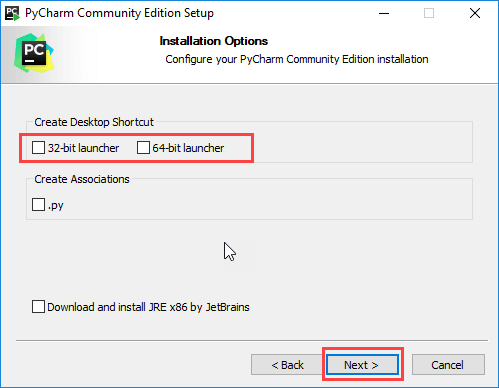
**Step 2)** Once the download is complete, run the exe for install PyCharm. The setup wizard should have started. Click “Next”.



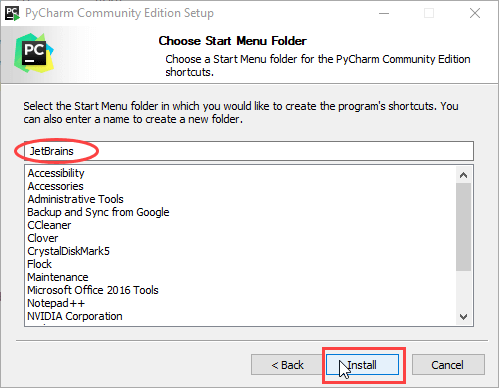
**Step 3)** On the next screen, Change the installation path if required. Click “Next”.



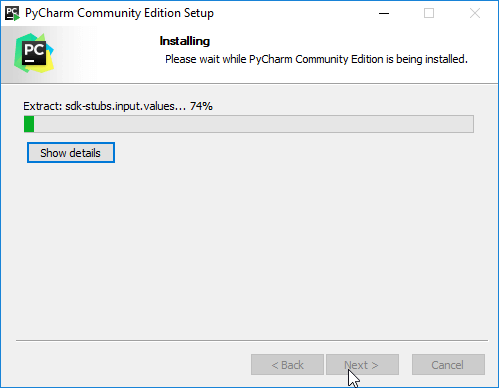
**Step 4)** On the next screen, you can create a desktop shortcut if you want and click on “Next”.



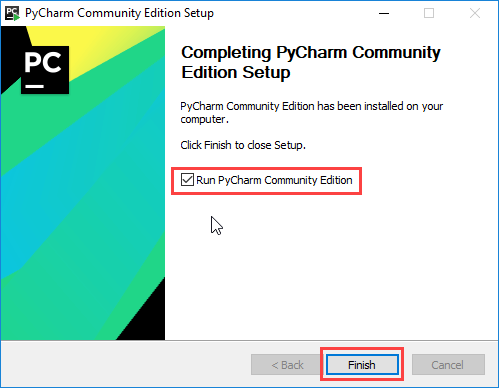
**Step 5)** Choose the start menu folder. Keep selected JetBrains and click on “Install”.



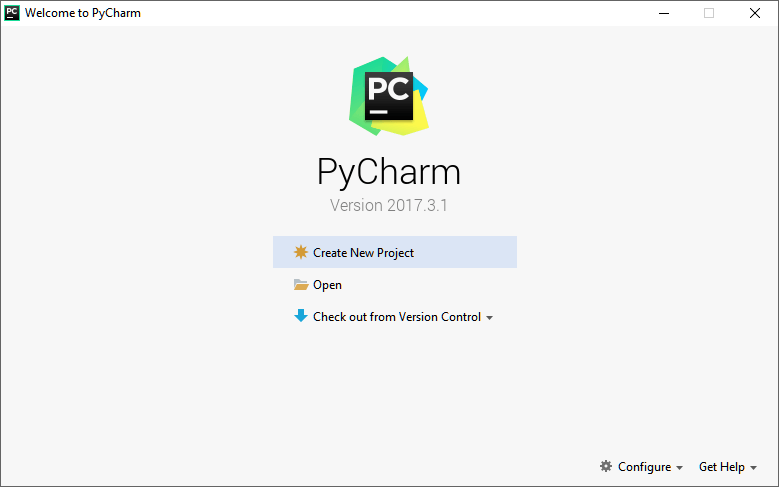
**Step 6)** Wait for the installation to finish.



**Step 7)** Once installation finished, you should receive a message screen that PyCharm is installed. If you want to go ahead and run it, click the “Run PyCharm Community Edition” box first and click “Finish”.



**Step 8)**After you click on "Finish," the Following screen will appear.

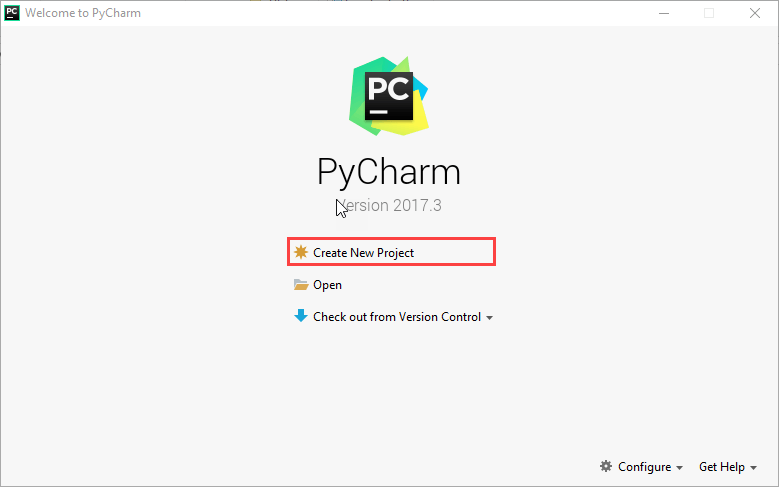


# Hello World: Create your First Python Program

In the last tutorial, we completed our Python installation and setup. It's time to create your first program.

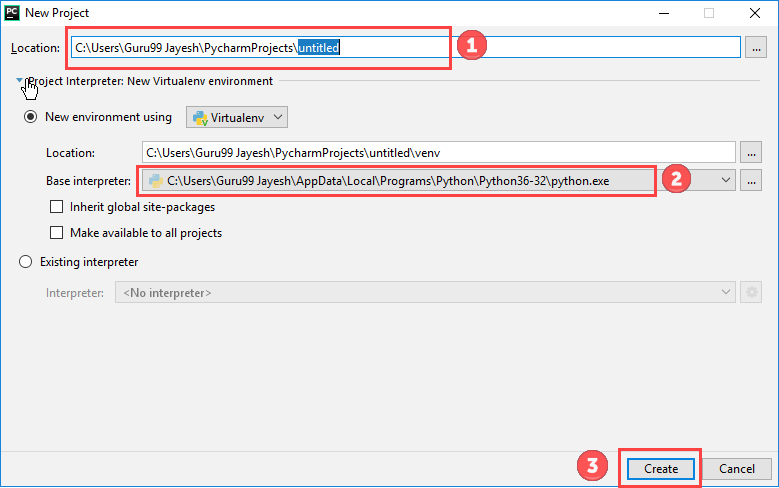
## Creating First Program

**Step 1)** Open PyCharm Editor. You can see the introductory screen for PyCharm. To create a new project, click on “Create New Project”.

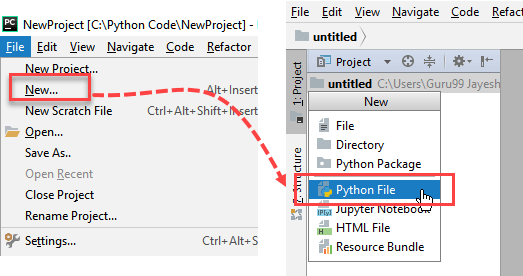


**Step 2)**You will need to select a location.

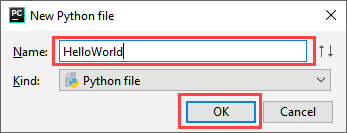
1. You can select the location where you want the project to be created. If you don’t want to change location than keep it as it is but at least change the name from “untitled” to something more meaningful, like “FirstProject”.
2. PyCharm should have found the Python interpreter you installed earlier.
3. Next Click the “Create” Button.



**Step 3)**Now Go up to the “File” menu and select “New”. Next, select “Python File”.



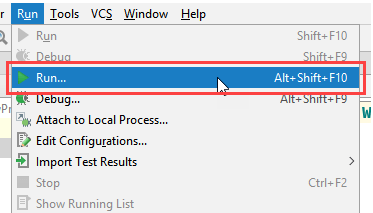
**Step 4)**A new pop up will appear. Now type the name of the file you want (Here we give “HelloWorld”) and hit “OK”.



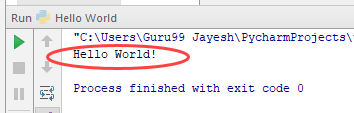
**Step 5)**Now type a simple program - print (‘Hello World!’).



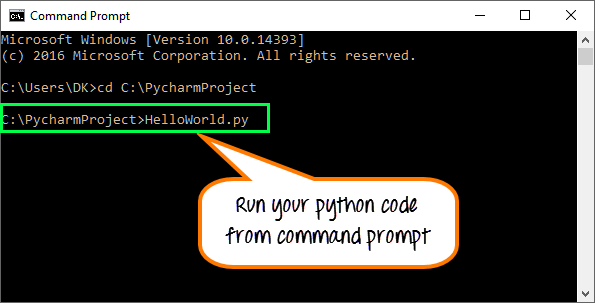
**Step 6)**Now Go up to the “Run” menu and select “Run” to run your program.

[](https://www.guru99.com/images/Pythonnew/PythonS3.6.png)

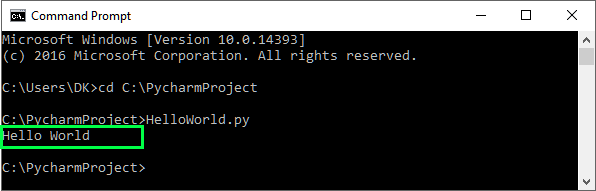
**Step 7)**You can see the output of your program at the bottom of the screen.

[](https://www.guru99.com/images/Pythonnew/Python3.5.png)

**Step 8)**Don't worry if you don't have Pycharm Editor installed, you can still run the code from the command prompt. Enter the correct path of a file in command prompt to run the program.



The output of the code would be

[](https://www.guru99.com/images/Pythonnew/Python3.7.png)

**Step 9)**If you are still not able to run the program, we have Python Editor for you.

Please run the given code at [Python Online Editor](https://www.guru99.com/execute-python-online.html)

print("Hello World")

# How to Print in Python with Examples

In this tutorial, you will learn-

* [How to print simple string?](https://www.guru99.com/print-python-examples.html#1)
* [How to print blank lines](https://www.guru99.com/print-python-examples.html#2)
* [Print end command](https://www.guru99.com/print-python-examples.html#2)

## How to print simple string?

More often then not you require to Print strings in your coding construct.

Here is how to do it in Python 3

**Example: 1**

To print the Welcome to Guru99, use the print () function as follows:

print ("Welcome to Guru99")

**Output:**

Welcome to Guru99

In Python 2, same example will look like

print "Welcome to Guru99"

**Example 2:**

If you want to print the name of five countries, you can write:

print("USA")

print("Canada")

print("Germany")

print("France")

print("Japan")

**Output:**

USA

Canada

Germany

France

Japan

## How to print blank lines

Sometimes you need to print one blank line in your Python program. Following are an example to perform this task.

**Example:**

Let us print 8 blank lines. You can type:

print (8 \* "\n")

or:

print ("\n\n\n\n\n\n\n\n\n")

**Here is the code**

print ("Welcome to Guru99")

print (8 \* "\n")

print ("Welcome to Guru99")

**Output**

Welcome to Guru99

Welcome to Guru99

## Print end command

By default, python's print() function ends with a newline. This function comes with a parameter called 'end.' The default value of this parameter is '\n,' i.e., the new line character. You can end a print statement with any character or string using this parameter. This is available in only in Python 3+

**Example 1:**

print ("Welcome to", end = ' ')

print ("Guru99", end = '!')

**Output:**

Welcome to Guru99!

**Example 2:**

# ends the output with '@.'

print("Python" , end = '@')

**Output:**

Python@

**Python Main Function with Examples: Understand \_\_main\_\_**

Before we jump more into Python coding, we get familiarize with Python Main function and its importance.

Consider the following code

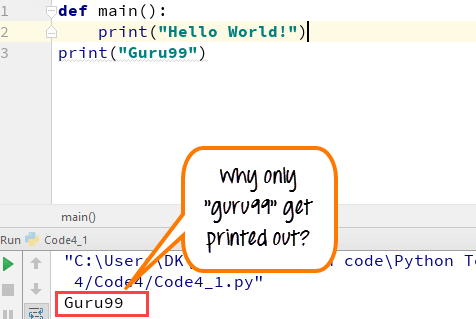
def main():

print "hello world!"

print "Guru99"

Here we got two pieces of print one is defined within a main function that is "Hello World" and the other is independent which is "Guru99". When you run the function def main ():

* Only "Guru99" prints out
* and not the code "Hello World."

[](https://www.guru99.com/images/Pythonnew/Python4_1.png)

It is because we did not declare the call **function "if\_\_name\_\_== "\_\_main\_\_".**

* When Python interpreter reads a source file, it will execute all the code found in it.
* When Python runs the "source file" as the main program, it sets the special variable (\_\_name\_\_) to have a value ("\_\_main\_\_").
* When you execute the main function, it will then read the "if" statement and checks whether \_\_name\_\_ does equal to \_\_main\_\_.
* In Python **"if\_\_name\_\_== "\_\_main\_\_"**allows you to run the Python files either as **reusable modules or standalone programs**.

Like C, Python uses == for comparison while = for assignment. Python interpreter uses the main function in two ways

* import: \_\_name\_\_= module's filename

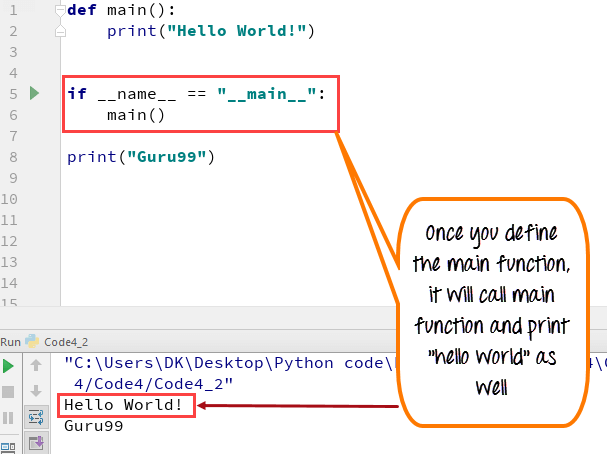
if statement==false, and the script in \_\_main\_\_ will not be executed

* direct run:\_\_name\_\_=\_\_main\_\_

if statement == True, and the script in \_main\_will be executed

* So when the code is executed, it will check for module name with "if."

It is important that after defining the main function, you call the code by if\_\_name\_\_== "\_\_main\_\_" and then run the code, only then you will get the output "hello world!" in the programming console as shown below.



**Note**: Make sure that after defining a main function, you leave some indent and not declare the code right below the def main(): function otherwise it will give indent error.

def main():

print("Hello World!")

if \_\_name\_\_== "\_\_main\_\_":

main()

print("Guru99")

Above examples are Python 3 codes, if you want to use Python 2, please consider following code

def main():

print "Hello World!"

if \_\_name\_\_== "\_\_main\_\_":

main()

print "Guru99"

In Python 3, you do not need to use if\_\_name. Following code also works

def main():

print("Hello World!")

main()

print("Guru99")

# Python Variables: Declare, Concatenate, Global & Local

## What is a Variable in Python?

A Python variable is a reserved memory location to store values. In other words, a variable in a python program gives data to the computer for processing.

Every value in Python has a datatype. Different data types in Python are Numbers, List, Tuple, Strings, Dictionary, etc. Variables can be declared by any name or even alphabets like a, aa, abc, etc.

In this tutorial, we will learn,

* [How to Declare and use a Variable](https://www.guru99.com/variables-in-python.html#1)
* [Re-declare a Variable](https://www.guru99.com/variables-in-python.html#2)
* [Concatenate Variables](https://www.guru99.com/variables-in-python.html#3)
* [Local & Global Variables](https://www.guru99.com/variables-in-python.html#4)
* [Delete a variable](https://www.guru99.com/variables-in-python.html#5)

## How to Declare and use a Variable

Let see an example. We will declare variable "a" and print it.

a=100

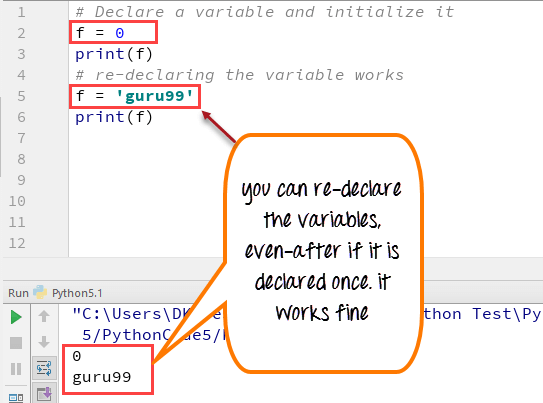
print a

## Re-declare a Variable

You can re-declare the variable even after you have declared it once.

Here we have variable initialized to f=0.

Later, we re-assign the variable f to value "guru99"

[](https://www.guru99.com/images/Pythonnew/Python5.1.png)

**Python 2 Example**

# Declare a variable and initialize it

f = 0

print f

# re-declaring the variable works

f = 'guru99'

print f

**Python 3 Example**

# Declare a variable and initialize it

f = 0

print(f)

# re-declaring the variable works

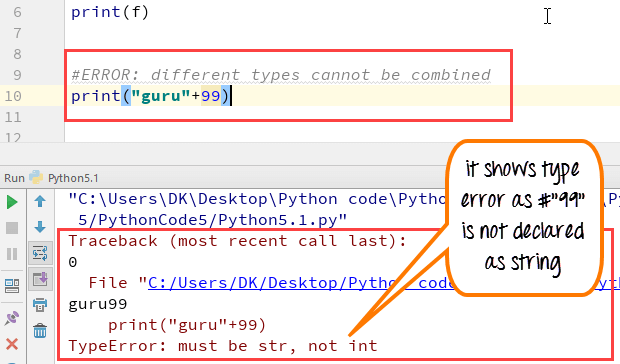
f = 'guru99'

print(f)

## Concatenate Variables

Let's see whether you can concatenate different data types like string and number together. For example, we will concatenate "Guru" with the number "99".

Unlike Java, which concatenates number with string without declaring number as string, Python requires declaring the number as string otherwise it will show a TypeError

[](https://www.guru99.com/images/Pythonnew/Python5.2.png)

For the following code, you will get undefined output -

a="Guru"

b = 99

print a+b

Once the integer is declared as string, it can concatenate both "Guru" + **str**("99")= "Guru99" in the output.

a="Guru"

b = 99

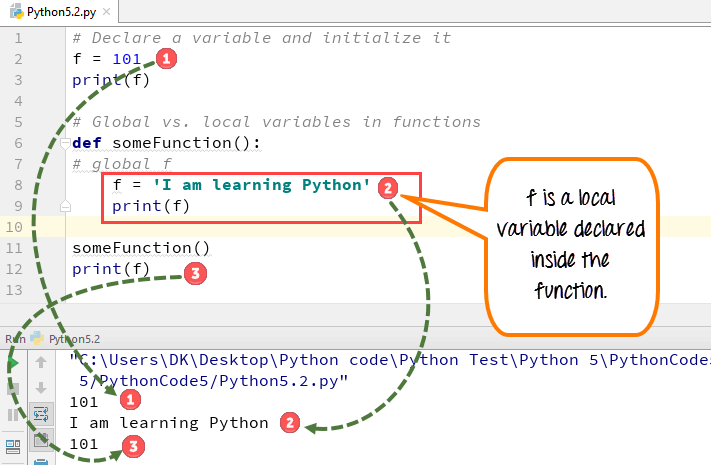
print(a+str(b))

## Local & Global Variables

In Python when you want to use the same variable for rest of your program or module you declare it a global variable, while if you want to use the variable in a specific function or method, you use a local variable.

Let's understand this difference between local and global variable with the below program.

1. Variable "f" is **global** in scope and is assigned value 101 which is printed in output
2. Variable f is again declared in function and assumes **local** scope. It is assigned value "I am learning Python." which is printed out as an output. This variable is different from the global variable "f" define earlier
3. Once the function call is over, the local variable f is destroyed. At line 12, when we again, print the value of "f" is it displays the value of global variable f=101

[](https://www.guru99.com/images/Pythonnew/Python5.3.png)

**Python 2 Example**

# Declare a variable and initialize it

f = 101

print f

# Global vs. local variables in functions

def someFunction():

# global f

f = 'I am learning Python'

print f

someFunction()

print f

**Python 3 Example**

# Declare a variable and initialize it

f = 101

print(f)

# Global vs. local variables in functions

def someFunction():

# global f

f = 'I am learning Python'

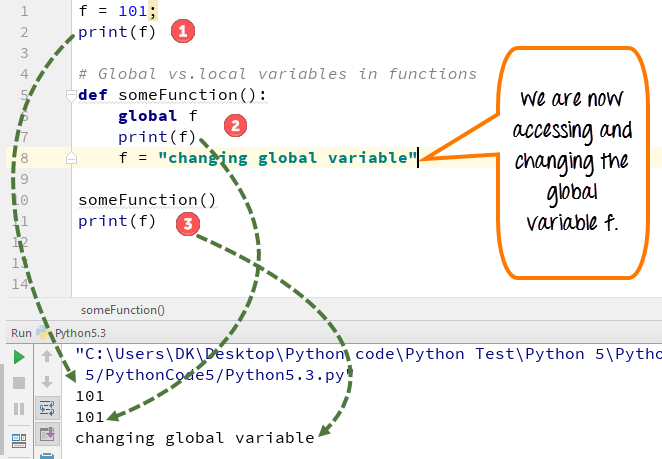
print(f)

someFunction()

print(f)

Using the keyword **global,**you can reference the global variable inside a function.

1. Variable "f" is **global** in scope and is assigned value 101 which is printed in output
2. Variable f is declared using the keyword **global**. This is **NOT** a **local variable**, but the same global variable declared earlier. Hence when we print its value, the output is 101
3. We changed the value of "f" inside the function. Once the function call is over, the changed value of the variable "f" persists. At line 12, when we again, print the value of "f" is it displays the value "changing global variable"

[](https://www.guru99.com/images/Pythonnew/Python5.4.png)

**Python 2 Example**

f = 101;

print f

# Global vs.local variables in functions

def someFunction():

global f

print f

f = "changing global variable"

someFunction()

print f

**Python 3 Example**

f = 101;

print(f)

# Global vs.local variables in functions

def someFunction():

global f

print(f)

f = "changing global variable"

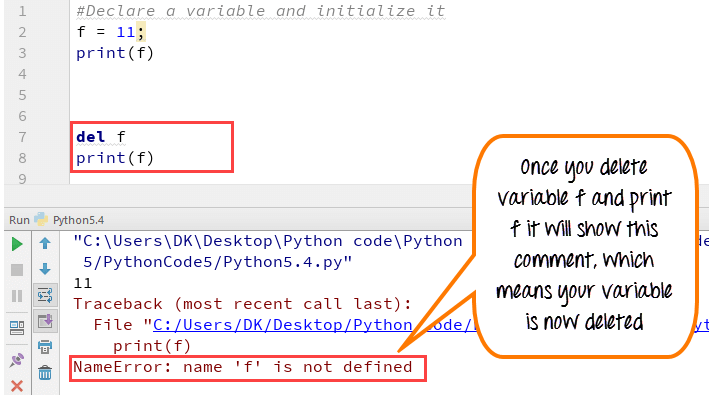
someFunction()

print(f)

## Delete a variable

You can also delete variable using the command **del** "variable name".

In the example below, we deleted variable f, and when we proceed to print it, we get error "**variable name is not defined**" which means you have deleted the variable.

[](https://www.guru99.com/images/Pythonnew/Python5.5.png)

f = 11;

print(f)

del f

print(f)

## Summary:

* Variables are referred to "envelop" or "buckets" where information can be maintained and referenced. Like any other programming language Python also uses a variable to store the information.
* Variables can be declared by any name or even alphabets like a, aa, abc, etc.
* Variables can be re-declared even after you have declared them for once
* In Python you cannot concatenate string with number directly, you need to declare them as a separate variable, and after that, you can concatenate number with string
* Declare local variable when you want to use it for current function
* Declare Global variable when you want to use the same variable for rest of the program
* To delete a variable, it uses keyword "del".

# Python Strings: Replace, Join, Split, Reverse, Uppercase & Lowercase

In Python everything is object and string are an object too. Python string can be created simply by enclosing characters in the double quote.

For example:

var = **"Hello World!"**

In this tutorial, we will learn -

* [Accessing Values in Strings](https://www.guru99.com/learning-python-strings-replace-join-split-reverse.html#1)
* [Various String Operators](https://www.guru99.com/learning-python-strings-replace-join-split-reverse.html#2)
* [Some more examples](https://www.guru99.com/learning-python-strings-replace-join-split-reverse.html#3)
* [Python String replace() Method](https://www.guru99.com/learning-python-strings-replace-join-split-reverse.html#4)
* [Changing upper and lower case strings](https://www.guru99.com/learning-python-strings-replace-join-split-reverse.html#5)
* [Using "join" function for the string](https://www.guru99.com/learning-python-strings-replace-join-split-reverse.html#6)
* [Reversing String](https://www.guru99.com/learning-python-strings-replace-join-split-reverse.html#7)
* [Split Strings](https://www.guru99.com/learning-python-strings-replace-join-split-reverse.html#8)

## Accessing Values in Strings

**Python does not support a character type**, these are treated as strings of length one, also considered as substring.

We use square brackets for slicing along with the index or indices to obtain a substring.

var1 = "Guru99!"

var2 = "Software Testing"

print ("var1[0]:",var1[0])

print ("var2[1:5]:",var2[1:5])

## Various String Operators

There are various string operators that can be used in different ways like concatenating different string.

Suppose if a=guru and b=99 then a+b= "guru99". Similarly, if you are using a\*2, it will "GuruGuru". Likewise, you can use other operators in string.

| **Operator** | **Description** | **Example** |  |
| --- | --- | --- | --- |
| [] | Slice- it gives the letter from the given index | a[1] will give "u" from the word Guru as such ( 0=G, 1=u, 2=r and 3=u) | x="Guru"  print x[1] |
| [ : ] | Range slice-it gives the characters from the given range | x [1:3] it will give "ur" from the word Guru. Remember it will not consider 0 which is G, it will consider word after that is ur. | x="Guru"  print x[1:3] |
| in | Membership-returns true if a letter exist in the given string | u is present in word Guru and hence it will give 1 (True) | x="Guru"  print "u" in x |
| not in | Membership-returns true if a letter exist is not in the given string | l not present in word Guru and hence it will give 1 | x="Guru"  print "l" not in x |
| r/R | Raw string suppresses actual meaning of escape characters. | Print r'\n' prints \n and print R'/n' prints \n |  |
| % - Used for string format | %r - It insert the canonical string representation of the object (i.e., repr(o)) %s- It insert the presentation string representation of the object (i.e., str(o)) %d- it will format a number for display | The output of this code will be "guru 99". | name = 'guru'  number = 99  print'%s %d' % (name,number) |
| + | It concatenates 2 strings | It concatenate strings and gives the result | x="Guru"  y="99"  print x+y |
| \* | Repeat | It prints the character twice. | x="Guru"  y="99"  print x\*2 |

## Some more examples

You can update Python String by re-assigning a variable to another string. The new value can be related to previous value or to a completely different string all together.

x = "Hello World!"

print(x[:6])

print(x[0:6] + "Guru99")

**Note** : - Slice:6 or 0:6 has the same effect

## Python String replace() Method

The method replace() returns a copy of the string in which the values of old string have been replaced with the new value.

oldstring = 'I like Guru99'

newstring = oldstring.replace('like', 'love')

print(newstring)

## Changing upper and lower case strings

In Python, you can even change the string to upper case or lower case.

string="python at guru99"

print(string.upper())

Likewise, you can also do for other function as well like capitalize

string="python at guru99"

print(string.capitalize())

You can also convert your string to lower case

string="PYTHON AT GURU99"

print(string.lower())

## Using "join" function for the string

The join function is a more flexible way for concatenating string. With join function, you can add any character into the string.

For example, if you want to add a colon (:) after every character in the string "Python" you can use the following code.

print(":".join("Python"))

## Reversing String

By using the reverse function, you can reverse the string. For example, if we have string "12345" and then if you apply the code for the reverse function as shown below.

string="12345"

print(''.join(reversed(string)))

## Split Strings

Split strings is another function that can be applied in Python let see for string "guru99 career guru99". First here we will split the string by using the command word.split and get the result.

word="guru99 career guru99"

print(word.split(' '))

To understand this better we will see one more example of split, instead of space (' ') we will replace it with ('r') and it will split the string wherever 'r' is mentioned in the string

word="guru99 career guru99"

print(word.split('r'))

**Important Note:**

**In Python, Strings are immutable.**

**Consider the following code**

x = "Guru99"

x.replace("Guru99","Python")

print(x)

will still return Guru99. This is because x.replace("Guru99","Python") returns **a copy of X with replacements made**

**You will need to use the following code to observe changes**

x = "Guru99"

x = x.replace("Guru99","Python")

print(x)

Above codes are Python 3 examples, If you want to run in Python 2 please consider following code.

**Python 2 Example**

#Accessing Values in Strings

var1 = "Guru99!"

var2 = "Software Testing"

print "var1[0]:",var1[0]

print "var2[1:5]:",var2[1:5]

#Some more examples

x = "Hello World!"

print x[:6]

print x[0:6] + "Guru99"

#Python String replace() Method

oldstring = 'I like Guru99'

newstring = oldstring.replace('like', 'love')

print newstring

#Changing upper and lower case strings

string="python at guru99"

print string.upper()

string="python at guru99"

print string.capitalize()

string="PYTHON AT GURU99"

print string.lower()

#Using "join" function for the string

print":".join("Python")

#Reversing String

string="12345"

print''.join(reversed(string))

#Split Strings

word="guru99 career guru99"

print word.split(' ')

word="guru99 career guru99"

print word.split('r')

x = "Guru99"

x.replace("Guru99","Python")

print x

x = "Guru99"

x = x.replace("Guru99","Python")

print x

Python has introduced a .format function which does way with using the cumbersome %d and so on for string formatting.

## Summary:

Since Python is an object-oriented programming language, many functions can be applied to Python objects. A notable feature of Python is its indenting source statements to make the code easier to read.

* Accessing values through slicing - square brackets are used for slicing along with the index or indices to obtain a substring.
  + In slicing, if range is declared [1:5], it can actually fetch the value from range [1:4]
* You can update Python String by re-assigning a variable to another string
* Method replace() returns a copy of the string in which the occurrence of old is replaced with new.
  + Syntax for method replace: oldstring.replace("value to change","value to be replaced")
* String operators like [], [ : ], in, Not in, etc. can be applied to concatenate the string, fetching or inserting specific characters into the string, or to check whether certain character exist in the string
* Other string operations include
  + Changing upper and lower case
  + Join function to glue any character into the string
  + Reversing string
  + Split string

# Python TUPLE - Pack, Unpack, Compare, Slicing, Delete, Key

### What is Tuple in Python?

A tuple is just like a list of a sequence of immutable python objects. The difference between list and tuple is that list are declared in square brackets and can be changed while **tuple is declared in parentheses** and cannot be changed. However, you can take portions of existing tuples to make new tuples.

**Tuple Syntax**

Tup = ('Jan','feb','march')

To write an empty tuple, you need to write as two parentheses containing nothing-

tup1 = ();

For writing tuple for a single value, you need to include a comma, even though there is a single value. Also at the end you need to write semicolon as shown below.

Tup1 = (50,);

Tuple indices begin at 0, and they can be concatenated, sliced and so on.

In this tutorial, we will learn-

* [Packing and Unpacking](https://www.guru99.com/python-tuples-tutorial-comparing-deleting-slicing-keys-unpacking.html#1)
* [Comparing tuples](https://www.guru99.com/python-tuples-tutorial-comparing-deleting-slicing-keys-unpacking.html#2)
* [Using tuples as keys in dictionaries](https://www.guru99.com/python-tuples-tutorial-comparing-deleting-slicing-keys-unpacking.html#3)
* [Deleting Tuples](https://www.guru99.com/python-tuples-tutorial-comparing-deleting-slicing-keys-unpacking.html#4)
* [Slicing of Tuple](https://www.guru99.com/python-tuples-tutorial-comparing-deleting-slicing-keys-unpacking.html#5)
* [Built-in functions with Tuple](https://www.guru99.com/python-tuples-tutorial-comparing-deleting-slicing-keys-unpacking.html#6)
* [Advantages of tuple over list](https://www.guru99.com/python-tuples-tutorial-comparing-deleting-slicing-keys-unpacking.html#7)

**Tuple Assignment**

Python has tuple assignment feature which enables you to assign more than one variable at a time. In here, we have assigned tuple 1 with the persons information like name, surname, birth year, etc. and another tuple 2 with the values in it like number (1,2,3,….,7).

For Example,

(name, surname, birth year, favorite movie and year, profession, birthplace) = Robert

Here is the code,

tup1 = ('Robert', 'Carlos','1965','Terminator 1995', 'Actor','Florida');

tup2 = (1,2,3,4,5,6,7);

print(tup1[0])

print(tup2[1:4])

* Tuple 1 includes list of information of Robert
* Tuple 2 includes list of numbers in it
* We call the value for [0] in tuple and for tuple 2 we call the value between 1 and 4
* Run the code- It gives name Robert for first tuple while for second tuple it gives number (2,3 and 4)

## Packing and Unpacking

In packing, we place value into a new tuple while in unpacking we extract those values back into variables.

x = ("Guru99", 20, "Education") # tuple packing

(company, emp, profile) = x # tuple unpacking

print(company)

print(emp)

print(profile)

## Comparing tuples

A comparison operator in Python can work with tuples.

The comparison starts with a first element of each tuple. If they do not compare to =,< or > then it proceed to the second element and so on.

It starts with comparing the first element from each of the tuples

Let's study this with an example-

**#case 1**

a=(5,6)

b=(1,4)

if (a>b):print("a is bigger")

else: print("b is bigger")

**#case 2**

a=(5,6)

b=(5,4)

if (a>b):print("a is bigger")

else: print ("b is bigger")

**#case 3**

a=(5,6)

b=(6,4)

if (a>b):print("a is bigger")

else: print("b is bigger")

**Case1:** Comparison starts with a first element of each tuple. In this case 5>1, so the output a is bigger

**Case 2:** Comparison starts with a first element of each tuple. In this case 5>5 which is inconclusive. So it proceeds to the next element. 6>4, so the output a is bigger

**Case 3:** Comparison starts with a first element of each tuple. In this case 5>6 which is false. So it goes into the else block and prints "b is bigger."

## Using tuples as keys in dictionaries

Since tuples are hashable, and list is not, we must use tuple as the key if we need to create a composite key to use in a dictionary.

**Example**: We would come across a composite key if we need to create a telephone directory that maps, first-name, last-name, pairs of telephone numbers, etc. Assuming that we have declared the variables as last and first number, we could write a dictionary assignment statement as shown below:

directory[last,first] = number

Inside the brackets, the expression is a tuple. We could use tuple assignment in a for loop to navigate this dictionary.

for last, first in directory:

print first, last, directory[last, first]

This loop navigates the keys in the directory, which are tuples. It assigns the elements of each tuple to last and first and then prints the name and corresponding telephone number.

**Tuples and dictionary**

Dictionary can return the list of tuples by calling items, where each tuple is a key value pair.

a = {'x':100, 'y':200}

b = list(a.items())

print(b)

## Deleting Tuples

Tuples are immutable and cannot be deleted. You cannot delete or remove items from a tuple. But deleting tuple entirely is possible by using the keyword

del

## Slicing of Tuple

To fetch specific sets of sub-elements from tuple or list, we use this unique function called slicing. Slicing is not only applicable to tuple but also for array and list.

x = ("a", "b","c", "d", "e")

print(x[2:4])

The output of this code will be ('c', 'd').

**Here is the Python 2 Code for all above example**

tup1 = ('Robert', 'Carlos','1965','Terminator 1995', 'Actor','Florida');

tup2 = (1,2,3,4,5,6,7);

print tup1[0]

print tup2[1:4]

#Packing and Unpacking

x = ("Guru99", 20, "Education") # tuple packing

(company, emp, profile) = x # tuple unpacking

print company

print emp

print profile

#Comparing tuples

#case 1

a=(5,6)

b=(1,4)

if (a>b):print "a is bigger"

else: print "b is bigger"

#case 2

a=(5,6)

b=(5,4)

if (a>b):print "a is bigger"

else: print "b is bigger"

#case 3

a=(5,6)

b=(6,4)

if (a>b):print "a is bigger"

else: print "b is bigger"

#Tuples and dictionary

a = {'x':100, 'y':200}

b = a.items()

print b

#Slicing of Tuple

x = ("a", "b","c", "d", "e")

print x[2:4]

## Built-in functions with Tuple

To perform different task, tuple allows you to use many built-in functions like all(), any(), enumerate(), max(), min(), sorted(), len(), tuple(), etc.

## Advantages of tuple over list

* Iterating through tuple is faster than with list, since tuples are immutable.
* Tuples that consist of immutable elements can be used as key for dictionary, which is not possible with list
* If you have data that is immutable, implementing it as tuple will guarantee that it remains write-protected

**Summary**:

Python has tuple assignment feature which enables you to assign more than one variable at a time.

* Packing and Unpacking of Tuples
  + In packing, we place value into a new tuple while in unpacking we extract those values back into variables.
* A comparison operator in Python can work with tuples.
* Using tuples as keys in dictionaries
  + Tuples are hashable, and list are not
  + We must use tuple as the key if we need to create a composite key to use in a dictionary
  + Dictionary can return the list of tuples by calling items, where each tuple is a key value pair
* Tuples are immutable and cannot be deleted. You cannot delete or remove items from a tuple. But deleting tuple entirely is possible by using the keyword "del"
* To fetch specific sets of sub-elements from tuple or list, we use this unique function called slicing

# Python Dictionary(Dict): Update, Cmp, Len, Sort, Copy, Items, str Example

Dictionaries are another example of a data structure. A dictionary is used to map or associate things you want to store the keys you need to get them. A dictionary in Python is just like a dictionary in the real world. Python Dictionary are defined into two elements Keys and Values.

* Keys will be a single element
* Values can be a list or list within a list, numbers, etc.

In this tutorial, we are going to learn,

* [Python Dictionary Methods](https://www.guru99.com/python-dictionary-beginners-tutorial.html#1)
* [Copying dictionary](https://www.guru99.com/python-dictionary-beginners-tutorial.html#2)
* [Updating Dictionary](https://www.guru99.com/python-dictionary-beginners-tutorial.html#3)
* [Delete Keys from the dictionary](https://www.guru99.com/python-dictionary-beginners-tutorial.html#4)
* [Dictionary items() Method](https://www.guru99.com/python-dictionary-beginners-tutorial.html#5)
* [Sorting the Dictionary](https://www.guru99.com/python-dictionary-beginners-tutorial.html#6)
* [Python Dictionary in-built Functions](https://www.guru99.com/python-dictionary-beginners-tutorial.html#7)
* [Dictionary len() Method](https://www.guru99.com/python-dictionary-beginners-tutorial.html#8)
* [Variable Types](https://www.guru99.com/python-dictionary-beginners-tutorial.html#9)
* [Python List cmp() Method](https://www.guru99.com/python-dictionary-beginners-tutorial.html#10)
* [Dictionary Str(dict)](https://www.guru99.com/python-dictionary-beginners-tutorial.html#11)

**Syntax for Python Dictionary**:

Dict = { ' Tim': 18, xyz,.. }

Dictionary is listed in curly brackets, inside these curly brackets, keys and values are declared. Each key is separated from its value by a colon (:) while each element is separated by commas.

**Properties of Dictionary Keys**

There are two important points while using dictionary keys

* More than one entry per key is not allowed ( no duplicate key is allowed)
* The values in the dictionary can be of any type while the keys must be immutable like numbers, tuples or strings.
* Dictionary keys are case sensitive- Same key name but with the different case are treated as different keys in Python dictionaries.

**Python 2 Example**

Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}

print (Dict['Tiffany'])

**Python 3 Example**

Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}

print((Dict['Tiffany']))

* In code, we have dictionary name "Dict"
* We declared the name and age of the person in the dictionary, where name is "Keys" and age is the"value"
* Now run the code
* It retrieves the age of tiffany from the dictionary.

## Python Dictionary Methods

### Copying dictionary

You can also copy the entire dictionary to new dictionary. For example, here we have copied our original dictionary to new dictionary name "Boys" and "Girls".

**Python 2 Example**

Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}

Boys = {'Tim': 18,'Charlie':12,'Robert':25}

Girls = {'Tiffany':22}

studentX=Boys.copy()

studentY=Girls.copy()

print studentX

print studentY

**Python 3 Example**

Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}

Boys = {'Tim': 18,'Charlie':12,'Robert':25}

Girls = {'Tiffany':22}

studentX=Boys.copy()

studentY=Girls.copy()

print(studentX)

print(studentY)

* We have the original dictionary (Dict) with the name and age of the boys and girls together
* But we want boys list separate from girls list, so we defined the element of boys and girls in a separate dictionary name "Boys" and "Girls."
* Now again we have created new dictionary name "studentX" and "studentY", where all the keys and values of boy dictionary are copied into studentX, and the girls will be copied in studentY
* So now you don't have to look into the whole list in main dictionary( Dict) to check who is boy and who is girl, you just have to print studentX if you want boys list and StudentY if you want girls list
* So, when you run the studentX and studentY dictionary, it will give all the element present in the dictionary of "boys" and "girls" separately

### Updating Dictionary

You can also update a dictionary by adding a new entry or a key-value pair to an existing entry or by deleting an existing entry. Here in the example we will add another name "Sarah" to our existing dictionary.

**Python 2 Example**

Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}

Dict.update({"Sarah":9})

print Dict

**Python 3 Example**

Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}

Dict.update({"Sarah":9})

print(Dict)

* Our existing dictionary "Dict" does not have the name "Sarah."
* We use the method Dict.update to add Sarah to our existing dictionary
* Now run the code, it adds Sarah to our existing dictionary

### Delete Keys from the dictionary

Python dictionary gives you the liberty to delete any element from the dictionary list. Suppose you don't want the name Charlie in the list, so you can delete the key element by following code.

**Python 2 Example**

Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}

del Dict ['Charlie']

print Dict

**Python 3 Example**

Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}

del Dict ['Charlie']

print(Dict)

When you run this code, it should print the dictionary list without Charlie.

* We used the code del Dict
* When code executed, it has deleted the Charlie from the main dictionary

### Dictionary items() Method

The items() method returns a list of tuple pairs (Keys, Value) in the dictionary.

**Python 2 Example**

Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}

print "Students Name: %s" % Dict.items()

**Python 3 Example**

Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}

print("Students Name: %s" % list(Dict.items()))

* We use the code items() method for our Dict.
* When code was executed, it returns a list of items ( keys and values) from the dictionary

**Check if a given key already exists in a dictionary**

For a given list, you can also check whether our child dictionary exists in a main dictionary or not. Here we have two sub-dictionaries "Boys" and "Girls", now we want to check whether our dictionary Boys exist in our main "Dict" or not. For that, we use the forloop method with else if method.

**Python 2 Example**

Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}

Boys = {'Tim': 18,'Charlie':12,'Robert':25}

Girls = {'Tiffany':22}

for key in Dict.keys():

if key in Boys.keys():

print True

else:

print False

**Python 3 Example**

Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}

Boys = {'Tim': 18,'Charlie':12,'Robert':25}

Girls = {'Tiffany':22}

for key in list(Dict.keys()):

if key in list(Boys.keys()):

print(True)

else:

print(False)

* The forloop in code checks each key in the main dictionary for Boys keys
* If it exists in the main dictionary, it should print true or else it should print false
* When you execute the code, it will print "True" for three times, as we got three elements in our "Boys" dictionary
* So it indicates that the "Boys" exist in our main dictionary (Dict)

### Sorting the Dictionary

In the dictionary, you can also sort the elements. For example, if we want to print the name of the elements of our dictionary alphabetically we have to use the forloop. It will sort each element of dictionary accordingly.

**Python 2 Example**

Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}

Boys = {'Tim': 18,'Charlie':12,'Robert':25}

Girls = {'Tiffany':22}

Students = Dict.keys()

Students.sort()

for S in Students:

print":".join((S,str(Dict[S])))

**Python 3 Example**

Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}

Boys = {'Tim': 18,'Charlie':12,'Robert':25}

Girls = {'Tiffany':22}

Students = list(Dict.keys())

Students.sort()

for S in Students:

print(":".join((S,str(Dict[S]))))

* We declared the variable students for our dictionary "Dict."
* Then we use the code Students.sort, which will sort the element inside our dictionary
* But to sort each element in dictionary, we run the forloop by declaring variable S
* Now, when we execute the code the forloop will call each element from the dictionary, and it will print the string and value in an order

## Python Dictionary in-built Functions

### Dictionary len() Method

The len() function gives the number of pairs in the dictionary.

For example,

**Python 2 Example**

Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}

print "Length : %d" % len (Dict)

**Python 3 Example**

Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}

print("Length : %d" % len (Dict))

When len (Dict) function is executed it gives the output at "4" as there are four elements in our dictionary

### Variable Types

Python does not require to explicitly declare the reserve memory space; it happens automatically. The assign values to variable "=" equal sign are used. The code to determine the variable type is " %type (Dict)."

**Python 2 Example**

Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}

print "variable Type: %s" %type (Dict)

**Python 3 Example**

Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}

print("variable Type: %s" %type (Dict))

* Use the code %type to know the variable type
* When code was executed, it tells a variable type is a dictionary

### Python List cmp() Method

The compare method cmp() is used in Python to compare values and keys of two dictionaries. If method returns 0 if both dictionaries are equal, 1 if dic1 > dict2 and -1 if dict1 < dict2.

**Python 2 Example**

Boys = {'Tim': 18,'Charlie':12,'Robert':25}

Girls = {'Tiffany':22}

print cmp(Girls, Boys)

**Python 3 Example**

cmp is not supported in Python 3

* We have two dictionary name "Boys" and "Girls."
* Which ever you declare first in code "cmp(Girls, Boys)" will be considered as dictionary 1. In our case, we declared "Girls" first, so it will be considered as dictionary 1 and boys as dictionary 2
* When code is executed it prints out -1, It indicates that our dictionary 1 is less than dictionary 2.

### Dictionary Str(dict)

With Str() method, you can make a dictionary into a printable string format.

**Python 2 Example**

Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}

print "printable string:%s" % str (Dict)

**Python 3 Example**

Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}

print("printable string:%s" % str (Dict))

* Use the code % str (Dict)
* It will return the dictionary elements into a printable string format

**Here is the list of all Dictionary Methods**

|  |  |  |
| --- | --- | --- |
| **Method** | **Description** | **Syntax** |
| copy() | Copy the entire dictionary to new dictionary | dict.copy() |
| update() | Update a dictionary by adding a new entry or a key-value pair to an existing entry or by deleting an existing entry. | Dict.update([other]) |
| items() | Returns a list of tuple pairs (Keys, Value) in the dictionary. | dictionary.items() |
| sort() | You can sort the elements | dictionary.sort() |
| len() | Gives the number of pairs in the dictionary. | len(dict) |
| cmp() | Compare values and keys of two dictionaries | cmp(dict1, dict2) |
| Str() | Make a dictionary into a printable string format | Str(dict) |

### Summary:

Dictionaries in a programming language is a type of data-structure used to store information connected in someway. Python Dictionary are defined into two elements Keys and Values. Dictionaries do not store their information in any particular order, so you may not get your information back in the same order you entered it.

* Keys will be a single element
* Values can be a list or list within a list, numbers, etc.
* More than one entry per key is not allowed ( no duplicate key is allowed)
* The values in the dictionary can be of any type while the keys must be immutable like numbers, tuples or strings.
* Dictionary keys are case sensitive- Same key name but with the different case are treated as different keys in Python dictionaries.

# Python Operators: Arithmetic, Logical, Comparison, Assignment, Bitwise & Precedence

Operators are used to perform operations on values and variables. Operators can manipulate individual items and returns a result. The data items are referred as operands or arguments. Operators are either represented by keywords or special characters. For example, for identity operators we use keyword "is" and "is not".

In this tutorial, we going to learn various operators

* [Arithmetic Operators](https://www.guru99.com/python-operators-complete-tutorial.html#1)
* [Comparison Operators](https://www.guru99.com/python-operators-complete-tutorial.html#2)
* [Python Assignment Operators](https://www.guru99.com/python-operators-complete-tutorial.html#3)
* [Logical Operators or Bitwise Operators](https://www.guru99.com/python-operators-complete-tutorial.html#4)
* [Membership Operators](https://www.guru99.com/python-operators-complete-tutorial.html#5)
* [Identity Operators](https://www.guru99.com/python-operators-complete-tutorial.html#6)
* [Operator precedence](https://www.guru99.com/python-operators-complete-tutorial.html#7)

## Arithmetic Operators

Arithmetic Operators perform various arithmetic calculations like addition, subtraction, multiplication, division, %modulus, exponent, etc. There are various methods for arithmetic calculation in Python like you can use the eval function, declare variable & calculate, or call functions.

**Example**: For arithmetic operators we will take simple example of addition where we will add two-digit 4+5=9

x= 4

y= 5

print(x + y)

Similarly, you can use other arithmetic operators like for multiplication(\*), division (/), substraction (-), etc.

## Comparison Operators

These operators compare the values on either side of the operand and determine the relation between them. It is also referred as relational operators. Various comparison operators are ( ==, != , <>, >,<=, etc)

**Example**: For comparison operators we will compare the value of x to the value of y and print the result in true or false. Here in example, our value of x = 4 which is smaller than y = 5, so when we print the value as x>y, it actually compares the value of x to y and since it is not correct, it returns false.

x = 4

y = 5

print(('x > y is',x>y))

Likewise, you can try other comparison operators (x < y, x==y, x!=y, etc.)

## Python Assignment Operators

Python assignment operators are used for assigning the value of the right operand to the left operand. Various assignment operators used in Python are (+=, - = , \*=, /= , etc.)

**Example**: Python assignment operators is simply to assign the value, for example

num1 = 4

num2 = 5

print(("Line 1 - Value of num1 : ", num1))

print(("Line 2 - Value of num2 : ", num2))

**Example of compound assignment operator**

We can also use a compound assignment operator, where you can add, subtract, multiply right operand to left and assign addition (or any other arithmetic function) to the left operand.

* Step 1: Assign value to num1 and num2
* Step 2: Add value of num1 and num2 (4+5=9)
* Step 3: To this result add num1 to the output of Step 2 ( 9+4)
* Step 4: It will print the final result as 13

num1 = 4

num2 = 5

res = num1 + num2

res += num1

print(("Line 1 - Result of + is ", res))

## Logical Operators

Logical operators in Python are used for conditional statements are true or false. Logical operators in Python are AND, OR and NOT. For logical operators following condition are applied.

* For AND operator – It returns TRUE if both the operands (right side and left side) are true
* For OR operator- It returns TRUE if either of the operand (right side or left side) is true
* For NOT operator- returns TRUE if operand is false

**Example**: Here in example we get true or false based on the value of a and b

a = True

b = False

print(('a and b is',a and b))

print(('a or b is',a or b))

print(('not a is',not a))

## Membership Operators

These operators test for membership in a sequence such as lists, strings or tuples. There are two membership operators that are used in Python. (in, not in). It gives the result based on the variable present in specified sequence or string

**Example**: For example here we check whether the value of x=4 and value of y=8 is available in list or not, by using **in** and **not in**operators.

x = 4

y = 8

list = [1, 2, 3, 4, 5 ];

if ( x in list ):

print("Line 1 - x is available in the given list")

else:

print("Line 1 - x is not available in the given list")

if ( y not in list ):

print("Line 2 - y is not available in the given list")

else:

print("Line 2 - y is available in the given list")

* Declare the value for x and y
* Declare the value of list
* Use the "in" operator in code with if statement to check the value of x existing in the list and print the result accordingly
* Use the "not in" operator in code with if statement to check the value of y exist in the list and print the result accordingly
* Run the code- When the code run it gives the desired output

## Identity Operators

To compare the memory location of two objects, Identity Operators are used. The two identify operators used in Python are (is, is not).

* Operator is: It returns true if two variables point the same object and false otherwise
* Operator is not: It returns false if two variables point the same object and true otherwise

Following operands are in decreasing order of precedence.

Operators in the same box evaluate left to right

|  |  |
| --- | --- |
| **Operators (Decreasing order of precedence)** | **Meaning** |
| \*\* | Exponent |
| \*, /, //, % | Multiplication, Division, Floor division, Modulus |
| +, - | Addition, Subtraction |
| <= < > >= | Comparison operators |
| = %= /= //= -= += \*= \*\*= | Assignment Operators |
| is is not | Identity operators |
| in not in | Membership operators |
| not or and | Logical operators |

**Example**:

x = 20

y = 20

if ( x is y ):

print("x & y SAME identity")

y=30

if ( x is not y ):

print("x & y have DIFFERENT identity")

* Declare the value for variable x and y
* Use the operator "is" in code to check if value of x is same as y
* Next we use the operator "is not" in code if value of x is not same as y
* Run the code- The output of the result is as expected

## Operator precedence

The operator precedence determines which operators need to be evaluated first. To avoid ambiguity in values, precedence operators are necessary. Just like in normal multiplication method, multiplication has a higher precedence than addition. For example in 3+ 4\*5, the answer is 23, to change the order of precedence we use a parentheses (3+4)\*5, now the answer is 35. Precedence operator used in Python are (unary + - ~, \*\*, \* / %, + - , &) etc.

v = 4

w = 5

x = 8

y = 2

z = 0

z = (v+w) \* x / y;

print("Value of (v+w) \* x/ y is ", z)

* Declare the value of variable v,w…z
* Now apply the formula and run the code
* The code will execute and calculate the variable with higher precedence and will give the output

### Python 2 Example

Above examples are Python 3 codes, if you want to use Python 2, please consider following codes

#Arithmetic Operators

x= 4

y= 5

print x + y

#Comparison Operators

x = 4

y = 5

print('x > y is',x>y)

#Assignment Operators

num1 = 4

num2 = 5

print ("Line 1 - Value of num1 : ", num1)

print ("Line 2 - Value of num2 : ", num2)

#compound assignment operator

num1 = 4

num2 = 5

res = num1 + num2

res += num1

print ("Line 1 - Result of + is ", res)

#Logical Operators

a = True

b = False

print('a and b is',a and b)

print('a or b is',a or b)

print('not a is',not a)

#Membership Operators

x = 4

y = 8

list = [1, 2, 3, 4, 5 ];

if ( x in list ):

print "Line 1 - x is available in the given list"

else:

print "Line 1 - x is not available in the given list"

if ( y not in list ):

print "Line 2 - y is not available in the given list"

else:

print "Line 2 - y is available in the given list"

#Identity Operators

x = 20

y = 20

if ( x is y ):

print "x & y SAME identity"

y=30

if ( x is not y ):

print "x & y have DIFFERENT identity"

#Operator precedence

v = 4

w = 5

x = 8

y = 2

z = 0

z = (v+w) \* x / y;

print "Value of (v+w) \* x/ y is ", z

### Summary:

Operators in a programming language are used to perform various operations on values and variables. In Python, you can use operators like

* There are various methods for arithmetic calculation in Python as you can use the eval function, declare variable & calculate, or call functions
* Comparison operators often referred as relational operators are used to compare the values on either side of them and determine the relation between them
* Python assignment operators are simply to assign the value to variable
* Python also allows you to use a compound assignment operator, in a complicated arithmetic calculation, where you can assign the result of one operand to the other
* For AND operator – It returns TRUE if both the operands (right side and left side) are true
* For OR operator- It returns TRUE if either of the operand (right side or left side) is true
* For NOT operator- returns TRUE if operand is false
* There are two membership operators that are used in Python. (in, not in).
* It gives the result based on the variable present in specified sequence or string
* The two identify operators used in Python are (is, is not)
* It returns true if two variables point the same object and false otherwise
* Precedence operator can be useful when you have to set priority for which calculation need to be done first in a complex calculation.

# Python Functions Examples: Call, Indentation, Arguments & Return Values

### What is a Function in Python?

A Functions in Python are used to utilize the code in more than one place in a program, sometimes also called method or procedures. Python provides you many inbuilt functions like print(), but it also gives freedom to create your own functions.

In this tutorial, we will learn

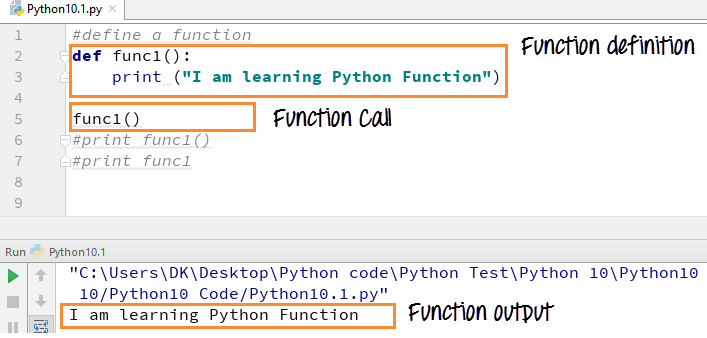
* [How to define and call a function in Python](https://www.guru99.com/functions-in-python.html#1)
* [Significance of Indentation (Space) in Python](https://www.guru99.com/functions-in-python.html#2)
* [How Function Return Value?](https://www.guru99.com/functions-in-python.html#3)
* [Arguments in Functions](https://www.guru99.com/functions-in-python.html#4)

## How to define and call a function in Python

Function in Python is defined by the**"def "**statement followed by the function name and parentheses ( () )

Example:

Let us define a function by using the command " def func1():" and call the function. The output of the function will be **"I am learning Python function".**



The function **print func1()** calls our def func1(): and print the command " **I am learning Python function None.**"

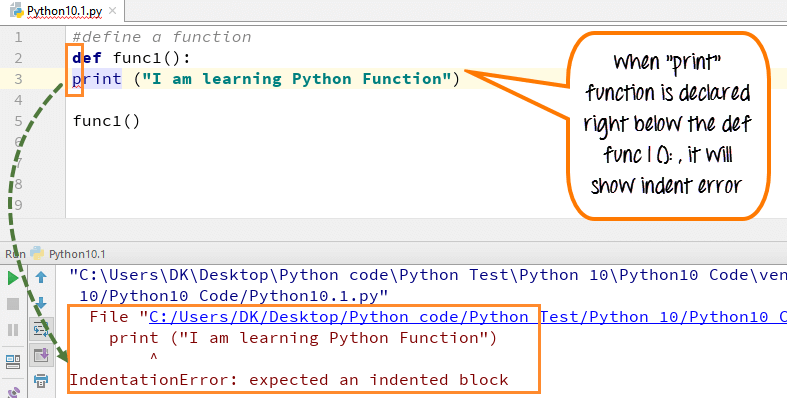
There are set of rules in Python to define a function.

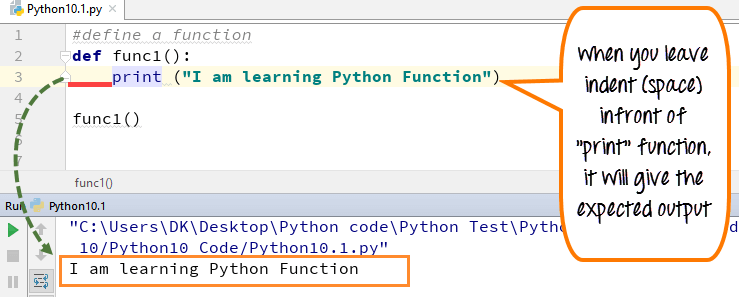
* Any args or input parameters should be placed within these parentheses
* The function first statement can be an optional statement- docstring or the documentation string of the function
* The code within every function starts with a colon (:) and should be indented (space)
* The statement return (expression) exits a function, optionally passing back a value to the caller. A return statement with no args is the same as return None.

## Significance of Indentation (Space) in Python

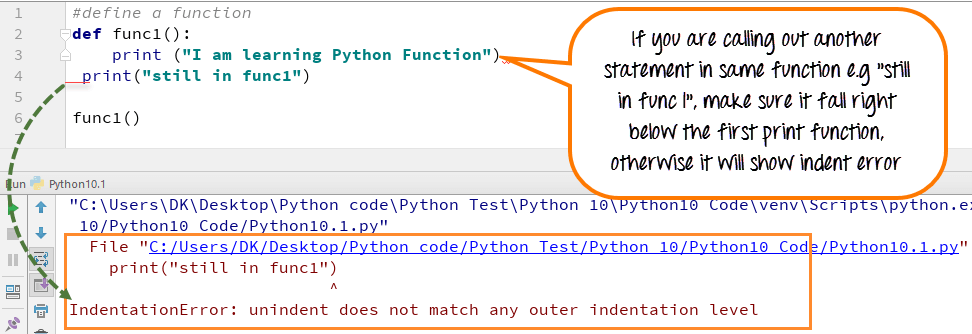
Before we get familiarize with Python functions, it is important that we understand the indentation rule to declare Python functions and these rules are applicable to other elements of Python as well like declaring conditions, loops or variable.

Python follows a particular style of indentation to define the code, since **Python functions don't have any explicit begin or end like curly braces to indicate the start and stop for the function, they have to rely on this indentation**. Here we take a simple example with "print" command. When we write "print" function right below the def func 1 (): It will show an "**indentation error: expected an indented block**".

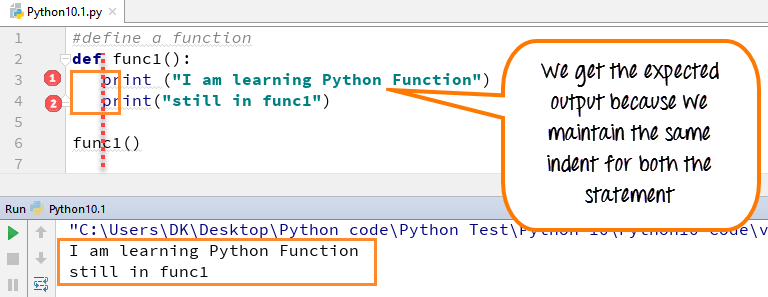
Now, when you add the indent (space) in front of "print" function, it should print as expected.

**At least, one indent is enough to make your code work successfully. But as a best practice it is advisable to leave about 3-4 indent to call your function**.

It is also necessary that while declaring indentation, you have to **maintain the same indent for the rest of your code**. For example, in below screen shot when we call another statement "still in func1" and when it is not declared right below the first print statement it will show an indentation error **"unindent does not match any other indentation level."**



Now, when we apply same indentation for both the statements and align them in the same line, it gives the expected output.

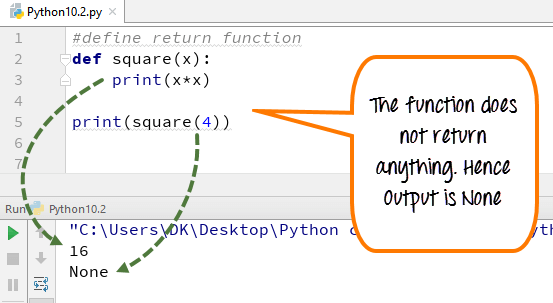


## How Function Return Value?

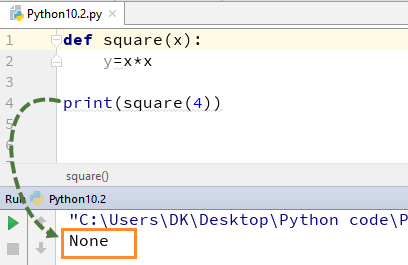
Return command in Python specifies what value to give back to the caller of the function.

Let's understand this with the following example

**Step 1)** Here - we see when function is not "return". For example, we want the square of 4, and it should give answer "16" when the code is executed. Which it gives when we simply use "print x\*x" code, but when you call function "print square" it gives "None" as an output. This is because when you call the function, recursion does not happen and fall off the end of the function. Python returns "None" for failing off the end of the function.

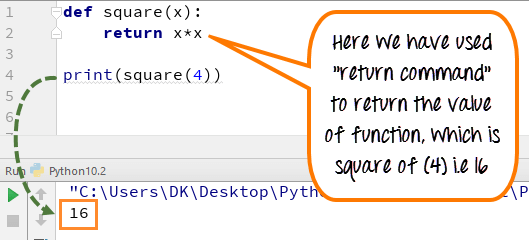


**Step 2)** To make this clearer we replace the print command with assignment command. Let's check the output.

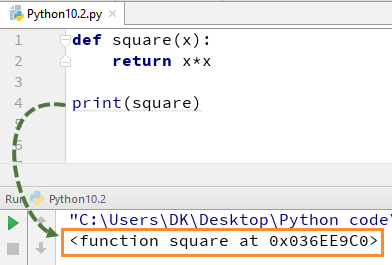


When you run the command "print square (4)" it actually returns the value of the object since we don't have any specific function to run over here it returns "None".

**Step 3)**Now, here we will see how to retrieve the output using "return" command. When you use the "return" function and execute the code, it will give the output "16."



**Step 4)** Functions in Python are themselves an object, and an object has some value. We will here see how Python treats an object. When you run the command "print square" it returns the value of the object. Since we have not passed any argument, we don't have any specific function to run over here it returns a default value (0x021B2D30) which is the location of the object. **In practical Python program, you probably won't ever need to do this.**



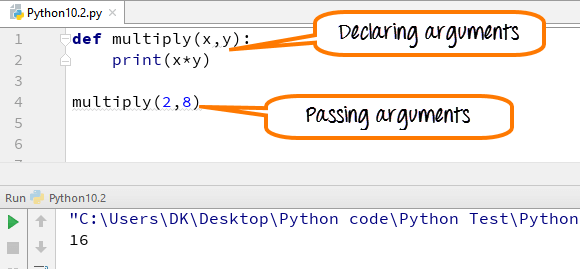
## Arguments in Functions

The argument is a value that is passed to the function when it's called.

In other words on the calling side, it is an argument and on the function side it is a parameter.

Let see how Python Args works -

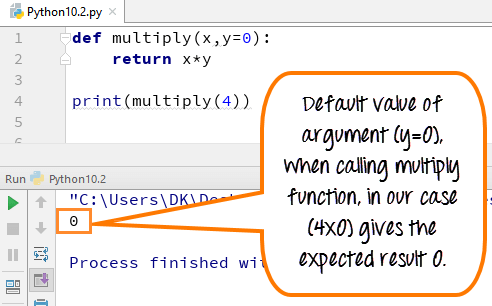
**Step 1)** Arguments are declared in the function definition. While calling the function, you can pass the values for that args as shown below



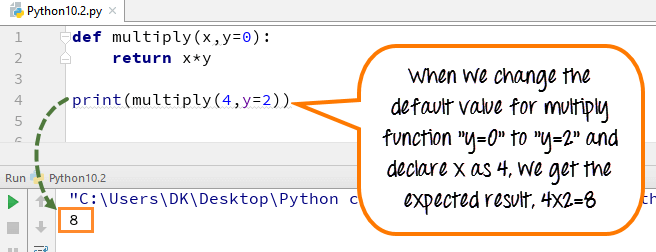
**Step 2)** To declare a default value of an argument, assign it a value at function definition.

[Python Functions Tutorial - Define, Call, Indentation & Arguments](https://www.guru99.com/images/Pythonnew/Python10.11.png)

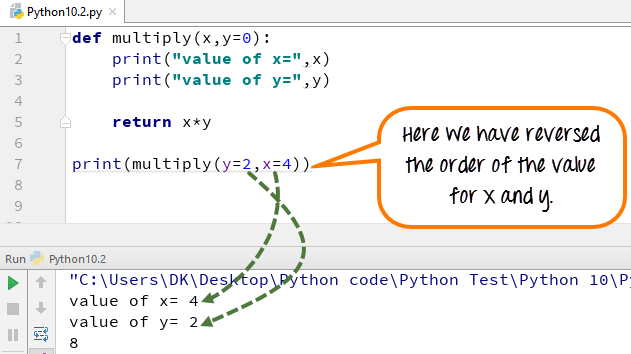
Example: x has no default values. Default values of y=0. When we supply only one argument while calling multiply function, Python assigns the supplied value to x while keeping the value of y=0. Hence the multiply of x\*y=0



**Step 3)**This time we will change the value to y=2 instead of the default value y=0, and it will return the output as (4x2)=8.

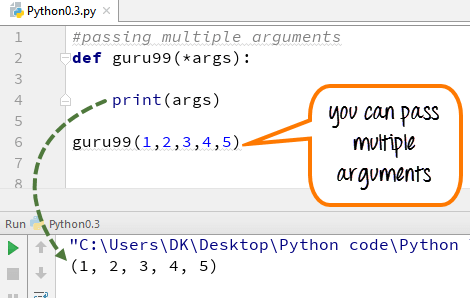


**Step 4)**You can also change the order in which the arguments can be passed in Python. Here we have reversed the order of the value x and y to x=4 and y=2.



**Step 5)** Multiple Arguments can also be passed as an array. Here in the example we call the multiple args (1,2,3,4,5) by calling the (\*args) function.

Example: We declared multiple args as number (1,2,3,4,5) when we call the (\*args) function; it prints out the output as (1,2,3,4,5)



**Tips**:

* In Python 2.7. **function overloading** is not supported in Python. Function Overloading is the ability to create multiple methods of the same name with a different implementation. Function Overloading is fully supported in Python 3
* There is quite a confusion between methods and functions. Methods in Python are associated with object instances while function are not. When Python calls a method, it binds the first parameter of that call to the appropriate object reference. In simple words, a standalone function in Python is a "function", whereas a function that is an attribute of a class or an instance is a "method".

### Here is the complete Python 3 code

#define a function

def func1():

print ("I am learning Python function")

print ("still in func1")

func1()

def square(x):

return x\*x

print(square(4))

def multiply(x,y=0):

print("value of x=",x)

print("value of y=",y)

return x\*y

print(multiply(y=2,x=4))

### Here is the complete Python 2 code

#define a function

def func1():

print " I am learning Python function"

print " still in func1"

func1()

def square(x):

return x\*x

print square(4)

def multiply(x,y=0):

print"value of x=",x

print"value of y=",y

return x\*y

print multiply(y=2,x=4)

### Summary:

Function in Python is a piece of reusable code that is used to perform single, related action. In this article, we will see

* Function defined by the **def** statement
* The code block within every function starts with a colon (:) and should be indented (space)
* Any arguments or input parameters should be placed within these parentheses, etc.
* At least one indent should be left before the code after declaring function
* Same indent style should be maintained throughout the code within def function
* For best practices three or four indents are considered best before the statement
* You can use the "return" command to return values to the function call.
* Python will print a random value like (0x021B2D30) when the argument is not supplied to the calling function. Example "print function."
* On the calling side, it is an argument and on the function side it is a parameter
* Default value in argument - When we supply only one argument while calling multiply function or any other function, Python assigns the other argument by default
* Python enables you to reverse the order of the argument as well

# Python IF, ELSE, ELIF, Nested IF & Switch Case Statement

### What are Conditional Statements?

Conditional Statement in Python perform different computations or actions depending on whether a specific Boolean constraint evaluates to true or false. Conditional statements are handled by IF statements in Python.

In this tutorial, we will see how to apply conditional statements in Python.

* [What is If Statement? How to Use it?](https://www.guru99.com/if-loop-python-conditional-structures.html#1)
* [What happen when "if condition" does not meet](https://www.guru99.com/if-loop-python-conditional-structures.html#2)
* [How to use "else condition"](https://www.guru99.com/if-loop-python-conditional-structures.html#3)
* [When "else condition" does not work](https://www.guru99.com/if-loop-python-conditional-structures.html#4)
* [How to use "elif" condition](https://www.guru99.com/if-loop-python-conditional-structures.html#5)
* [How to execute conditional statement with minimal code](https://www.guru99.com/if-loop-python-conditional-structures.html#6)
* [Nested IF Statement](https://www.guru99.com/if-loop-python-conditional-structures.html#7)
* [Switch Statement](https://www.guru99.com/if-loop-python-conditional-structures.html#8)

## What is If Statement? How to Use it?

In Python, If Statement is used for decision making. It will run the body of code only when IF statement is true.

When you want to justify one condition while the other condition is not true, then you use "if statement".

Syntax:

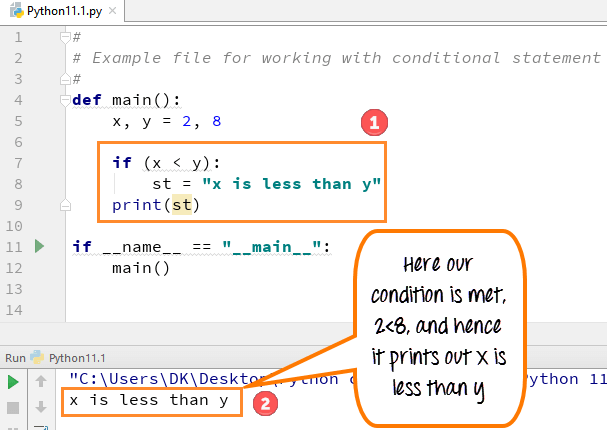
if expression

Statement

else

Statement

Let see an **example**-



#

#Example file for working with conditional statement

#

def main():

x,y =2,8

if(x < y):

st= "x is less than y"

print(st)

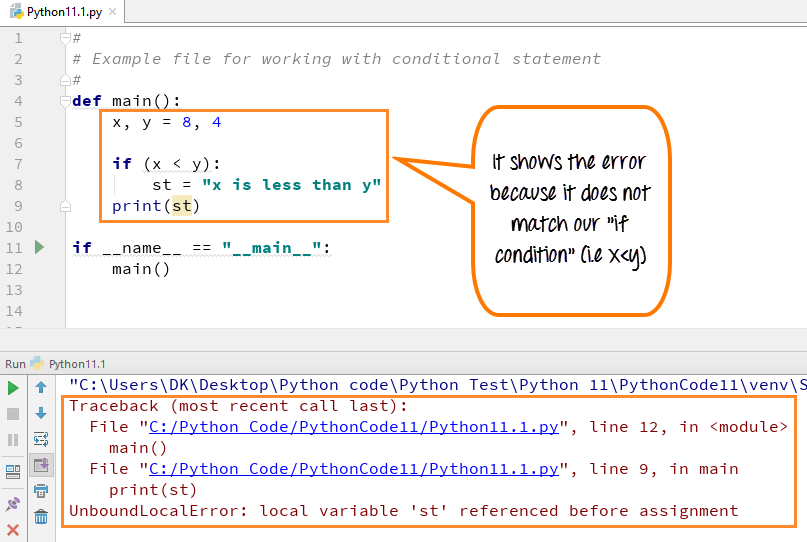
if \_\_name\_\_ == "\_\_main\_\_":

main()

* Code Line 5: We define two variables x, y = 2, 8
* Code Line 7: The if Statement checks for condition x<y which is **True** in this case
* Code Line 8: The variable st is set to "x is less than y."
* Code Line 9: The line print st will output the value of variable st which is "x is less than y",

## What happen when "if condition" does not meet

In this step, we will see what happens when your "if condition" does not meet.

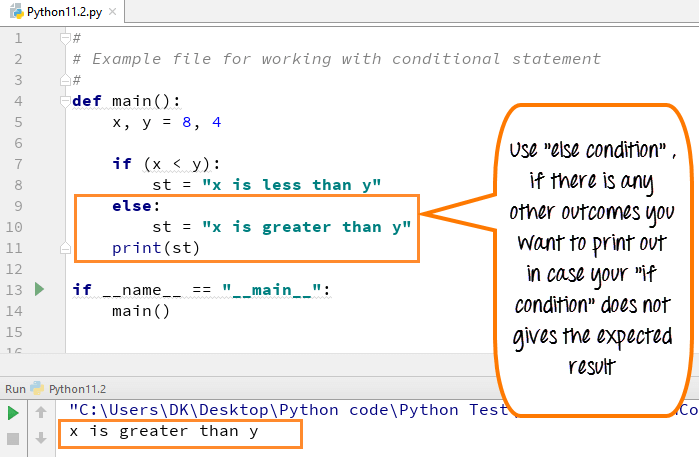


* Code Line 5: We define two variables x, y = 8, 4
* Code Line 7: The if Statement checks for condition x<y which is **False** in this case
* Code Line 8: The variable st is **NOT** set to "x is less than y."
* Code Line 9: The line print st - is trying to print the value of a variable that was never declared. Hence, we get an error.

## How to use "else condition"

The "else condition" is usually used when you have to judge one statement on the basis of other. If one condition goes wrong, then there should be another condition that should justify the statement or logic.

**Example**:



#

#Example file for working with conditional statement

#

def main():

x,y =8,4

if(x < y):

st= "x is less than y"

else:

st= "x is greater than y"

print (st)

if \_\_name\_\_ == "\_\_main\_\_":

main()

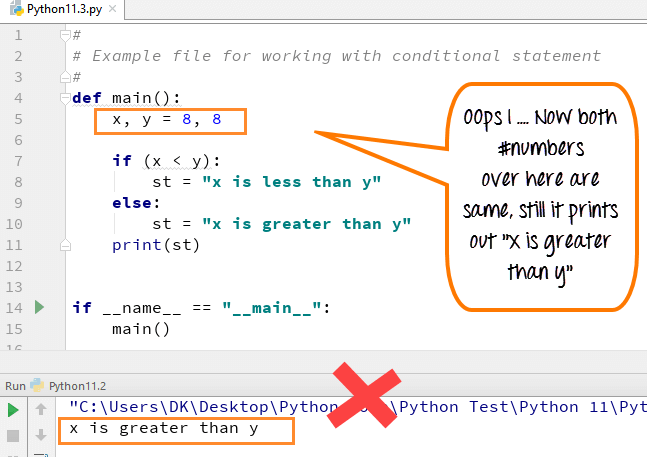
* Code Line 5: We define two variables x, y = 8, 4
* Code Line 7: The if Statement checks for condition x<y which is **False** in this case
* Code Line 9: The flow of program control goes to else condition
* Code Line 10: The variable st is set to "x is **greater** than y."
* Code Line 11: The line print st will output the value of variable st which is "x is greater than y",

## When "else condition" does not work

There might be many instances when your "else condition" won't give you the desired result. It will print out the wrong result as there is a mistake in program logic. In most cases, this happens when you have to justify more than two statement or condition in a program.

An **example** will better help you to understand this concept.

Here both the variables are same (8,8) and the program output is **"x is greater than y",** which is **WRONG**. This is because it checks the first condition (if condition), and if it fails, then it prints out the second condition (else condition) as default. In next step, we will see how we can correct this error.



#

#Example file for working with conditional statement

#

def main():

x,y =8,8

if(x < y):

st= "x is less than y"

else:

st= "x is greater than y"

print(st)

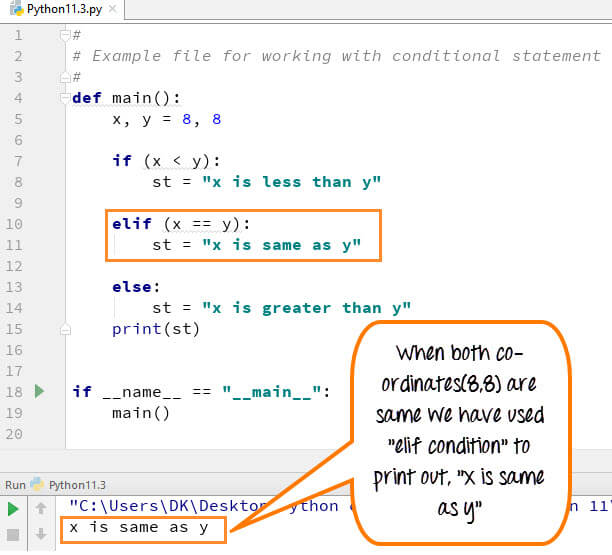
if \_\_name\_\_ == "\_\_main\_\_":

main()

## How to use "elif" condition

To correct the previous error made by "else condition", we can use **"elif"** statement. By using "**elif**" condition, you are telling the program to print out the third condition or possibility when the other condition goes wrong or incorrect.

**Example**



#

#Example file for working with conditional statement

#

def main():

x,y =8,8

if(x < y):

st= "x is less than y"

elif (x == y):

st= "x is same as y"

else:

st="x is greater than y"

print(st)

if \_\_name\_\_ == "\_\_main\_\_":

main()

* Code Line 5: We define two variables x, y = 8, 8
* Code Line 7: The if Statement checks for condition x<y which is **False** in this case
* Code Line 10: The flow of program control goes to the elseif condition. It checks whether x==y which is true
* Code Line 11: The variable st is set to "x is **same as**y."
* Code Line 15: The **flow of program control exits the if Statement (it will not get to the else Statement).** And print the variable st. The output is "x is same as y" which is correct

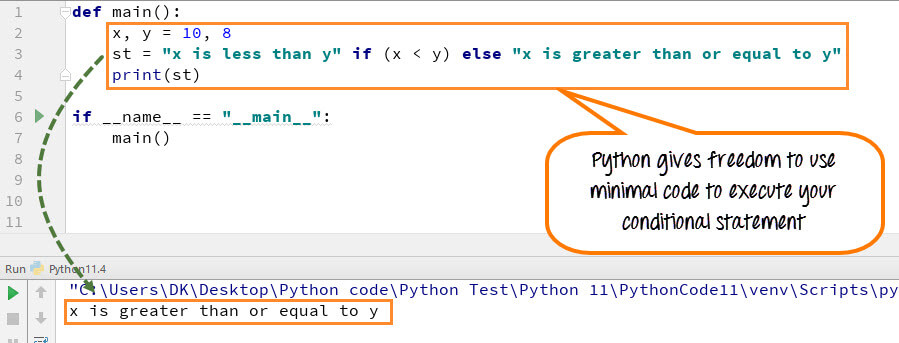
## How to execute conditional statement with minimal code

In this step, we will see how we can condense out the conditional statement. Instead of executing code for each condition separately, we can use them with a single code.

Syntax

A If B else C

**Example**:



def main():

x,y = 10,8

st = "x is less than y" if (x < y) else "x is greater than or equal to y"

print(st)

if \_\_name\_\_ == "\_\_main\_\_":

main()

* Code Line 2: We define two variables x, y = 10, 8
* Code Line 3: Variable st is set to "x is less than y "if x<y or else it is set to "x is greater than or equal to y". In this x>y variable st is set to **"x is greater than or equal to y."**
* Code Line 4: Prints the value of st and gives the correct output
* Instead of writing long code for conditional statements, Python gives you the freedom to write code in a short and concise way.

## Nested IF Statement

Following example demonstrates nested if Statement

total = 100

#country = "US"

country = "AU"

if country == "US":

if total <= 50:

print("Shipping Cost is $50")

elif total <= 100:

print("Shipping Cost is $25")

elif total <= 150:

print("Shipping Costs $5")

else:

print("FREE")

if country == "AU":

if total <= 50:

print("Shipping Cost is $100")

else:

print("FREE")

Uncomment Line 2 in above code and comment Line 3 and run the code again

## Switch Statement

**What is switch statement?**

A switch statement is a multiway branch statement that compares the value of a variable to the values specified in case statements.

Python language doesn’t have a switch statement.

Python uses dictionary mapping to implement switch statement in Python

**Example**

function(argument){

switch(argument) {

case 0:

return "This is Case Zero";

case 1:

return " This is Case One";

case 2:

return " This is Case Two ";

default:

return "nothing";

};

};

For the above switch statement Alternative in Python

def SwitchExample(argument):

switcher = {

0: " This is Case Zero ",

1: " This is Case One ",

2: " This is Case Two ",

}

return switcher.get(argument, "nothing")

if \_\_name\_\_ == "\_\_main\_\_":

argument = 1

print (SwitchExample(argument))

**Python 2 Example**

Above codes are Python 3 examples, If you want to run in Python 2 please consider following code.

# If Statement

#Example file for working with conditional statement

#

def main():

x,y =2,8

if(x < y):

st= "x is less than y"

print st

if \_\_name\_\_ == "\_\_main\_\_":

main()

# How to use "else condition"

#Example file for working with conditional statement

#

def main():

x,y =8,4

if(x < y):

st= "x is less than y"

else:

st= "x is greater than y"

print st

if \_\_name\_\_ == "\_\_main\_\_":

main()

# When "else condition" does not work

#Example file for working with conditional statement

#

def main():

x,y =8,8

if(x < y):

st= "x is less than y"

else:

st= "x is greater than y"

print st

if \_\_name\_\_ == "\_\_main\_\_":

main()

# How to use "elif" condition

#Example file for working with conditional statement

#

def main():

x,y =8,8

if(x < y):

st= "x is less than y"

elif (x == y):

st= "x is same as y"

else:

st="x is greater than y"

print st

if \_\_name\_\_ == "\_\_main\_\_":

main()

# How to execute conditional statement with minimal code

def main():

x,y = 10,8

st = "x is less than y" if (x < y) else "x is greater than or equal to y"

print st

if \_\_name\_\_ == "\_\_main\_\_":

main()

# Nested IF Statement

total = 100

#country = "US"

country = "AU"

if country == "US":

if total <= 50:

print "Shipping Cost is $50"

elif total <= 100:

print "Shipping Cost is $25"

elif total <= 150:

print "Shipping Costs $5"

else:

print "FREE"

if country == "AU":

if total <= 50:

print "Shipping Cost is $100"

else:

print "FREE"

#Switch Statement

def SwitchExample(argument):

switcher = {

0: " This is Case Zero ",

1: " This is Case One ",

2: " This is Case Two ",

}

return switcher.get(argument, "nothing")

if \_\_name\_\_ == "\_\_main\_\_":

argument = 1

print SwitchExample(argument)

### Summary:

A conditional statement in Python is handled by if statements and we saw various other ways we can use conditional statements like if and else over here.

* "if condition" – It is used when you need to print out the result when one of the conditions is true or false.
* "else condition"- it is used when you want to print out the statement when your one condition fails to meet the requirement
* "elif condition" – It is used when you have third possibility as the outcome. You can use multiple elif conditions to check for 4th,5th,6th possibilities in your code
* We can use minimal code to execute conditional statements by declaring all condition in single statement to run the code
* If Statement can be nested

# Python For & While Loops: Enumerate, Break, Continue Statement

### What is Loop?

Loops can execute a block of code number of times until a certain condition is met. Their usage is fairly common in programming. Unlike other programming language that have For Loop, while loop, dowhile, etc.

### What is For Loop?

For loop is used to iterate over elements of a sequence. It is often used when you have a piece of code which you want to repeat "n" number of time.

### What is While Loop?

While Loop is used to repeat a block of code. Instead of running the code block once, It executes the code block multiple times until a certain condition is met.

In this tutorial, we will learn

* [How to use "While Loop"](https://www.guru99.com/python-loops-while-for-break-continue-enumerate.html#1)
* [How to use "For Loop"](https://www.guru99.com/python-loops-while-for-break-continue-enumerate.html#2)
* [How to use For Loop for set of other things besides numbers](https://www.guru99.com/python-loops-while-for-break-continue-enumerate.html#3)
* [Break statements in For Loop](https://www.guru99.com/python-loops-while-for-break-continue-enumerate.html#4)
* [Continue statement in For Loop](https://www.guru99.com/python-loops-while-for-break-continue-enumerate.html#5)
* [Enumerate function for For Loop](https://www.guru99.com/python-loops-while-for-break-continue-enumerate.html#6)
* [Pratical Example](https://www.guru99.com/python-loops-while-for-break-continue-enumerate.html#8)
* [How to use for loop to repeat the same statement over and again](https://www.guru99.com/python-loops-while-for-break-continue-enumerate.html#7)

## How to use "While Loop"

While loop does the exactly same thing what "if statement" does, but instead of running the code block once, they jump back to the point where it began the code and repeats the whole process again.

Syntax

while expression

Statement

**Example**:

#

#Example file for working with loops

#

def main():

x=0

#define a while loop

while(x <4):

print(x)

x = x+1

if \_\_name\_\_ == "\_\_main\_\_":

main()

* Code Line 4: Variable x is set to 0
* Code Line 7: While loop checks for condition x<4. The current value of x is 0. Condition is true. Flow of control enters into while Loop
* Code Line 8: Value of x is printed
* Code Line 9: x is incremented by 1. Flow of control goes back to line 7. Now the value of x is 1 which is less than 4. The condition is true, and again the while loop is executed. This continues till x becomes 4, and the while condition becomes false.

## How to use "For Loop"

In Python, "for loops" are called **iterators.**

Just like while loop, "For Loop" is also used to repeat the program.

But unlike while loop which depends on condition true or false. "For Loop" depends on the elements it has to iterate.

**Example**:

#

#Example file for working with loops

#

def main():

x=0

#define a while loop

# while(x <4):

# print x

# x = x+1

#Define a for loop

for x in range(2,7):

print(x)

if \_\_name\_\_ == "\_\_main\_\_":

main()

For Loop iterates with number declared in the range.

For example,

**For Loop** for x in range (2,7)

When this code is executed, it will print the number between 2 and 7 (2,3,4,5,6). In this code, number 7 is not considered inside the range.

For Loops can also be used for a set of other things and not just number. We will see thin in next section.

## How to use For Loop for String

In this step, we will see how "for loops" can also be used for other things besides numbers.

**Example**:

def main():

#use a for loop over a collection

Months = ["Jan","Feb","Mar","April","May","June"]

for m in Months:

print(m)

if \_\_name\_\_ == "\_\_main\_\_":

main()

Code Line 3: We store the months ("Jan, Feb , Mar,April,May,June") in variable Months

Code Line 4: We iterate the for loop over each value in Months. The current value of Months in stored in variable m

## How to use break statements in For Loop

Breakpoint is a unique function in For Loop that allows you to break or terminate the execution of the for loop

**Example**:

def main():

#use a for loop over a collection

#Months = ["Jan","Feb","Mar","April","May","June"]

#for m in Months:

#print m

# use the break and continue statements

for x in range (10,20):

if (x == 15): break

#if (x % 2 == 0) : continue

print(x)

if \_\_name\_\_ == "\_\_main\_\_":

main()

In this example, we declared the numbers from 10-20, but we want that our for loop to terminate at number 15 and stop executing further. For that, we declare break function by defining (x==15): break, so as soon as the code calls the number 15 it terminates the program Code Line 10 declare variable x between range (10, 20)

* Code Line 11 declare the condition for breakpoint at x==15,
* Code Line 12 checks and repeats the steps until it reaches number 15
* Code Line 13 Print the result in output

## How to use "continue statement" in For Loop

Continue function, as the name indicates, will terminate the current iteration of the for loop BUT will continue execution of the remaining iterations.

**Example**

def main():

#use a for loop over a collection

#Months = ["Jan","Feb","Mar","April","May","June"]

#for m in Months:

#print m

# use the break and continue statements

for x in range (10,20):

#if (x == 15): break

if (x % 5 == 0) : continue

print(x)

if \_\_name\_\_ == "\_\_main\_\_":

main()

Continue statement can be used in for loop when you want to fetch a specific value from the list.

In our example, we have declared value 10-20, but between these numbers we only want those number that are NOT divisible by 5 or in other words which don't give zero when divided by 5.

So, in our range (10,11, 12….19,20) only 3 numbers falls (10,15,20) that are divisible by 5 and rest are not.

So except number 10,15 & 20 the "for loop" will not continue and print out those number as output.

* Code line 10 declare the variable x for range (10, 20)
* Code line 12 declare the condition for x divided by 5=0 continue
* Code line 13 print the result

## How to use "enumerate" function for "For Loop"

Enumerate function in "for loop" does two things

* It returns the index number for the member
* And the member of the collection that we are looking at

**Example**:

Enumerate function is used for the numbering or indexing the members in the list.

Suppose, we want to do numbering for our month ( Jan, Feb, Marc, ….June), so we declare the variable i that enumerate the numbers while m will print the number of month in list.

def main():

#use a for loop over a collection

Months = ["Jan","Feb","Mar","April","May","June"]

for i, m in enumerate (Months):

print(i,m)

# use the break and continue statements

#for x in range (10,20):

#if (x == 15): break

#if (x % 5 == 0) : continue

#print x

if \_\_name\_\_ == "\_\_main\_\_":

main()

When code is executed the output of the enumerate function returns the months name with an index number like (0-Jan), (1- Feb), (2- March), etc.

* Code Line 3 declares the list of months [ Jan, Feb,…Jun]
* Code Line 4 declares variable i and m for For Loop
* Code Line 5 will print the result and again enter the For Loop for the rest of the months to enumerate

## Pratical Example

Let see another example for For Loop to repeat the same statement over and again.

|  |  |
| --- | --- |
| **Python loop** | **Working Code for all exercises** |
| Code for while loop | def main():  x=0  while (x<4):    print x  x= x+1  if \_\_name\_\_== "\_\_main\_\_":  main() |
| For Loop Simple Example | def main():  x=0    for x in range (2,7):    print x    if \_\_name\_\_== "\_\_main\_\_":  main() |
| Use of for loop in string | def main():    Months = ["Jan","Feb","Mar","April","May","June"]  for m in (Months):  print m    if \_\_name\_\_== "\_\_main\_\_":  main() |
| Use break-statement in for loop | def main():    for x in range (10,20):  if (x == 15): break  print x    if \_\_name\_\_== "\_\_main\_\_":  main() |
| Use of Continue statement in for loop | def main():    for x in range (10,20):  if (x % 5 == 0): continue  print x    if \_\_name\_\_== "\_\_main\_\_":  main() |
| Code for "enumerate function" with "for loop" | def main():    Months = ["Jan","Feb","Mar","April","May","June"]  for i, m in enumerate (Months):  print i,m    if \_\_name\_\_== "\_\_main\_\_":  main() |

## How to use for loop to repeat the same statement over and again

You can use for loop for even repeating the same statement over and again. Here in the example we have printed out word "guru99" three times.

Example: To repeat same statement number of times, we have declared the number in variable i (i in 123). So when you run the code as shown below, it prints the statement (guru99) that many times the number declared for our the variable in ( i in 123).

for i in '123':

print "guru99",i,

Like other programming languages, Python also uses a loop but instead of using a range of different loops it is restricted to only two loops "While loop" and "for loop".

* While loops are executed based on whether the conditional statement is true or false.
* For loops are called iterators, it iterates the element based on the condition set
* Python For loops can also be used for a set of various other things (specifying the collection of elements we want to loop over)
* Breakpoint is used in For Loop to break or terminate the program at any particular point
* Continue statement will continue to print out the statement, and prints out the result as per the condition set
* Enumerate function in "for loop" returns the member of the collection that we are looking at with the index number

**Python 2 Example**

Above codes are Python 3 examples, If you want to run in Python 2 please consider following code.

# How to use "While Loop"

#Example file for working with loops

#

def main():

x=0

#define a while loop

while(x <4):

print x

x = x+1

if \_\_name\_\_ == "\_\_main\_\_":

main()

#How to use "For Loop"

#Example file for working with loops

#

def main():

x=0

#define a while loop

# while(x <4):

# print x

# x = x+1

#Define a for loop

for x in range(2,7):

print x

if \_\_name\_\_ == "\_\_main\_\_":

main()

#How to use For Loop for String

def main():

#use a for loop over a collection

Months = ["Jan","Feb","Mar","April","May","June"]

for m in Months:

print m

if \_\_name\_\_ == "\_\_main\_\_":

main()

#How to use break statements in For Loop

def main():

#use a for loop over a collection

#Months = ["Jan","Feb","Mar","April","May","June"]

#for m in Months:

#print m

# use the break and continue statements

for x in range (10,20):

if (x == 15): break

#if (x % 2 == 0) : continue

print x

if \_\_name\_\_ == "\_\_main\_\_":

main()

#How to use "continue statement" in For Loop

def main():

#use a for loop over a collection

#Months = ["Jan","Feb","Mar","April","May","June"]

#for m in Months:

#print m

# use the break and continue statements

for x in range (10,20):

#if (x == 15): break

if (x % 5 == 0) : continue

print x

if \_\_name\_\_ == "\_\_main\_\_":

main()

#How to use "enumerate" function for "For Loop"

def main():

#use a for loop over a collection

Months = ["Jan","Feb","Mar","April","May","June"]

for i, m in enumerate (Months):

print i,m

# use the break and continue statements

#for x in range (10,20):

#if (x == 15): break

#if (x % 5 == 0) : continue

#print x

if \_\_name\_\_ == "\_\_main\_\_":

main()

# Python OOPs: Class, Object, Inheritance and Constructor with Example

### What is a Class?

A Class is a logical grouping of data and functions. It gives the freedom to create data structures that contains arbitrary content and hence easily accessible.

For example, for any bank employee who want to fetch the customer details online would go to **customer class**, where all its attributes like transaction details, withdrawal and deposit details, outstanding debt, etc. would be listed out.

In this tutorial, we will learn,

* [How to define Python classes](https://www.guru99.com/python-class-objects-object-oriented-programming-oop-s.html#2)
* [How Inheritance works](https://www.guru99.com/python-class-objects-object-oriented-programming-oop-s.html#3)
* [Python Constructors](https://www.guru99.com/python-class-objects-object-oriented-programming-oop-s.html#4)

## How to define Python classes

To define class you need to consider following points

**Step 1)** In Python, classes are defined by the **"Class"**keyword

class myClass():

**Step 2)** Inside classes, you can define functions or methods that are part of this class

def method1 (self):

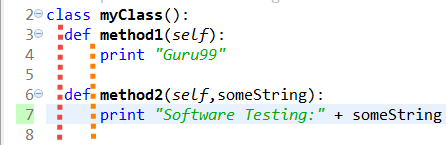
print "Guru99"

def method2 (self,someString):

print "Software Testing:" + someString

* Here we have defined method1 that prints "Guru99."
* Another method we have defined is method2 that prints "Software Testing"+ SomeString. SomeString is the variable supplied by the calling method

**Step 3)** Everything in a class is indented, just like the code in the function, loop, if statement, etc. Anything not indented is not in the class

[](https://www.guru99.com/images/7-2015/071515_1030_PythonClass1.png)

**NOTE**: About using "self" in Python

* The self-argument refers to the object itself. Hence the use of the word self. So inside this method, self will refer to the specific instance of this object that's being operated on.
* Self is the name preferred by convention by Pythons to indicate the first parameter of instance methods in Python. It is part of the Python syntax to access members of objects

**Step 4)** To make an object of the class

c = myClass()

**Step 5)** To call a method in a class

c.method1()

c.method2(" Testing is fun")

* Notice that when we call the method1 or method2, we don't have to supply the self-keyword. That's automatically handled for us by the Python runtime.
* Python runtime will pass "self" value when you call an instance method on in instance, whether you provide it deliberately or not
* You just have to care about the non-self arguments

**Step 6)**Here is the complete code

# Example file for working with classes

class myClass():

def method1(self):

print("Guru99")

def method2(self,someString):

print("Software Testing:" + someString)

def main():

# exercise the class methods

c = myClass ()

c.method1()

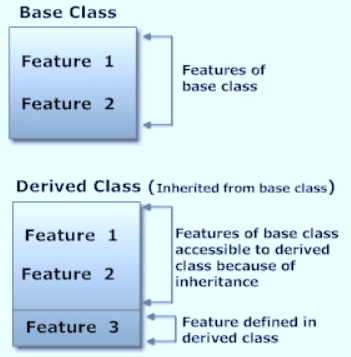
c.method2(" Testing is fun")

if \_\_name\_\_== "\_\_main\_\_":

main()

## How Inheritance works

Inheritance is a feature used in object-oriented programming; it refers to defining a new class with less or no modification to an existing class. The new class is called **derived class** and from one which it inherits is called the **base**. Python supports inheritance; it also supports **multiple inheritances**. A class can inherit attributes and behavior methods from another class called subclass or heir class.



**Python Inheritance Syntax**

class DerivedClass(BaseClass):

body\_of\_derived\_class

**Step 1)** Run the following code

# Example file for working with classes

class myClass():

def method1(self):

print("Guru99")

class childClass(myClass):

#def method1(self):

#myClass.method1(self);

#print ("childClass Method1")

def method2(self):

print("childClass method2")

def main():

# exercise the class methods

c2 = childClass()

c2.method1()

#c2.method2()

if \_\_name\_\_== "\_\_main\_\_":

main()

Notice that the in childClass, method1 is not defined but it is derived from the parent myClass. The output is "Guru99."

**Step 2)** Uncomment Line # 8 & 10. Run the code

Now, the method 1 is defined in the childClass and output "childClass Method1" is correctly shown.

**Step 3)** Uncomment Line #9. Run the code

You can a method of the parent class using the syntax

ParentClassName.MethodName(self)

In our case, we call, myClass.method1(self) and Guru99 is printed as expected

**Step 4**) Uncomment Line #19. Run the code.

Method 2 of the child class is called and "childClass method2" is printed as expected.

## Python Constructors

A constructor is a class function that instantiates an object to predefined values.

It begins with a double underscore (\_). It \_\_init\_\_() method

In below example we are taking name of the user using constructor.

class User:

name = ""

def \_\_init\_\_(self, name):

self.name = name

def sayHello(self):

print("Welcome to Guru99, " + self.name)

User1 = User("Alex")

User1.sayHello()

Output will be:

Welcome to Guru99, Alex

**Python 2 Example**

Above codes are Python 3 examples, If you want to run in Python 2 please consider following code.

# How to define Python classes

# Example file for working with classes

class myClass():

def method1(self):

print "Guru99"

def method2(self,someString):

print "Software Testing:" + someString

def main():

# exercise the class methods

c = myClass ()

c.method1()

c.method2(" Testing is fun")

if \_\_name\_\_== "\_\_main\_\_":

main()

#How Inheritance works

# Example file for working with classes

class myClass():

def method1(self):

print "Guru99"

class childClass(myClass):

#def method1(self):

#myClass.method1(self);

#print "childClass Method1"

def method2(self):

print "childClass method2"

def main():

# exercise the class methods

c2 = childClass()

c2.method1()

#c2.method2()

if \_\_name\_\_== "\_\_main\_\_":

main()

### Summary:

"Class" is a logical grouping of functions and data. Python class provides all the standard features of Object Oriented Programming.

* Class inheritance mechanism
* A derived class that override any method of its base class
* A method can call the method of a base class with the same name
* Python Classes are defined by keyword **"class"** itself
* Inside classes, you can define functions or methods that are part of the class
* Everything in a class is indented, just like the code in the function, loop, if statement, etc.
* The self argument in Python refers to the object itself. Self is the name preferred by convention by Pythons to indicate the first parameter of instance methods in Python
* Python runtime will pass "self" value automatically when you call an instance method on in instance, whether you provide it deliberately or not
* In Python, a class can inherit attributes and behavior methods from another class called subclass or heir class.

**Python Regex: re.match(), re.search(), re.findall() with Example**

### hat is Regular Expression?

A regular expression in a programming language is a special text string used for describing a search pattern. It is extremely useful for extracting information from text such as code, files, log, spreadsheets or even documents.

While using the regular expression the first thing is to recognize is that everything is essentially a character, and we are writing patterns to match a specific sequence of characters also referred as string. Ascii or latin letters are those that are on your keyboards and Unicode is used to match the foreign text. It includes digits and punctuation and all special characters like $#@!%, etc.

In this tutorial, we will learn-

* [Regular Expression Syntax](https://www.guru99.com/python-regular-expressions-complete-tutorial.html#1)
* [Example of w+ and ^ Expression](https://www.guru99.com/python-regular-expressions-complete-tutorial.html#2)
* [Example of \s expression in re.split function](https://www.guru99.com/python-regular-expressions-complete-tutorial.html#3)
* [Using regular expression methods](https://www.guru99.com/python-regular-expressions-complete-tutorial.html#4)
* [Using re.match()](https://www.guru99.com/python-regular-expressions-complete-tutorial.html#5)
* [Finding Pattern in Text (re.search())](https://www.guru99.com/python-regular-expressions-complete-tutorial.html#6)
* [Using re.findall for text](https://www.guru99.com/python-regular-expressions-complete-tutorial.html#7)
* [Python Flags](https://www.guru99.com/python-regular-expressions-complete-tutorial.html#8)
* [Example of re.M or Multiline Flags](https://www.guru99.com/python-regular-expressions-complete-tutorial.html#9)

For instance, a regular expression could tell a program to search for specific text from the string and then to print out the result accordingly. Expression can include

* Text matching
* Repetition
* Branching
* Pattern-composition etc.

In Python, a regular expression is denoted as RE (REs, regexes or regex pattern) are imported through **re module**. Python supports regular expression through libraries. In Python regular expression supports various things like **Modifiers, Identifiers, and White space characters**.

|  |  |  |  |
| --- | --- | --- | --- |
| **Identifiers** | **Modifiers** | **White space characters** | **Escape required** |
| \d= any number (a digit) | \d represents a digit.Ex: \d{1,5} it will declare digit between 1,5 like 424,444,545 etc. | \n = new line | . + \* ? [] $ ^ () {} | \ |
| \D= anything but a number (a non-digit) | + = matches 1 or more | \s= space |  |
| \s = space (tab,space,newline etc.) | ? = matches 0 or 1 | \t =tab |  |
| \S= anything but a space | \* = 0 or more | \e = escape |  |
| \w = letters ( Match alphanumeric character, including "\_") | $ match end of a string | \r = carriage return |  |
| \W =anything but letters ( Matches a non-alphanumeric character excluding "\_") | ^ match start of a string | \f= form feed |  |
| . = anything but letters (periods) | | matches either or x/y | ----------------- |  |
| \b = any character except for new line | [] = range or "variance" | ---------------- |  |
| \. | {x} = this amount of preceding code | ----------------- |  |

## Regular Expression Syntax

**RE**

import re

* "re" module included with Python primarily used for string searching and manipulation
* Also used frequently for web page "Scraping" (extract large amount of data from websites)

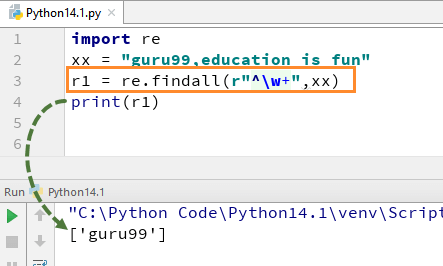
We will begin the expression tutorial with this simple exercise by using the expressions (w+) and (^).

## Example of w+ and ^ Expression

* **"^":**This expression matches the start of a string
* **"w+**": This expression matches the alphanumeric character in the string

Here we will see an example of how we can use w+ and ^ expression in our code. We cover re.findall function later in this tutorial but for a while we simply focus on \w+ and \^ expression.

For example, for our string "guru99, education is fun" if we execute the code with w+ and^, it will give the output "guru99".



import re

xx = "guru99,education is fun"

r1 = re.findall(r"^\w+",xx)

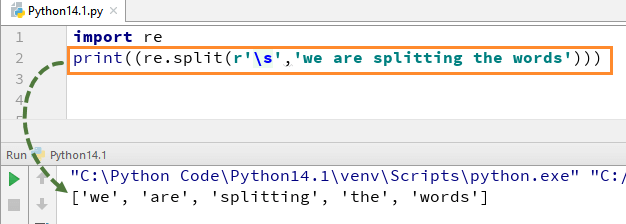
print(r1)

Remember, if you remove +sign from the w+, the output will change, and it will only give the first character of the first letter, i.e., [g]

## Example of \s expression in re.split function

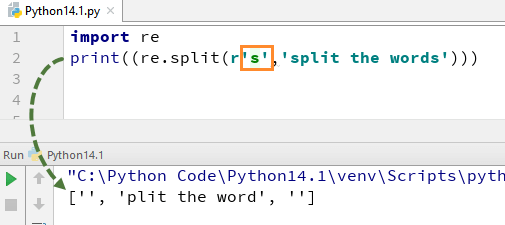
* "s": This expression is used for creating a space in the string

To understand how this regular expression works in Python, we begin with a simple example of a split function. In the example, we have split each word using the "re.split" function and at the same time we have used expression \s that allows to parse each word in the string separately.



When you execute this code it will give you the output ['we', 'are', 'splitting', 'the', 'words'].

Now, let see what happens if you remove "\" from s. There is no 's' alphabet in the output, this is because we have removed '\' from the string, and it evaluates "s" as a regular character and thus split the words wherever it finds "s" in the string.



Similarly, there are series of other regular expressions in Python that you can use in various ways in Python like \d,\D,$,\.,\b, etc.

Here is the complete code

import re

xx = "guru99,education is fun"

r1 = re.findall(r"^\w+", xx)

print((re.split(r'\s','we are splitting the words')))

print((re.split(r's','split the words')))

Next, we will going to see the types of methods that are used with regular expressions.

## Using regular expression methods

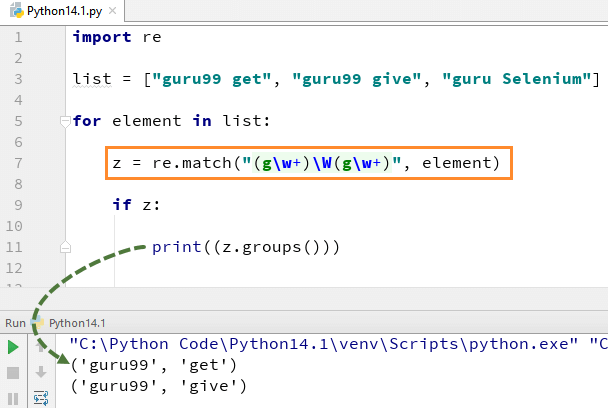
The "re" package provides several methods to actually perform queries on an input string. The method we going to see are

* re.match()
* re.search()
* re.findall()

**Note**: Based on the regular expressions, Python offers two different primitive operations. The match method checks for a match only at the beginning of the string while search checks for a match anywhere in the string.

## Using re.match()

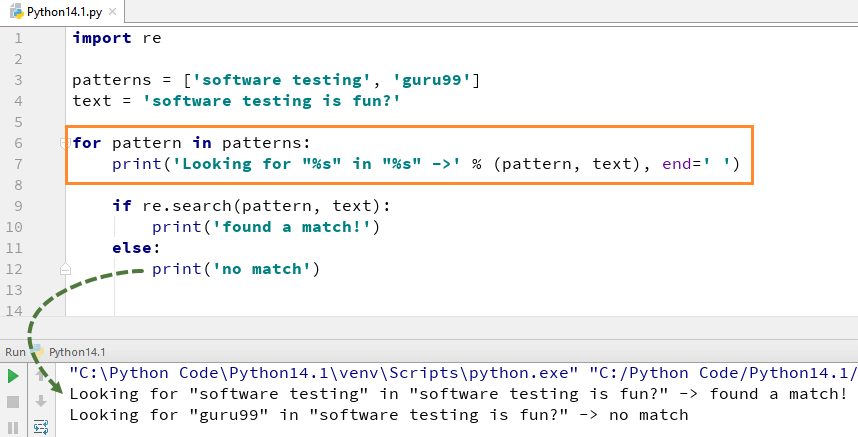
The match function is used to match the RE pattern to string with optional flags. In this method, the expression "w+" and "\W" will match the words starting with letter 'g' and thereafter, anything which is not started with 'g' is not identified. To check match for each element in the list or string, we run the forloop.

[](https://www.guru99.com/images/Pythonnew/Python14.4.png)

## Finding Pattern in Text (re.search())

A regular expression is commonly used to search for a pattern in a text. This method takes a regular expression pattern and a string and searches for that pattern with the string.

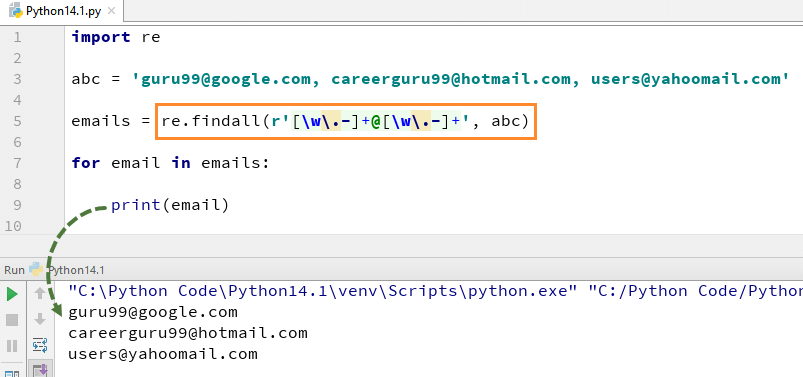
In order to use search() function, you need to import re first and then execute the code. The search() function takes the "pattern" and "text" to scan from our main string and returns a match object when the pattern is found or else not match if the pattern is not found.



For example here we look for two literal strings "Software testing" "guru99", in a text string "Software[Testing](https://www.guru99.com/software-testing.html)is fun". For "software testing" we found the match hence it returns the output as "found a match", while for word "guru99" we could not found in string hence it returns the output as "No match".

## Using re.findall for text

Re.findall() module is used when you want to iterate over the lines of the file, it will return a list of all the matches in a single step. For example, here we have a list of e-mail addresses, and we want all the e-mail addresses to be fetched out from the list, we use the re.findall method. It will find all the e-mail addresses from the list.



Here is the complete code

import re

list = ["guru99 get", "guru99 give", "guru Selenium"]

for element in list:

z = re.match("(g\w+)\W(g\w+)", element)

if z:

print((z.groups()))

patterns = ['software testing', 'guru99']

text = 'software testing is fun?'

for pattern in patterns:

print('Looking for "%s" in "%s" ->' % (pattern, text), end=' ')

if re.search(pattern, text):

print('found a match!')

else:

print('no match')

abc = ['guru99@google.com](mailto:'guru99@google.com)

, [careerguru99@hotmail.com](mailto:careerguru99@hotmail.com), [users@yahoomail.com](mailto:users@yahoomail.com)'

emails = re.findall(r'[\w\.-]+@[\w\.-]+', abc)

for email in emails:

print(email)

## Python Flags

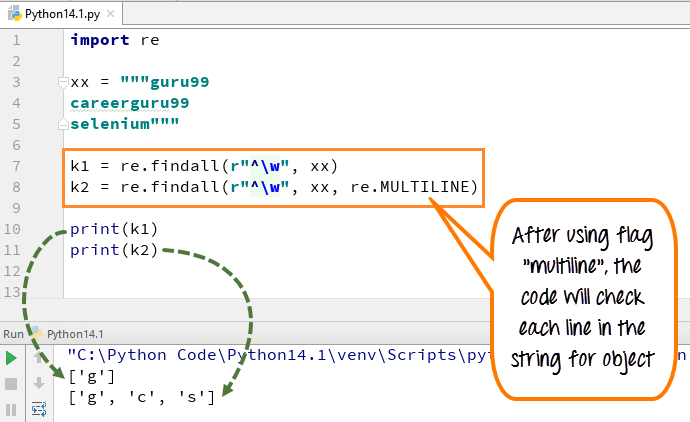
Many Python Regex Methods and Regex functions take an optional argument called Flags. This flags can modify the meaning of the given Regex pattern. To understand these we will see one or two example of these Flags.

Various flags used in Python includes

|  |  |
| --- | --- |
| **Syntax for Regex Flags** | **What does this flag do** |
| [re.M] | Make begin/end consider each line |
| [re.I] | It ignores case |
| [re.S] | Make [ . ] |
| [re.U] | Make { \w,\W,\b,\B} follows Unicode rules |
| [re.L] | Make {\w,\W,\b,\B} follow locale |
| [re.X] | Allow comment in Regex |

## Example of re.M or Multiline Flags

In multiline the pattern character [^] match the first character of the string and the beginning of each line (following immediately after the each newline). While expression small "w" is used to mark the space with characters. When you run the code the first variable "k1" only prints out the character 'g' for word guru99, while when you add multiline flag, it fetches out first characters of all the elements in the string.



Here is the code

import re

xx = """guru99

careerguru99

selenium"""

k1 = re.findall(r"^\w", xx)

k2 = re.findall(r"^\w", xx, re.MULTILINE)

print(k1)

print(k2)

* We declared the variable xx for string " guru99…. careerguru99….selenium"
* Run the code without using flags multiline, it gives the output only 'g' from the lines
* Run the code with flag "multiline", when you print 'k2' it gives the output as 'g', 'c' and 's'
* So, the difference we can see after and before adding multi-lines in above example.

Likewise, you can also use other Python flags like re.U (Unicode), re.L (Follow locale), re.X (Allow Comment), etc.

**Python 2 Example**

Above codes are Python 3 examples, If you want to run in Python 2 please consider following code.

# Example of w+ and ^ Expression

import re

xx = "guru99,education is fun"

r1 = re.findall(r"^\w+",xx)

print r1

# Example of \s expression in re.split function

import re

xx = "guru99,education is fun"

r1 = re.findall(r"^\w+", xx)

print (re.split(r'\s','we are splitting the words'))

print (re.split(r's','split the words'))

# Using re.findall for text

import re

list = ["guru99 get", "guru99 give", "guru Selenium"]

for element in list:

z = re.match("(g\w+)\W(g\w+)", element)

if z:

print(z.groups())

patterns = ['software testing', 'guru99']

text = 'software testing is fun?'

for pattern in patterns:

print 'Looking for "%s" in "%s" ->' % (pattern, text),

if re.search(pattern, text):

print 'found a match!'

else:

print 'no match'

abc = ['guru99@google.com](mailto:'guru99@google.com), [careerguru99@hotmail.com](mailto:careerguru99@hotmail.com), [users@yahoomail.com](mailto:users@yahoomail.com)'

emails = re.findall(r'[\w\.-]+@[\w\.-]+', abc)

for email in emails:

print email

# Example of re.M or Multiline Flags

import re

xx = """guru99

careerguru99

selenium"""

k1 = re.findall(r"^\w", xx)

k2 = re.findall(r"^\w", xx, re.MULTILINE)

print k1

print k2

### Summary

A regular expression in a programming language is a special text string used for describing a search pattern. It includes digits and punctuation and all special characters like $#@!%, etc. Expression can include literal

* Text matching
* Repetition
* Branching
* Pattern-composition etc.

In Python, a regular expression is denoted as RE (REs, regexes or regex pattern) are embedded through re module.

* "re" module included with Python primarily used for string searching and manipulation
* Also used frequently for webpage "Scraping" (extract large amount of data from websites)
* **Regular Expression Methods include** re.match(),re.search()& re.findall()
* **Python Flags** Many Python Regex Methods and Regex functions take an optional argument called Flags
* This flags can modify the meaning of the given Regex pattern
* Various Python flags used in Regex Methods are re.M, re.I, re.S, etc.

**Python DateTime, TimeDelta, Strftime(Format) with Examples**

In Python, **date, time and datetime** classes provides a number of function to deal with dates, times and time intervals. Date and datetime are an object in Python, so when you manipulate them, you are actually manipulating objects and not string or timestamps. Whenever you manipulate dates or time, you need to import datetime function.

The datetime classes in Python are categorized into main 5 classes.

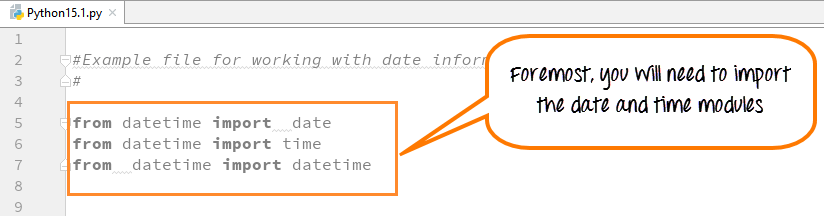
* date – Manipulate just date ( Month, day, year)
* time – Time independent of the day (Hour, minute, second, microsecond)
* datetime – Combination of time and date (Month, day, year, hour, second, microsecond)
* timedelta— A duration of time used for manipulating dates
* tzinfo— An abstract class for dealing with time zones

In this tutorial, we will learn-

* [How to Use Date & DateTime Class](https://www.guru99.com/date-time-and-datetime-classes-in-python.html#1)
* [Print Date using date.today()](https://www.guru99.com/date-time-and-datetime-classes-in-python.html#2)
* [Python Current Date and Time: now() today()](https://www.guru99.com/date-time-and-datetime-classes-in-python.html#3)
* [How to Format Date and Time Output with Strftime()](https://www.guru99.com/date-time-and-datetime-classes-in-python.html#4)
* [How to use Timedelta Objects](https://www.guru99.com/date-time-and-datetime-classes-in-python.html#5)

## How to Use Date & DateTime Class

**Step 1**) Before you run the code for datetime, it is important that you import the date time modules as shown in the screenshot below.



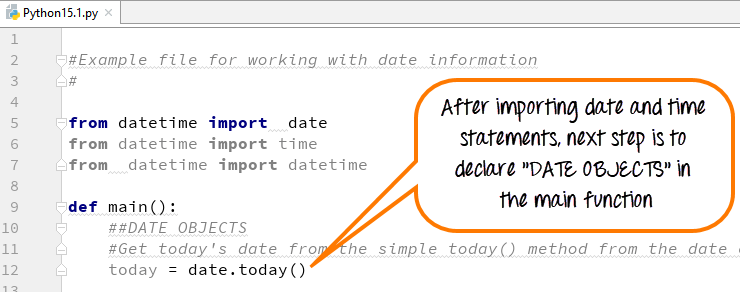
These import statements are pre-defined pieces of functionality in the Python library that let you manipulates dates and times, without writing any code.

Consider the following points before executing the datetime code

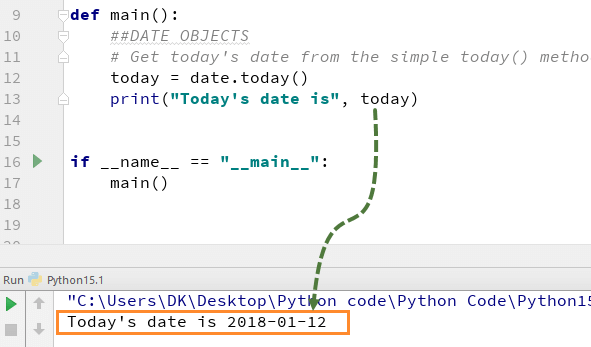
from datetime import date

This line tells the Python interpreter that from the datetime module import the date class We are not writing the code for this date functionality alas just importing it for our use

**Step 2**) Next, we create an instance of the date object.



**Step 3**) Next, we print the date and run the code.

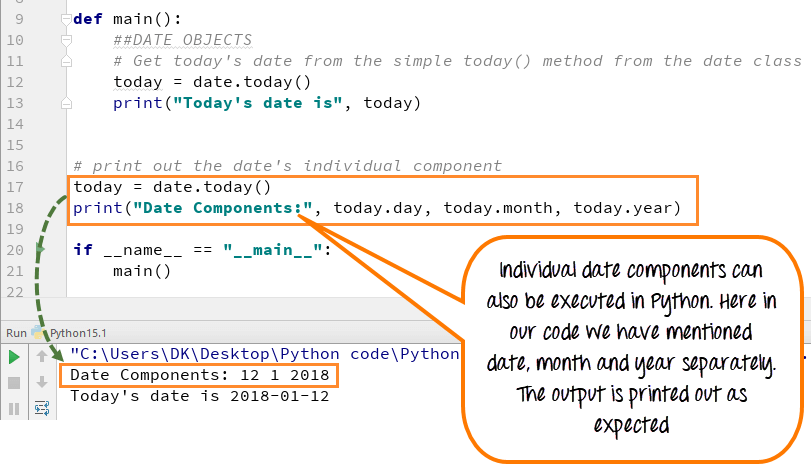


The output is as expected.

## Print Date using date.today()

date.today function has several properties associated with it. We can print individual day/month/year and many other things

Let's see an example

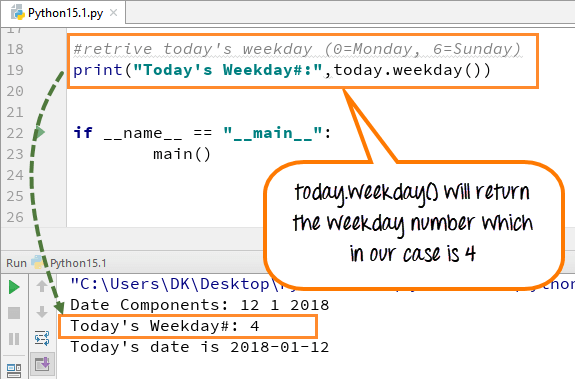


### Today's Weekday Number

The date.today() function also gives you the weekday number. Here is the Weekday Table which start with Monday as 0 and Sunday as 6

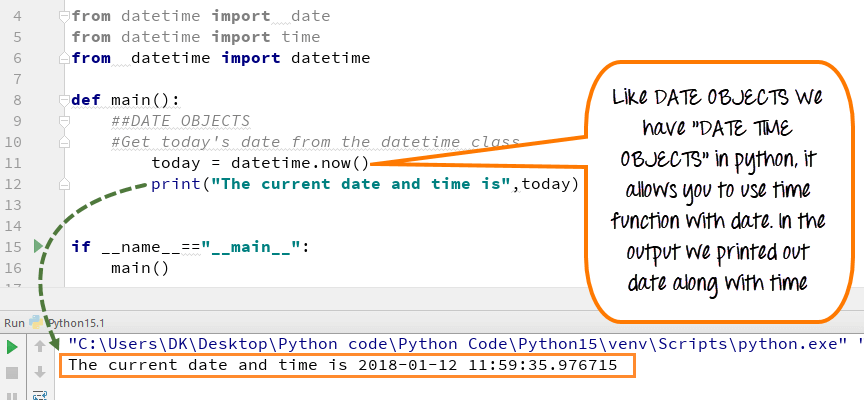
|  |  |
| --- | --- |
| **Day** | **WeekDay Number** |
| **Monday** | 0 |
| **Tuesday** | 1 |
| **Wednesday** | 2 |
| **Thursday** | 3 |
| **Friday** | 4 |
| **Saturday** | 5 |
| **Sunday** | 6 |

Weekday Number is useful for arrays whose index is dependent on the Day of the week.



## Python Current Date and Time: now() today()

**Step 1)** Like Date Objects, we can also use **"DATETIME OBJECTS"** in Python. It gives date along with time in **hours, minutes, seconds and milliseconds.**



When we execute the code for datetime, it gives the output with current date and time.

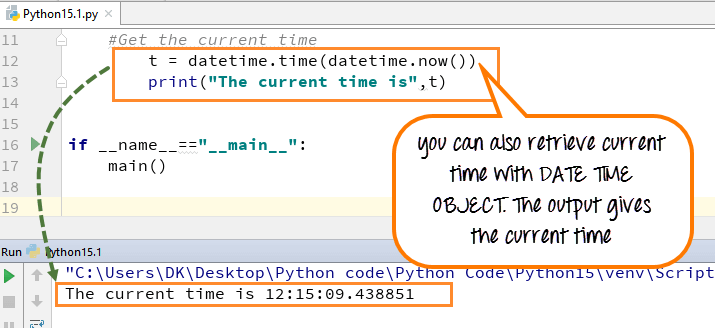
**Step 2)** With "DATETIME OBJECT", you can also call time class.

Suppose we want to print just the current time without the date.

t = datetime.time(datetime.now())

* We had imported the time class. We will be assigning it the current value of time using datetime.now()
* We are assigning the value of the current time to the variable t.

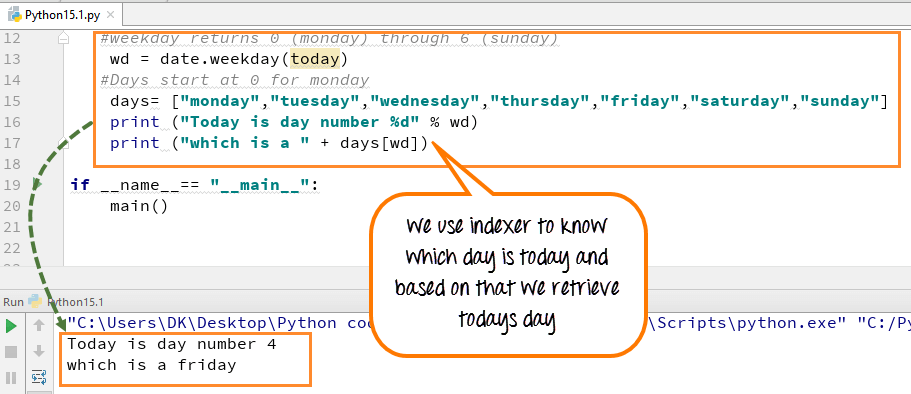
And this will give me just the time. So let's run this program.



Okay, so you can see that here I got the date and time. And then the next line, I've got just the time by itself

**Step 3)** We will apply our weekday indexer to our weekday's arrayList to know which day is today

* Weekdays operator (wd) is assigned the number from (0-6) number depending on what the current weekday is. Here we declared the array of the list for days (Mon, Tue, Wed…Sun).
* Use that index value to know which day it is. In our case, it is #2, and it represents Wednesday, so in the output it will print out "Which is a Wednesday."



Here is the complete code to get current date and time using datetime now

**Here is the complete code to get current date and time using datetime now**

from datetime import date

from datetime import time

from datetime import datetime

def main():

##DATETIME OBJECTS

#Get today's date from datetime class

today=datetime.now()

#print (today)

# Get the current time

#t = datetime.time(datetime.now())

#print "The current time is", t

#weekday returns 0 (monday) through 6 (sunday)

wd=date.weekday(today)

#Days start at 0 for monday

days= ["monday","tuesday","wednesday","thursday","friday","saturday","sunday"]

print("Today is day number %d" % wd)

print("which is a " + days[wd])

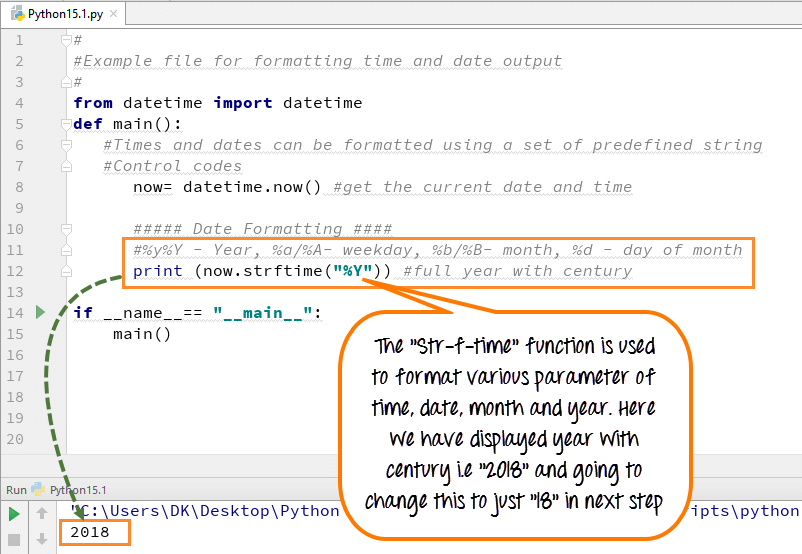
if \_\_name\_\_== "\_\_main\_\_":

main()

## How to Format Date and Time Output with Strftime()

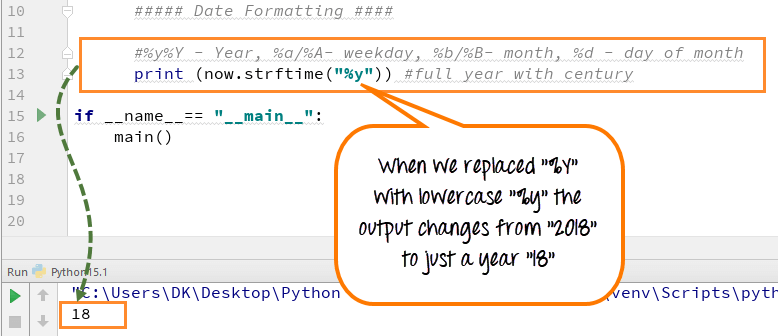
As of now we have learned, how to use datetime and date object in Python. We will advance a step further and learn how to use a formatting function to format Time and Date.

**Step 1)**First we will see a simple step of how to format the year. It is better to understand with an example.

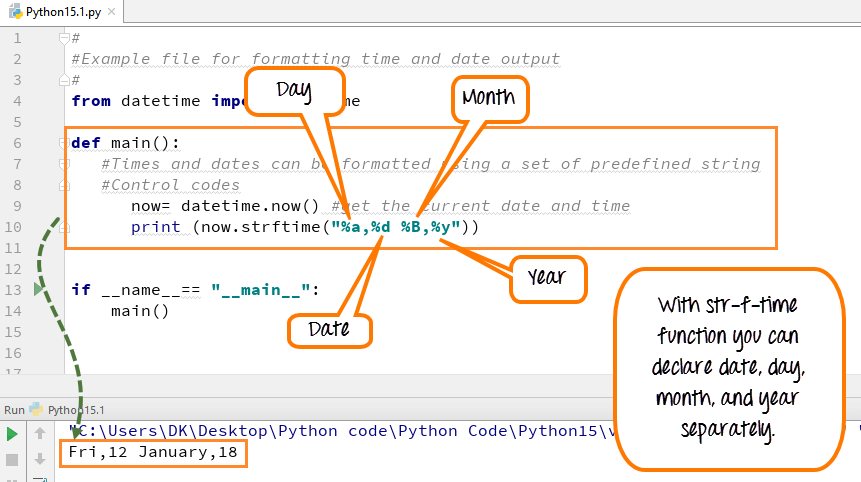


* We used the "**strftime function"** for formatting.
* This function uses different **control code** to give an output.
* Each control code resembles different parameters like year,month, weekday and date **[(%y/%Y – Year), (%a/%A- weekday), (%b/%B- month), (%d - day of month)] .**
* In our case, it is **("%Y")**which resembles year, it prints out the full year with the century (e.g., 2018).

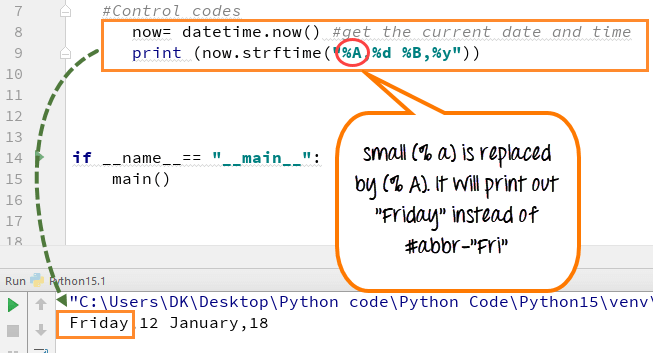
**Step 2)**Now if you replace ("%Y") with lowercase, i.e., ( "%y) and execute the code the output will display only (18) and not (2018). The century of the year will not display as shown in the screenshot below



**Step 3)** Strf function can declare the date, day, month and year separately. Also with small changes in the control code in strftime function you can format the style of the text.

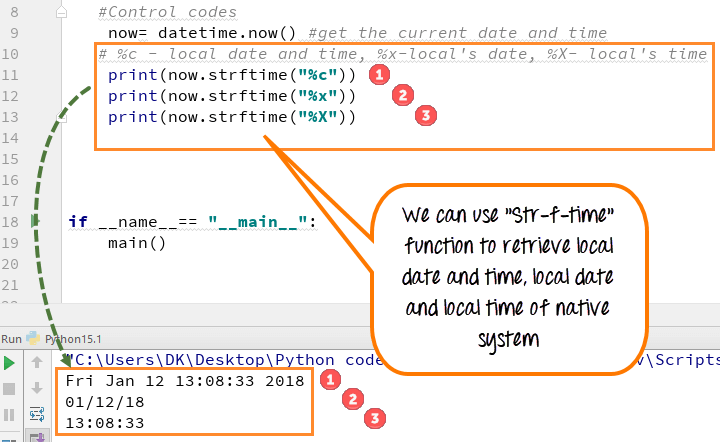


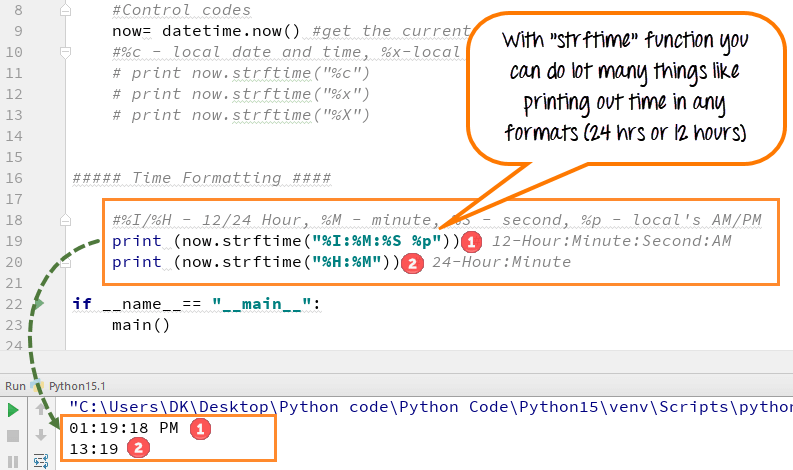
Inside the strftime function if you replace (%a) with capital A, i.e., (%A) the output will print out as "Firday" instead of just an abbreviation "Fri".



**Step 4)** With the help of "Strftime" function we can also retrieve local system time, date or both.

1. %C- indicates the local date and time
2. %x- indicates the local date
3. %X- indicates the local time



In the output, you can see the result as expected

Just by defining control code like %I/H for hour, % M for minute, %S for second, one can call time for different formats

**12 hours** time is declared [print now.strftime("%I:%M:%S %P) ]

**24 hours**time is declared [print now.strftime("%H:%M")]

**Here is the complete code to convert datetime to String object.**

#

#Example file for formatting time and date output

#

from datetime import datetime

def main():

#Times and dates can be formatted using a set of predefined string

#Control codes

now= datetime.now() #get the current date and time

#%c - local date and time, %x-local's date, %X- local's time

print(now.strftime("%c"))

print(now.strftime("%x"))

print(now.strftime("%X"))

##### Time Formatting ####

#%I/%H - 12/24 Hour, %M - minute, %S - second, %p - local's AM/PM

print(now.strftime("%I:%M:%S %p")) # 12-Hour:Minute:Second:AM

print(now.strftime("%H:%M")) # 24-Hour:Minute

if \_\_name\_\_== "\_\_main\_\_":

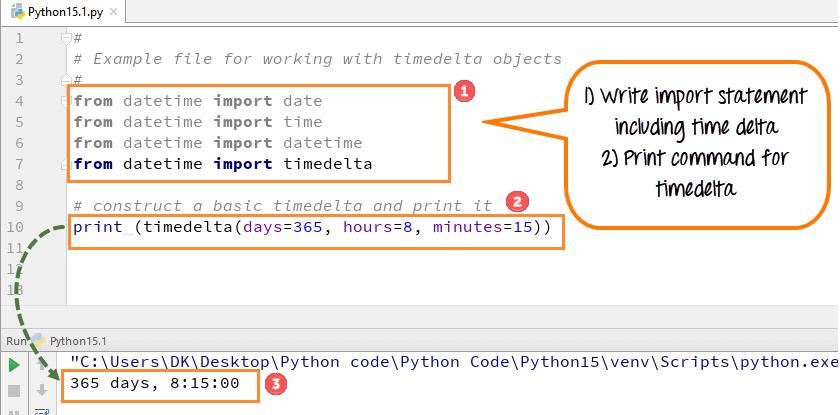
main()

## How to use Timedelta Objects

**With timedelta objects, you can estimate the time for both future and the past.** In other words, it is a timespan to predict any special day, date or time.

**Remember this function is not for printing out the time or date, but something to CALCULATE about the future or past**. Let's see an example to understand it better.

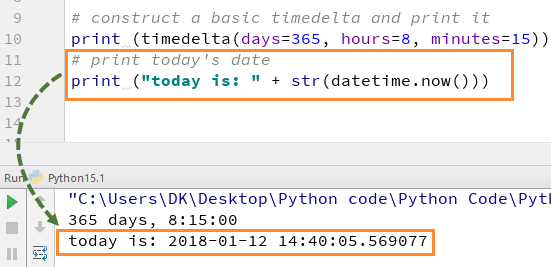
**Step 1)** To run Timedelta Objects, you need to declare the import statement first and then execute the code



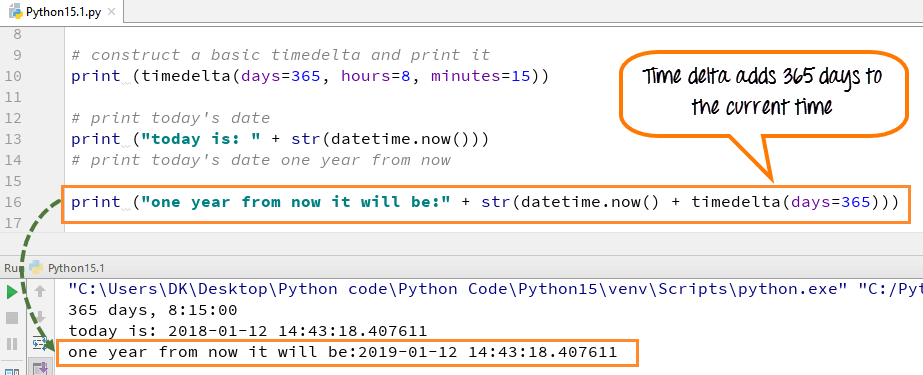
1. Write import statement for timedelta
2. Now write the code to print out object from time delta as shown in screen shot
3. Run the code. The timedelta represents a span of 365 days, 8 hrs and 15 minutes and prints the same

Confusing? Next step will help-

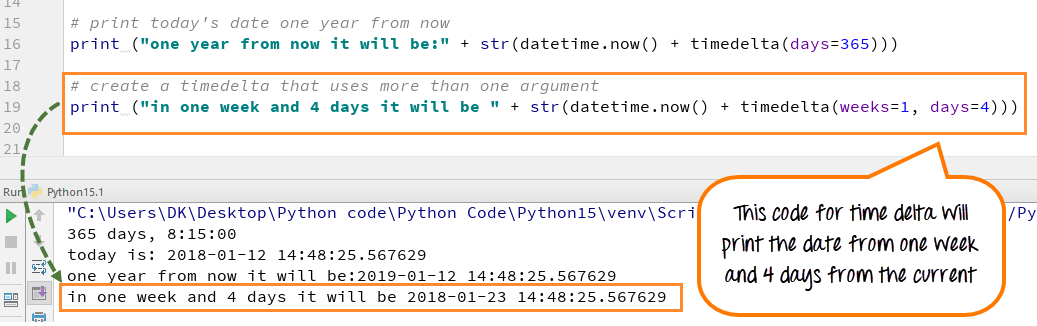
**Step 2)** Let's get today's date and time to check whether our import statement is working well. When code is executed, it prints out today's date which means our import statement is working well



**Step 3)** We will see how we can retrieve date a year from now through delta objects. When we run the code, it gives the output as expected.

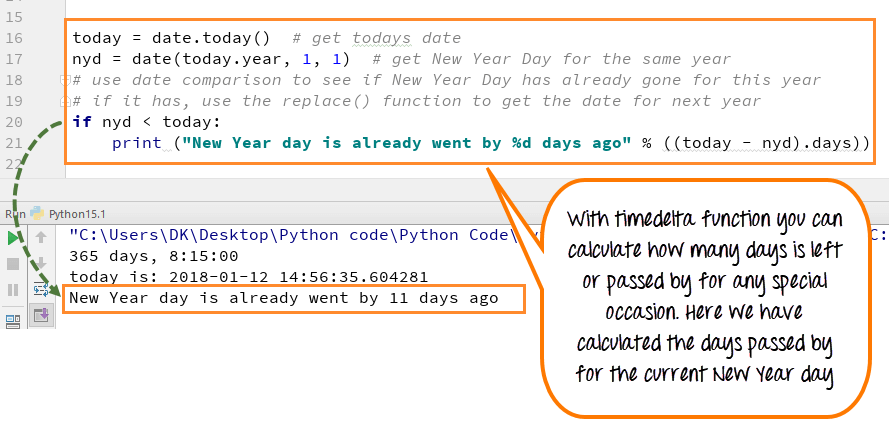


**Step 4)** Another example of how time delta can be used to calculate future date from current date and time



**Step 5)** Let's look into a more complex example. I would like to determine how many days past the New Year. Here is how we will proceed

* Using today= date.today() we will get todays date
* We know the newyear is always on 1-Jan, but the year could be different. Using nyd= date(today.year,1,1) we store the new year in variable nyd
* if nyd < today: compares whether the current date is greater than the new year. If yes, it enters the while loop
* ((today-nyd).days) gives the difference between a current date and new year in DAYS



The output shows that "New Year Day already went by 11 days ago."

**Here is the complete working code**

#

# Example file for working with timedelta objects

#

from datetime import date

from datetime import time

from datetime import datetime

from datetime import timedelta

# construct a basic timedelta and print it

print (timedelta(days=365, hours=8, minutes=15))

# print today's date

print ("today is: " + str(datetime.now()))

# print today's date one year from now

print ("one year from now it will be:" + str(datetime.now() + timedelta(days=365)))

# create a timedelta that uses more than one argument

# print (in one week and 4 days it will be " + str(datetime.now() + timedelta(weeks=1, days=4)))

# How many days until New Year's Day?

today = date.today() # get todays date

nyd = date(today.year, 1, 1) # get New Year Day for the same year

# use date comparison to see if New Year Day has already gone for this year

# if it has, use the replace() function to get the date for next year

if nyd < today:

print ("New Year day is already went by %d days ago" % ((today - nyd).days))

## Python 2 Example

from datetime import date

from datetime import time

from datetime import datetime

def main():

##DATETIME OBJECTS

#Get today's date from datetime class

today=datetime.now()

#print today

# Get the current time

#t = datetime.time(datetime.now())

#print "The current time is", t

#weekday returns 0 (monday) through 6 (sunday)

wd = date.weekday(today)

#Days start at 0 for monday

days= ["monday","tuesday","wednesday","thursday","friday","saturday","sunday"]

print "Today is day number %d" % wd

print "which is a " + days[wd]

if \_\_name\_\_== "\_\_main\_\_":

main()

#

#Example file for formatting time and date output

#

from datetime import datetime

def main():

#Times and dates can be formatted using a set of predefined string

#Control codes

now= datetime.now() #get the current date and time

#%c - local date and time, %x-local's date, %X- local's time

print now.strftime("%c")

print now.strftime("%x")

print now.strftime("%X")

##### Time Formatting ####

#%I/%H - 12/24 Hour, %M - minute, %S - second, %p - local's AM/PM

print now.strftime("%I:%M:%S %p") # 12-Hour:Minute:Second:AM

print now.strftime("%H:%M") # 24-Hour:Minute

if \_\_name\_\_== "\_\_main\_\_":

main()

#

# Example file for working with timedelta objects

#

from datetime import date

from datetime import time

from datetime import datetime

from datetime import timedelta

# construct a basic timedelta and print it

print timedelta(days=365, hours=8, minutes=15)

# print today's date

print "today is: " + str(datetime.now())

# print today's date one year from now

print "one year from now it will be:" + str(datetime.now() + timedelta(days=365))

# create a timedelta that uses more than one argument

# print "in one week and 4 days it will be " + str(datetime.now() + timedelta(weeks=1, days=4))

# How many days until New Year's Day?

today = date.today() # get todays date

nyd = date(today.year, 1, 1) # get New Year Day for the same year

# use date comparison to see if New Year Day has already gone for this year

# if it has, use the replace() function to get the date for next year

if nyd < today:

print "New Year day is already went by %d days ago" % ((today - nyd).days)

### Summary

For manipulating dates and times in both simple and complex ways datetime module supplies different classes or categories like

* date – Manipulate just date ( Month, day, year)
* time – Time independent of the day (Hour, minute, second, microsecond)
* datetime – Combination of time and date (Month, day, year, hour, second, microsecond)
* timedelta— A duration of time used for manipulating dates
* tzinfo— An abstract class for dealing with timezones

**Using datetime objects**

* Importing datetime objects before executing the code is mandatory
* Using date.today function for printing individual date/month/year as well as indexing the day
* Using date.time object to get time in hours, minutes, seconds and milliseconds

**Formatting Time-Out with "str f time function"**

* Use "str f time function" to change the format of the year
* Print day, date, month and year separately,
* Call out time for any format 12 hrs or 24 hrs

**Timedelta Objects**

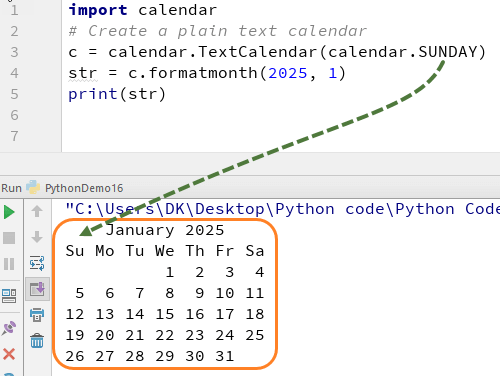
* With timedelta objects, you can estimate the time for both future and the past
* Calculate the total days left for the special day(birthday) from the current time
* Calculate the total days passed for special day(birthday) from the current time

**Python CALENDAR Tutorial with Example**

Calendar module in Python has the calendar class that allows the calculations for various task based on date, month, and year. On top of it, the TextCalendar and HTMLCalendar class in Python allows you to edit the calendar and use as per your requirement.

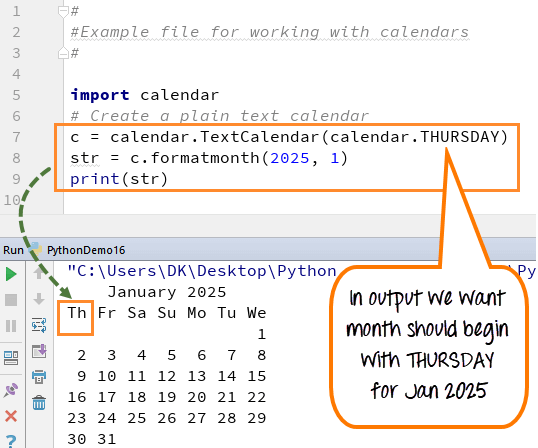
Let see what we can do with Python Calendar.

**Step1)** Run the code.

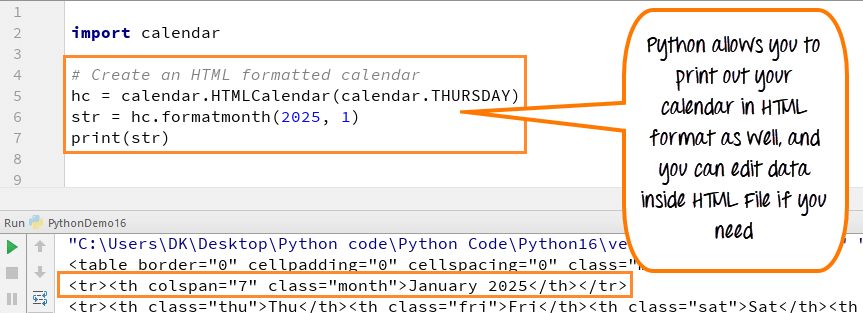


* Code Line # 1: We begin with "import calendar" which will import all the classes of this module.
* Code Line # 3: c= calendar.TextCalendar(calendar.SUNDAY) tells the interpreter to create a text calendar. Start of the month will be Sunday. In Python, you can format the calendar as you can change the day of the month to begin with
* Code Line # 4: str= c.formatmonth(2025,1) We are creating calendar for the year 2025, Month 1 – January
* Code Line # 5: print str will print the output.

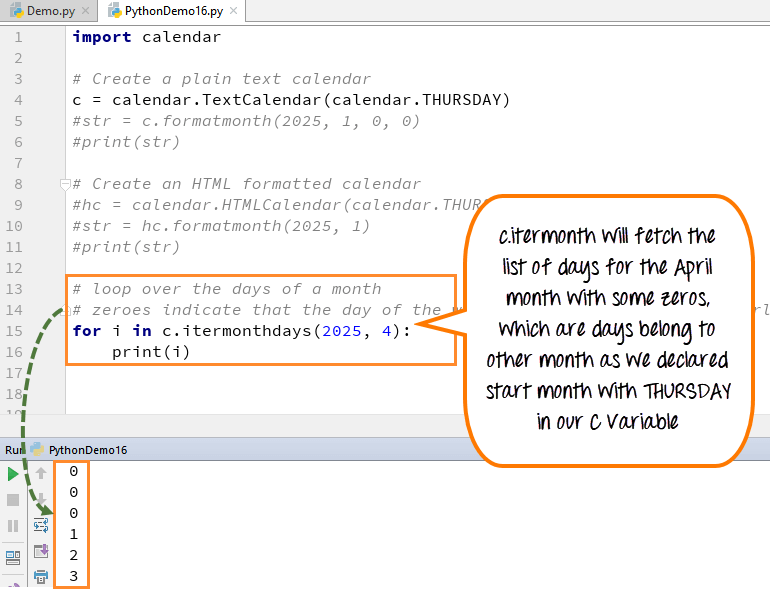
Let's quickly change the value from Sunday to Thursday and check the output



**Step 2)** You can also print out the Calendar in HTML format, this feature is helpful for developer if they want to make any changes in the look and feel of calendar

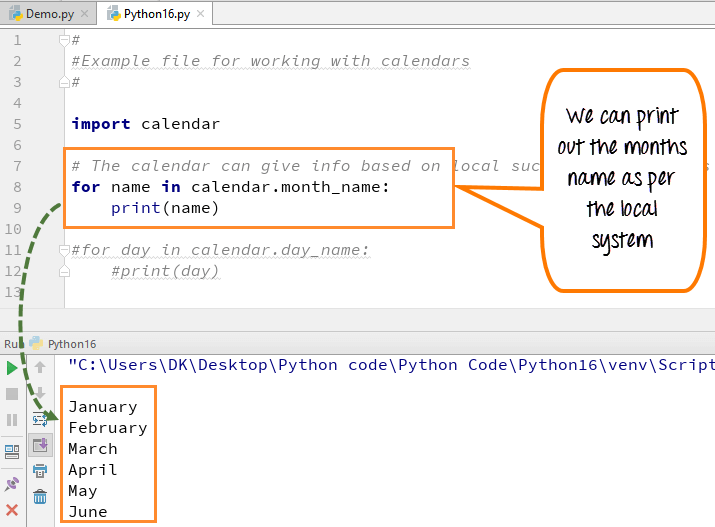


**Step 3)** Loops over the days of a month by using c.itermonthday (2025,4), it will fetch the total number of days for that month.

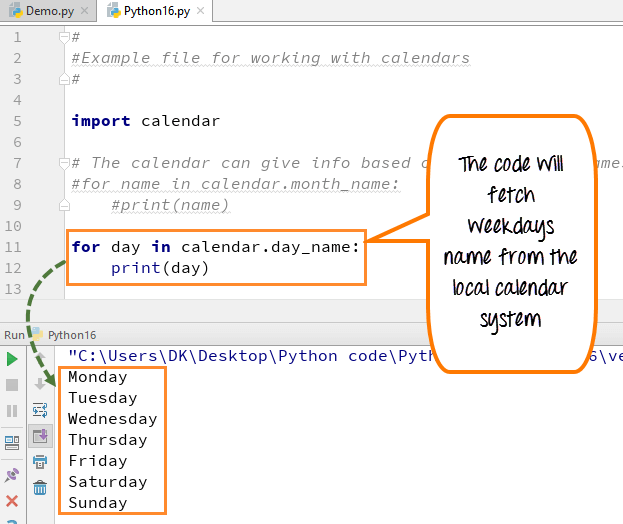


* When you execute the code to get the total number of days for a specific month say "April" then you will get 30 days in the output but you will also see some zeros along with these days at the beginning and sometimes at the end of it.
* Zeros in the output mean that the day of the week is in an overlapping month, which means it does not belong to that month.
* These zeros appears in output because, in your code you have mentioned day (Thursday), so when you call function "c.itermonthdays", it will start counting days from Thursday and your Thursday may not start with date 1st of April it might be 28thor 29th of March, so when you execute the code it will start counting days from 28th of march and any days after that till 1st of April. These days will be counted as zero and in the output you will see these zeroes and same is applicable to the end of the month.
* So except date 1-30 all the dates from previous as well as post month will appear in the output as zeroes.

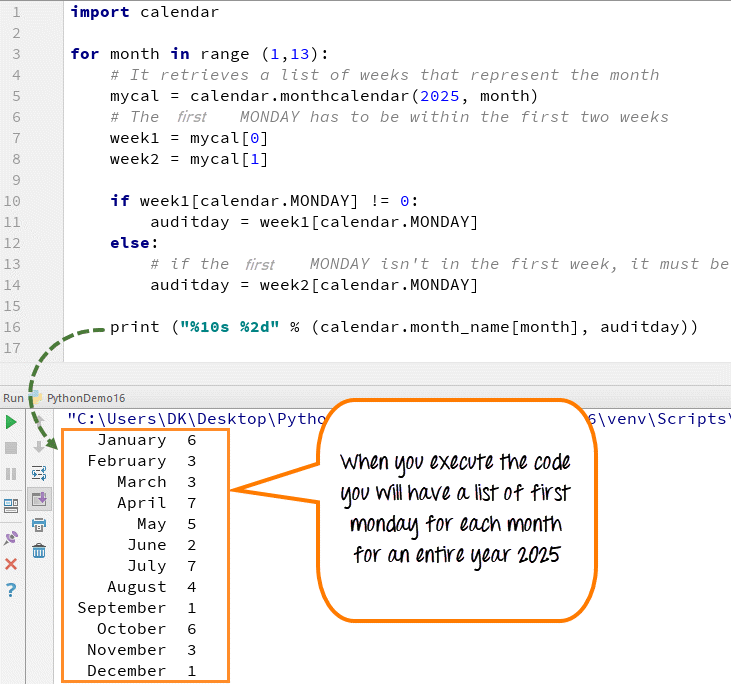
**Step 4)** You can fetch the data from the local system, like months or weekdays, etc



* The output over here shows that we have printed out the months name from the local system. Likewise, you can also fetch the weekdays name as shown below
* The output will depend on the local system, suppose if your local system is some other countries then it will give the output as per the local settings of that country. Here we have months so it won't be a difference but if it is a week or day, it will certainly differ.



**Step 5)** You can fetch the list of the specific day for a whole year. For example, there is an audit day on every first Monday of a week. You want to know the date of first Monday for each month. You can use this code



* mycal = calendar.monthcalendar(2025, month) will create calendar for the month
* Set variables week1 and week2 to the First and second week of the calendar
* Check if Week 1 contains Monday, set audit day
* Else set audit day as the first Monday in week 2
* The output shows the date for the first Monday that falls in that month.
* The length of this Cal object is going to be a certain length, based on how many weeks there in the month. In our case, it's going to be one or two as such first Monday of the week will be in the first week most often but if not then consider the second week. Let see in detail why we also consider the second week.
* Here we are using the calendar's constant Monday, the calendar object gives you constants that represent Sunday, Monday, Tuesday, so on, so forth. We've seen these previously. So, if in week one the day represented by the Monday constant is not equal to 0, remember zeros means days that belong to another month. So, in this case, if it's zero, it's going to be a Monday that belongs to the previous month. But if the first Monday is not equal to 0, that means that my audit day will be within the week one. Otherwise, if that is 0, then the first Monday isn't in the first week of the month, it's gotta be in the second.
* So, then I say okay, set my audit day variable to be the Monday represented by Week two. So, audit day is going to come back with whatever the day is for either the first or second week.

Here is the complete code

**Python 2 Example**

import calendar

# Create a plain text calendar

c = calendar.TextCalendar(calendar.THURSDAY)

str = c.formatmonth(2025, 1, 0, 0)

print str

# Create an HTML formatted calendar

hc = calendar.HTMLCalendar(calendar.THURSDAY)

str = hc.formatmonth(2025, 1)

print str

# loop over the days of a month

# zeroes indicate that the day of the week is in a next month or overlapping month

for i in c.itermonthdays(2025, 4):

print i

# The calendar can give info based on local such a names of days and months (full and abbreviated forms)

for name in calendar.month\_name:

print name

for day in calendar.day\_name:

print day

# calculate days based on a rule: For instance an audit day on the second Monday of every month

# Figure out what days that would be for each month, we can use the script as shown here

for month in range(1, 13):

# It retrieves a list of weeks that represent the month

mycal = calendar.monthcalendar(2025, month)

# The first MONDAY has to be within the first two weeks

week1 = mycal[1]

week2 = mycal[2]

if week1[calendar.MONDAY] != 0:

auditday = week1[calendar.MONDAY]

else:

# if the first MONDAY isn't in the first week, it must be in the second week

auditday = week2[calendar.MONDAY]

print "%10s %2d" % (calendar.month\_name[month], auditday)

**Python 3 Example**

import calendar

# Create a plain text calendar

c = calendar.TextCalendar(calendar.THURSDAY)

str = c.formatmonth(2025, 1, 0, 0)

print(str)

# Create an HTML formatted calendar

hc = calendar.HTMLCalendar(calendar.THURSDAY)

str = hc.formatmonth(2025, 1)

print(str)

# loop over the days of a month

# zeroes indicate that the day of the week is in a next month or overlapping month

for i in c.itermonthdays(2025, 4):

print(i)

# The calendar can give info based on local such a names of days and months (full and abbreviated forms)

for name in calendar.month\_name:

print(name)

for day in calendar.day\_name:

print(day)

# calculate days based on a rule: For instance an audit day on the second Monday of every month

# Figure out what days that would be for each month, we can use the script as shown here

for month in range(1, 13):

# It retrieves a list of weeks that represent the month

mycal = calendar.monthcalendar(2025, month)

# The first MONDAY has to be within the first two weeks

week1 = mycal[1]

week2 = mycal[2]

if week1[calendar.MONDAY] != 0:

auditday = week1[calendar.MONDAY]

else:

# if the first MONDAY isn't in the first week, it must be in the second week

auditday = week2[calendar.MONDAY]

print("%10s %2d" % (calendar.month\_name[month], auditday))

**Summary:**

* In Python, you can format the calendar the way you want as you can change the day of the month to begin
* Print out the Calendar in HTML format
* Fetch the data from the local system, like months or weekdays
* Fetch the list of the specific day for a whole year

**Python List: Comprehension, Apend, Sort, Length, Reverse EXAMPLES**

## What is a Python List?

A list is exactly what it sounds like, a container that contains different Python objects, which could be integers, words, values, etc. It is the equivalent of an array in other programming languages. It is represented by square brackets (and this is one of the attributes that differentiates it from tuples, which are separated by parentheses). It is also mutable, that is, it can be modified or updated; unlike tuples, which are immutable.

In this python tutorial, you will learn:

* [What is a Python List?](https://www.guru99.com/python-list-comprehension-sort-examples.html#1)
* [Examples of Python lists:](https://www.guru99.com/python-list-comprehension-sort-examples.html#2)
* [Accessing values within lists](https://www.guru99.com/python-list-comprehension-sort-examples.html#3)
* [List slicing](https://www.guru99.com/python-list-comprehension-sort-examples.html#4)
* [Updating lists](https://www.guru99.com/python-list-comprehension-sort-examples.html#5)
* [Deleting list elements](https://www.guru99.com/python-list-comprehension-sort-examples.html#6)
* [Appending list elements](https://www.guru99.com/python-list-comprehension-sort-examples.html#7)
* [List built-in functions (methods)](https://www.guru99.com/python-list-comprehension-sort-examples.html#8)
* [Looping through lists](https://www.guru99.com/python-list-comprehension-sort-examples.html#9)
* [List comprehensions](https://www.guru99.com/python-list-comprehension-sort-examples.html#10)

## Examples of Python lists:

Python lists can be homogeneous, meaning that they can contain the same type of objects; or heterogeneous, including different types of objects.

Examples of homogeneous lists include:

list of integers = [1, 2, 3, 8, 33]

list of animals = ['dog', 'cat', 'goat']

list of names = ['John', 'Travis', 'Sheila']

list of floating numbers = [2.2, 4.5, 9.8, 10.4]

Examples of heterogeneous lists include:

[2, 'cat', 34.33, 'Travis']

[2.22, 33, 'pen']

## Accessing values within lists

To access values within lists, the index of the objects inside the lists can be used. An index in Python lists refers to the position of an element within an ordered list. For example:

list = [3, 22, 30, 5.3, 20]

* The first value in the list above, 3, has an index of 0
* The second value, 22, has an index of 1
* The third value, 30, has an index of 2

and so on. To access each of the values from the list, you would use:

list[0] to access 3

list[1] to access 22

list[2] to access 30

list[3] to access 5.3

list[4] to access 20

The last member of a list can also be accessed by using the index -1. For example,

list[-1] = 20

## List slicing

List slicing is the method of splitting a subset of a list, and the indices of the list objects are also used for this. For example, using the same list example above;

list[:] = [3, 22, 30, 5.3, 20] (all the members of the list];

list[1:3] = [22, 30] (members of the list from index 1 to index 3, without the member at index 3);

list[:4] = [3, 22, 30, 5.3] (members of the list from index 0 to index 4, without the member at index 4)

list[2:-1] = [30, 5.3] (members of the list from index 2, which is the third element, to the second to the last element in the list, which is 5.3).

Python lists are upper-bound exclusive, and this means that the last index during list slicing is usually ignored. That is why

list[2:-1] = [30, 5.3]

, and not [30, 5.3, 20]. The same goes for all the other list slicing examples given above.

## Updating lists

Let's say you have a list = [physics, chemistry, mathematics], and you want to change the list to [biology, chemistry, mathematics], effectively changing the member at index 0. That can easily be done by assigning that index to the new member you want.

That is,

list = [physics, chemistry, mathematics]

list[0] = biology

print(list)

Output: [biology, chemistry, mathematics]

This replaces the member at index 0 (physics) with the new value you want (chemistry). This can be done for any member or subset of the list you want to change.

To give another example; let's say you have a list called integers and containing the numbers [2, 5, 9, 20, 27]. To replace 5 in that list with 10, you can do that with:

integers = [2, 5, 9, 20, 27]

integers[1] = 10

print(integers)

>>> [2, 10, 9, 20, 27]

To replace the last member of the list of integers, which is 27, with a free number like 30.5, you would use:

integers = [2, 5, 9, 20, 27]

integers[-1] = 30.5

print(integers)

>>> [2, 5, 9, 20, 30.5]

## Deleting list elements

There are 3 Python methods for deleting list elements: list.remove(), list.pop(), and del operator. Remove method takes the particular element to be removed as an argument while pop and del take the index of the element to be removed as an argument. For example:

list = [3, 5, 7, 8, 9, 20]

To delete 3 (the 1st element) from the list, you could use:

* list.remove(3) or
* list.pop[0], or
* del list[0]

To delete 8, the item at index 3, from the list, you could use:

* list.remove(8), or
* list.pop[3]

## Appending list elements

To append elements to a list, the append method is used, and this adds the element to the end of the list.

For example:

list\_1 = [3, 5, 7, 8, 9, 20]

list\_1.append(3.33)

print(list\_1)

>>> list\_1 = [3, 5, 7, 8, 9, 20, 3.33]

list\_1.append("cats")

print(list\_1)

>>> list\_1 = [3, 5, 7, 8, 9, 20, 3.33, "cats"]

## List built-in functions (methods)

The following is a list of list built-in functions and methods with their descriptions:

* **len(list):** this gives the length of the list as output. For example:

numbers = [2, 5, 7, 9]

print(len(numbers))

>>> 4

* **max(list):** returns the item in the list with the maximum value. For example:

numbers = [2, 5, 7, 9]

print(max(numbers))

>>> 9

* **min(list):** returns the item in the list with the minimum value. For example:

numbers = [2, 5, 7, 9]

print(min(numbers))

>>> 2

* **list(tuple):** converts a tuple object a list. For example;

animals = (cat, dog, fish, cow)

print(list(animals))

>>> [cat, dog, fish, cow]

* **list.append**(element): appends the element to the list. For example;

numbers = [2, 5, 7, 9]

numbers.append(15)

print(numbers)

>>> [2, 5, 7, 9, 15]

* **list.pop**(index): removes the element at the specified index from the list. For example;

numbers = [2, 5, 7, 9, 15]

numbers.pop(2)

print(numbers)

>>> [2, 5, 9, 15]

* **list.remove**(element):deletes the element from the list.For example;

values = [2, 5, 7, 9]

values.remove(2)

print(values)

>>> [5, 7, 9]

* **list.reverse**(): reverses the objects of the list. For example;

values = [2, 5, 7, 10]

values.reverse()

print(values)

>>> [10, 7, 5, 2]

* **list.index**(element): to get the index value of an element within the list. For example;

animals = ['cat', 'dog', 'fish', 'cow', 'goat']

fish\_index = animals.index('fish')

print(fish\_index)

>>> 2

* **sum**(list): to get the sum of all the values in the list, if the values are all numbers (integers or decimals). For example;

values = [2, 5, 10]

sum\_of\_values = sum(values)

print(sum\_of\_values)

>>> 17

If the list contains any element that is not a number, such as a string, the sum method would not work. You would get an error saying: "**TypeError**: unsupported operand type(s) for +: 'int' and 'str'"

* list.sort(): to arrange a list of integers, floating point numbers, or strings, in ascending or descending order. For example:

values = [1, 7, 9, 3, 5]

# To sort the values in ascending order:

values.sort()

print(values)

>>> [1, 3, 5, 7, 9]

Another example:

values = [2, 10, 7, 14, 50]

# To sort the values in descending order:

values.sort(reverse = True)

print(values)

>>> [50, 14, 10, 7, 2]

A list of strings can also be sorted, either alphabetically, or by length of the strings. For example;

# to sort the list by length of the elements

strings = ['cat', 'mammal', 'goat', 'is']

sort\_by\_alphabet = strings.sort()

sort\_by\_length = strings.sort(key = len)

print(sort\_by\_alphabet)

print(sort\_by\_length)

>>> ['cat', 'goat', 'is', 'mammal']

['is', 'cat', 'goat', 'mammal']

We can sort the same list alphabetically by using 'strings.

## Looping through lists

Looping through lists can be done in just the same way as any other looping function in Python. This way, a method can be performed on multiple elements of a list at the same time. For example:

list = [10, 20, 30, 40, 50, 60, 70].

To loop through all the elements of this list, and let's say, add 10 to each element:

for elem in list:

elem = elem + 5

print(elem)

>>>>15

25

35

45

55

65

75

To loop through the first three elements of the list, and delete all of them;

for elem in list[:3]:

list.remove(elem)

>>>list = [40, 50, 60, 70]

To loop through the 3rd (index 2) to last element on the list, and append them to a new list called new\_list:

new\_list = []

for elem in list[2:]:

new\_list.append(elem)

print(“New List: {}”.format(new\_list))

Output:

New List: [30, 40, 50, 60, 70]

In this way, any or method or function can be applied to the members of a list to perform a particular operation. You can either loop through all the members of the list, or loop through a subset of the list by using list slicing.

## List Comprehensions

List comprehensions are Python functions that are used for creating new sequences (such as lists, dictionaries, etc.) using sequences that have already been created. They help to reduce longer loops and make your code easier to read and maintain.

For example; let's say you wanted to create a list which contains the squares of all the numbers from 1 to 9:

list\_of squares = []

for int in range(1, 10):

square = int \*\* 2

list\_of\_squares.append(square)

print(list\_of\_squares)

List\_of\_squares using for loop:

[1, 4, 9, 16, 25, 36, 49, 64, 81]

To do the same thing with list comprehensions:

list\_of\_squares\_2 = [int\*\*2 for int in range(1, 10)]

print('List of squares using list comprehension: {}'.format(list\_of\_squares\_2))

Output using list comprehension:

[1, 4, 9, 16, 25, 36, 49, 64, 81]

As seen above, writing the code using list comprehensions is much shorter than using traditional for loops, and is also faster. This is just one example of using list comprehensions in place of for loops, but this can be replicated and used in a lot of places where for loops can also be used. Sometimes, going with a for loop is the better option, especially if the code is complex, but in many cases, list comprehensions will make your coding easier and faster.

Below is a table containing some list functions and methods, and their descriptions.

### Built-in Functions

|  |  |
| --- | --- |
| FUNCTION | DESCRIPTION |
| Round() | Rounds off the number passed as an argument to a specified number of digits and returns the floating point value |
| Min() | return minimum element of a given list |
| Max() | return maximum element of a given list |
| len() | Returns the length of the list |
| Enumerate() | This built-in function generates both the values and indexes of items in an iterable, so we don't need to count manually |
| Filter() | tests if each element of a list true or not |
| Lambda | An expression that can appear in places where a def (for creating functions) is not syntactic, inside a list literal or a function's call arguments |
| Map() | returns a list of the results after applying the given function to each item of a given iterable |
| Accumulate() | apply a particular function passed in its argument to all of the list elements returns a list containing the intermediate results |
| Sum() | Returns the sum of all the numbers in the list |
| Cmp() | This is used for comparing two lists and returns 1 if the first list is greater than the second list. |
| Insert | Insert element to list at particular position |

### List Methods

|  |  |
| --- | --- |
| FUNCTION | DESCRIPTION |
| Append() | Adds a new item to the end of the list |
| Clear() | Removes all items from the list |
| Copy() | Returns a copy of the original list |
| Extend() | Add many items to the end of the list |
| Count() | Returns the number of occurrences of a particular item in a list |
| Index() | Returns the index of a specific element of a list |
| Pop() | Deletes item from the list at particular index (delete by position) |
| Remove() | Deletes specified item from the list (delete by value) |
| Reverse() | In-place reversal method which reverses the order of the elements of the list |

## Summary

* A list is exactly what it sounds like, a container that contains different Python objects, which could be integers, words, values, etc.
* Python lists can be homogeneous, means they can contain the same type of objects; or heterogeneous, containing different types of objects.
* To access values within lists, the index of the objects inside the lists can be used.
* List slicing is the method of splitting a subset of a list, and the indices of the list objects are also used for this.
* Three methods for deleting list elements are : 1)list.remove(), 2)list.pop(), and 3)del operator
* Append method is used to append elements. This adds the element to the end of the list.
* Looping method of Python program can be performed on multiple elements of a data list at the same time.
* List comprehensions are Python functions that are used for creating new sequences (such as lists, dictionaries, etc.) using sequences that have already been created.

# Python File Handling: Create, Open, Append, Read, Write

In Python, there is no need for importing external library to read and write files. Python provides an inbuilt function for creating, writing and reading files.

In this tutorial, we will learn

* [How to Create a Text File](https://www.guru99.com/reading-and-writing-files-in-python.html#1)
* [How to Append Data to a File](https://www.guru99.com/reading-and-writing-files-in-python.html#2)
* [How to Read a File](https://www.guru99.com/reading-and-writing-files-in-python.html#3)
* [How to Read a File line by line](https://www.guru99.com/reading-and-writing-files-in-python.html#4)
* [File Modes in Python](https://www.guru99.com/reading-and-writing-files-in-python.html#5)

## How to Create a Text File

With Python you can create a .text files (guru99.txt) by using the code, we have demonstrated here how you can do this

**Step 1)**

f= open("guru99.txt","w+")

* We declared the variable f to open a file named textfile.txt. Open takes 2 arguments, the file that we want to open and a string that represents the kinds of permission or operation we want to do on the file
* Here we used "w" letter in our argument, which indicates write and the plus sign that means it will create a file if it does not exist in library
* Plus sign means, if File is not there, then create it
* The available option beside "w" are, "r" for read, and "a" for append

**Step 2)**

for i in range(10):

f.write("This is line %d\r\n" % (i+1))

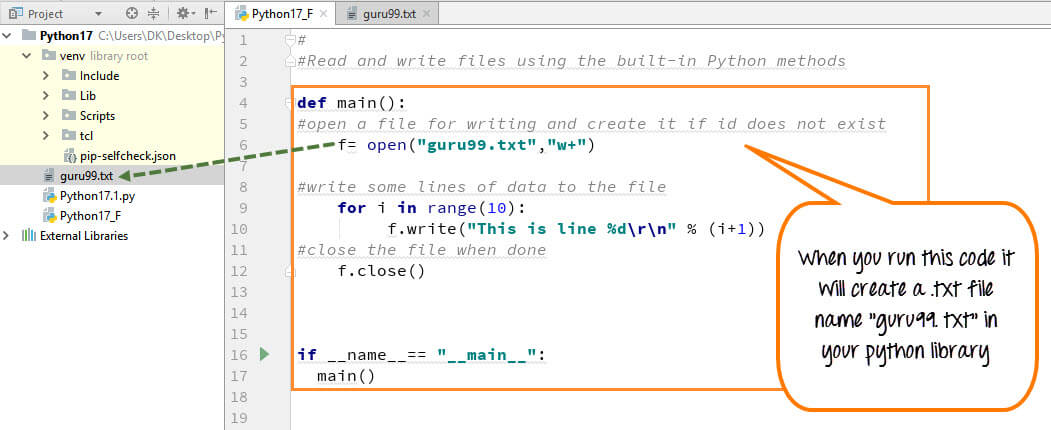
* We have a for loop that runs over a range of 10 numbers.
* Using the **write** function to enter data into the file.
* The output we want to iterate in the file is "this is line number", which we declare with write function and then percent d (displays integer)
* So basically we are putting in the line number that we are writing, then putting it in a carriage return and a new line character

**Step 3)**

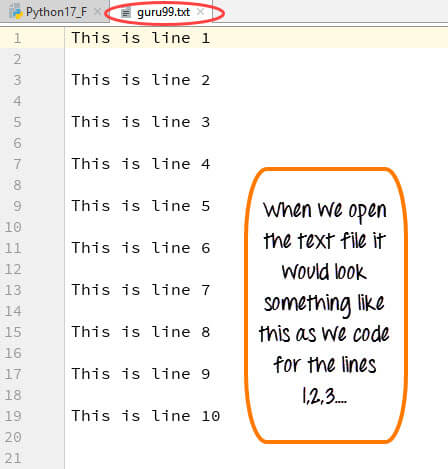
f.close()

* This will close the instance of the file guru99.txt stored

Here is the result after code execution



When you click on your text file in our case "guru99.txt" it will look something like this



## How to Append Data to a File

You can also append a new text to the already existing file or the new file.

**Step 1)**

f=open("guru99.txt", "a+")

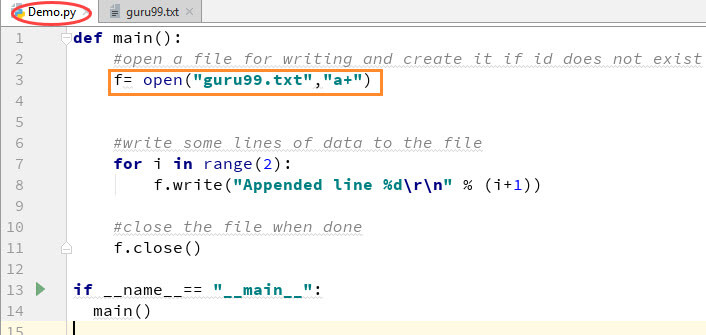
Once again if you could see a plus sign in the code, it indicates that it will create a new file if it does not exist. But in our case we already have the file, so we are not required to create a new file.

**Step 2)**

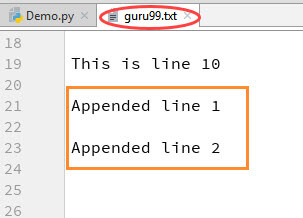
for i in range(2):

f.write("Appended line %d\r\n" % (i+1))

This will write data into the file in append mode.



You can see the output in "guru99.txt" file. The output of the code is that earlier file is appended with new data.

[](https://www.guru99.com/images/Pythonnew/Python17.4.jpg)

## How to Read a File

Not only you can create .txt file from Python but you can also call .txt file in a "read mode"(r).

**Step 1)** Open the file in Read mode

f=open("guru99.txt", "r")

**Step 2)** We use the mode function in the code to check that the file is in open mode. If yes, we proceed ahead

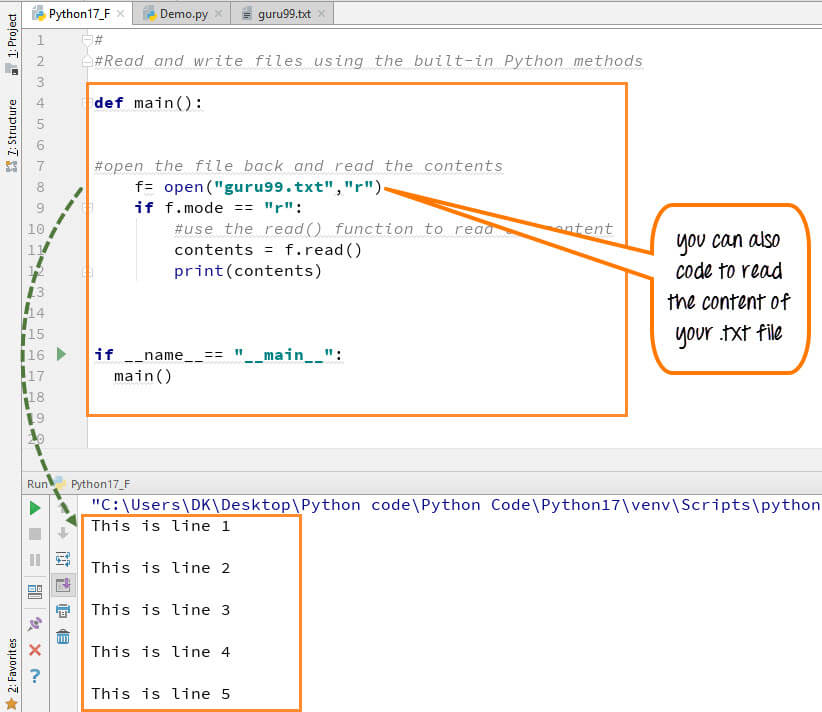
if f.mode == 'r':

**Step 3)** Use f.read to read file data and store it in variable content

contents =f.read()

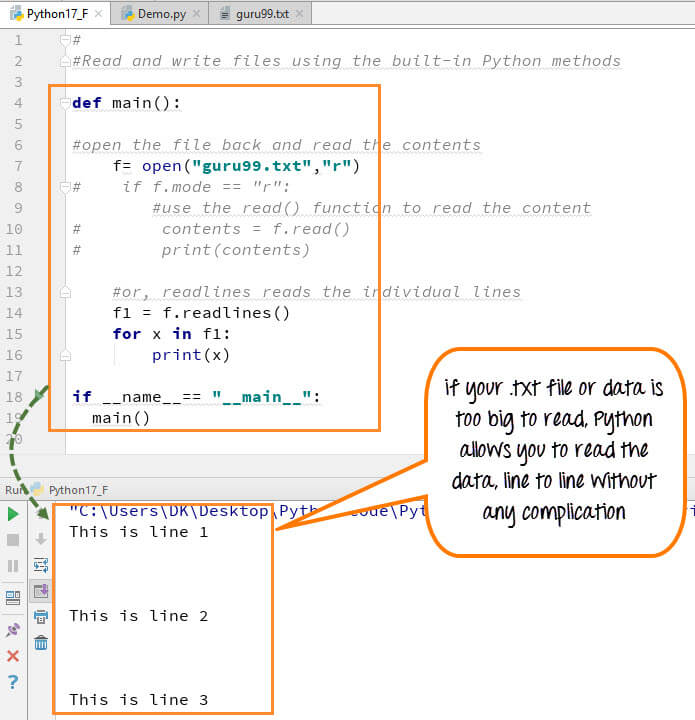
**Step 4)**print contents

Here is the output



## How to Read a File line by line

You can also read your .txt file line by line if your data is too big to read. This code will segregate your data in easy to ready mode



When you run the code (**f1=f.readlines())** for reading the file or document line by line, it will separate each line and present the file in a readable format. In our case the line is short and readable, the output will look similar to the read mode. But if there is a complex data file which is not readable, this piece of code could be useful.

## File Modes in Python

|  |  |
| --- | --- |
| **Mode** | **Description** |
| 'r' | This is the default mode. It Opens file for reading. |
| 'w' | This Mode Opens file for writing. If file does not exist, it creates a new file. If file exists it truncates the file. |
| 'x' | Creates a new file. If file already exists, the operation fails. |
| 'a' | Open file in append mode. If file does not exist, it creates a new file. |
| 't' | This is the default mode. It opens in text mode. |
| 'b' | This opens in binary mode. |
| '+' | This will open a file for reading and writing (updating) |

Here is the complete code

**Python 2 Example**

def main():

f= open("guru99.txt","w+")

#f=open("guru99.txt","a+")

for i in range(10):

f.write("This is line %d\r\n" % (i+1))

f.close()

#Open the file back and read the contents

#f=open("guru99.txt", "r")

# if f.mode == 'r':

# contents =f.read()

# print contents

#or, readlines reads the individual line into a list

#fl =f.readlines()

#for x in fl:

#print x

if \_\_name\_\_== "\_\_main\_\_":

main()

**Python 3 Example**

def main():

f= open("guru99.txt","w+")

#f=open("guru99.txt","a+")

for i in range(10):

f.write("This is line %d\r\n" % (i+1))

f.close()

#Open the file back and read the contents

#f=open("guru99.txt", "r")

#if f.mode == 'r':

# contents =f.read()

# print (contents)

#or, readlines reads the individual line into a list

#fl =f.readlines()

#for x in fl:

#print(x)

if \_\_name\_\_== "\_\_main\_\_":

main()

**Summary**

* Python allows you to read, write and delete files
* Use the function open("filename","w+") to create a file. The + tells the python interpreter to open file with read and write permissions.
* To append data to an existing file use the command open("Filename", "**a**")
* Use the read function to read the ENTIRE contents of a file
* Use the readlines function to read the content of the file one by one.

**Python Check If File or Directory Exists**

In this tutorial, we will learn how to determine whether a file (or directory) exists using Python. To check this, we use Built-in library functions.

There are different ways to verify a file or directory exists, using functions as listed below.

* [os.path.exists()](https://www.guru99.com/python-check-if-file-exists.html#1)
* [os.path.isfile()](https://www.guru99.com/python-check-if-file-exists.html#2)
* [os.path.isdir()](https://www.guru99.com/python-check-if-file-exists.html#3)
* [pathlibPath.exists()](https://www.guru99.com/python-check-if-file-exists.html#4)

## os.path.exists()

Using path.exists you can quickly check that a file or directory exists. Here are the steps

**Steps 1)** Before you run the code, it is important that you import the os.path module.

import os.path

from os import path

**Steps 2)** Now, use the path.exists() function to check whether a File Exists.

path.exists("guru99.txt")

**Steps 3)** Here is the complete code

import os.path

from os import path

def main():

print ("file exist:"+str(path.exists('guru99.txt')))

print ("File exists:" + str(path.exists('career.guru99.txt')))

print ("directory exists:" + str(path.exists('myDirectory')))

if \_\_name\_\_== "\_\_main\_\_":

main()

In our case only file guru99.txt is created in the working directory

**Output:**

File exists: True  
File exists: False  
directory exists: False

## os.path.isfile()

We can use the isfile command to check whether a given input is a file or directory.

import os.path

from os import path

def main():

print ("Is it File?" + str(path.isfile('guru99.txt')))

print ("Is it File?" + str(path.isfile('myDirectory')))

if \_\_name\_\_== "\_\_main\_\_":

main()

**Output:**

Is it File? True  
Is it File? False

## os.path.isdir()

If we want to confirm that a given path points to a directory, we can use the os.path.dir() function

import os.path

from os import path

def main():

print ("Is it Directory?" + str(path.isdir('guru99.txt')))

print ("Is it Directory?" + str(path.isdir('myDirectory')))

if \_\_name\_\_== "\_\_main\_\_":

main()

**Output:**

Is it Directory? False  
Is it Directory? True

## pathlibPath.exists() For Python 3.4

Python 3.4 and above versions have pathlib Module for handling with file system path. It used object-oriented approach to check if file exist or not.

import pathlib

file = pathlib.Path("guru99.txt")

if file.exists ():

print ("File exist")

else:

print ("File not exist")

**Output:**

File exist

**Complete Code**

Here is the complete code

import os

from os import path

def main():

# Print the name of the OS

print(os.name)

#Check for item existence and type

print("Item exists:" + str(path.exists("guru99.txt")))

print("Item is a file: " + str(path.isfile("guru99.txt")))

print("Item is a directory: " + str(path.isdir("guru99.txt")))

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Output:**

Item exists: True  
Item is a file: True  
Item is a directory: False

## How to check If File Exists

* os.path.exists() – Returns True if path or directory does exists.
* os.path.isfile() – Returns True if path is File.
* os.path.isdir() - Returns True if path is Directory.
* pathlib.Path.exists() - Returns True if path or directory does exists. (In Python 3.4 and above versions)

# Python COPY File using shutil.copy(), shutil.copystat()

## Python Copy File Methods

Python provides in-built functions for easily copying files using the Operating System Shell utilities.

Following command is used to Copy File

shutil.copy(src,dst)

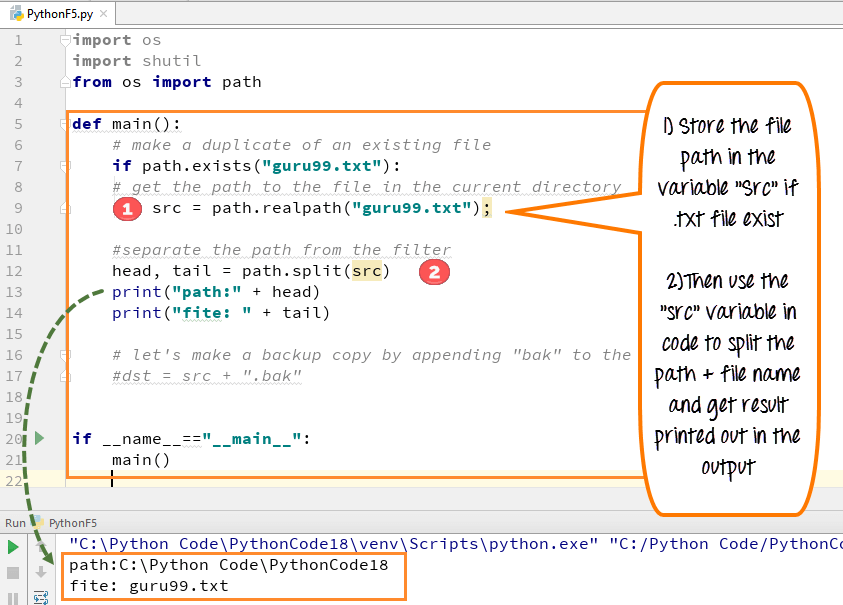
Following command is used to Copy File with MetaData Information

shutil.copystat(src,dst)

Here are the steps to copy file in Python

**Step 1)** Before, we copy a file, we need to get the the path to the original file in the current directory. In the code -

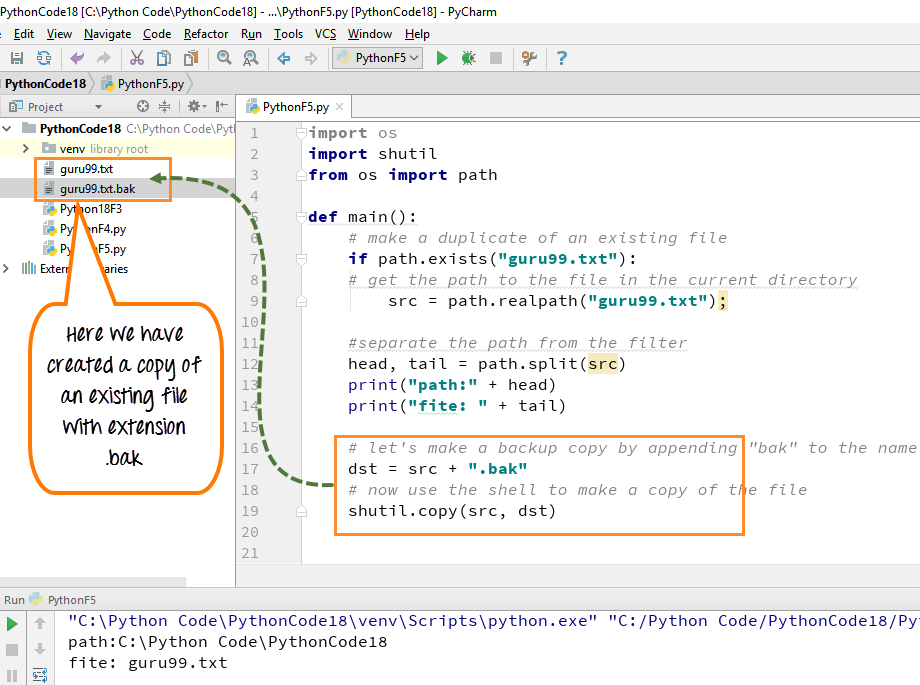
1. Declaring variable
2. Applying split function on variable



**Code Explanation**

* First we are going to check that our "guru99.txt" file exists or not. Since we have created guru99.txt file earlier, we know it exists, and we will carry on further with the code
* We store the file path in the variable "**src**" if your file exist
* Once we get the path, we going to separate the path and the file name
* For that, we are going to use the split path.split function on source variable
* Code when executed prints out "**file name**" and "**file path**" separately

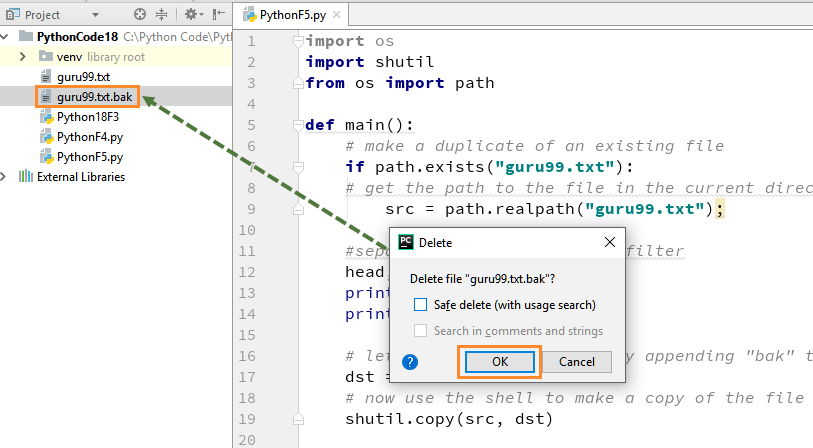
**Step 2)** We use Shutil Module to create a copy of the existing file. Here we used to create a copy of our existing file "guru99.txt."



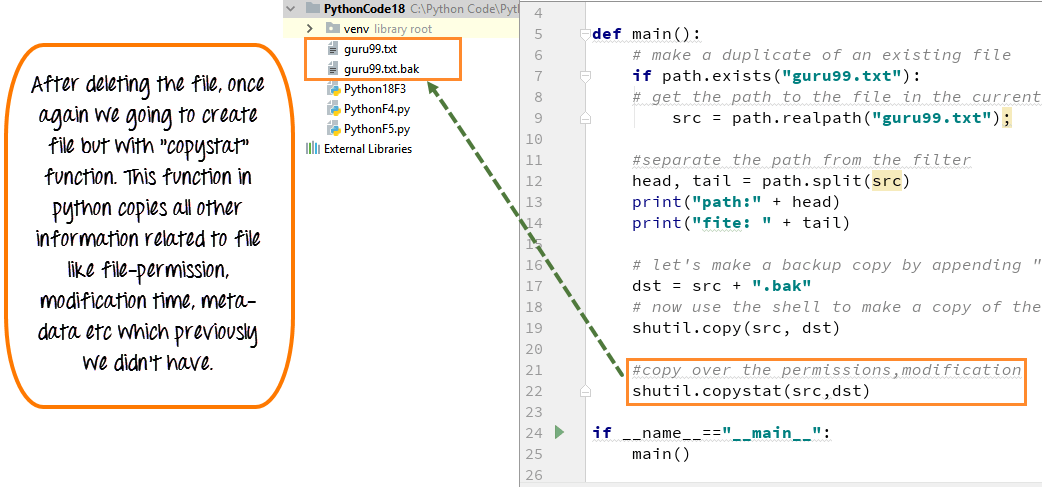
**Code Explanation**

* Take the original file name "guru99.txt" and add letters .bak at the end "guru99.txt.bak". This name with .bak extension is going to be our duplicate copy
* And then we going to use utility's copy function to copy from source to the destination
* When you run the code, you will see a duplicate file with .bak extension is created on right-hand side of the panel

**Step 3)** Copy function only copies the content of the file but no other information. To copy **meta-data** associated with the file, file permission and other information you have to use "**copystat**" function. Before we run this code, we have to delete our copy file "guru99.text.bak".



Once you deleted the file and run the program it will create a copy of your .txt file but this time with all the information like **file permission, modification time and meta-data information**. You can go to your O.S shell to verify the information.



Here is the code

import os

import shutil

from os import path

def main():

# make a duplicate of an existing file

if path.exists("guru99.txt"):

# get the path to the file in the current directory

src = path.realpath("guru99.txt");

#seperate the path from the filter

head, tail = path.split(src)

print("path:" +head)

print("file:" +tail)

#let's make a backup copy by appending "bak" to the name

dst = src+".bak"

# nowuse the shell to make a copy of the file

shutil.copy(src, dst)

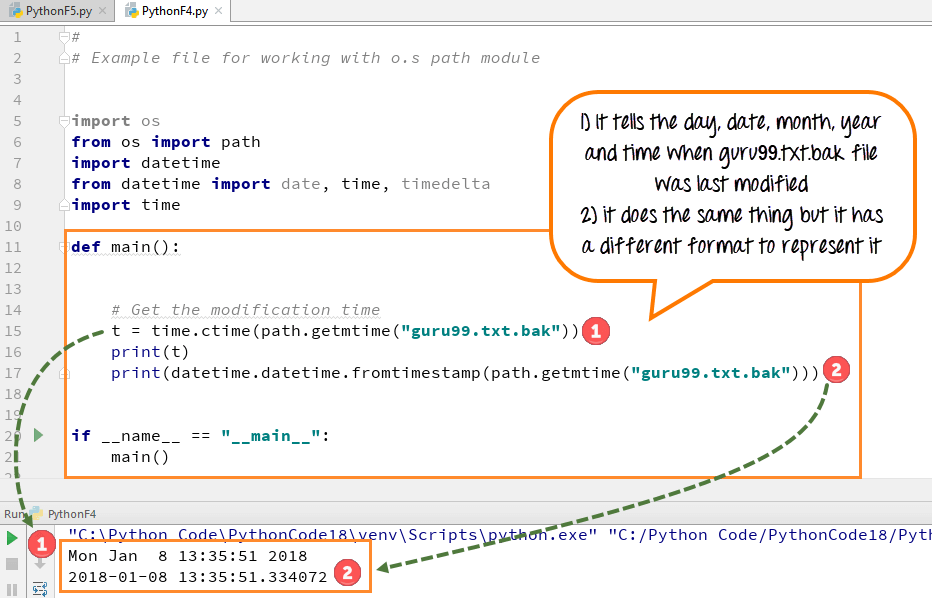
#copy over the permissions,modification

shutil.copystat(src,dst)

if \_\_name\_\_=="\_\_main\_\_":

main()

**Step 4)** You can fetch the information about the text file last modified



* Code Line#15- It tells the day, date, month, year and time when .txt file (guru99) was last modified. We use the path module to get the file modification time details, and then we are going to use the **time classes c time function**to convert that into a readable time. So when we run the code, we can see the file guru99.txt was last modified on **Mon, Jan 8th at 13:35 2018.**
* Code Line#17- It does the same thing giving information about file modification, but it has a different format to represent it. Here we use **Get Modification Time function** (path.getmtime("guru99.txt")). Now instead of using the c time function we going to use **From Time Stamp function**and going to construct a date time object. In output, you can see file modification time detail is printed out in different format **2018-01-08, 13:35:51.334072**

Here is the code

#

# Example file for working with o.s path module

import os

from os import path

import datetime

from datetime import date, time, timedelta

import time

def main():

# Get the modification time

t = time.ctime(path.getmtime("guru99.txt.bak"))

print(t)

print(datetime.datetime.fromtimestamp(path.getmtime("guru99.txt.bak")))

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Summary**

* To create a copy of the existing file by use code **shutil.copy (src,dst)**
* To copy all the information of original file to duplicate file like **file permission, modification time or meta-data information** by use code shutil.copystat(src,dst)

# Python Rename File and Directory using os.rename()

In Python, rename() method is used to rename a file or directory. It takes two arguments. Let's check the syntax.

### Syntax

This is the syntax for os.rename() method

os.rename(src, dst)

### Parameters

**src:** Source is the name of the file or directory. It should must already exist.

**dst:** Destination is the new name of the file or directory you want to change.

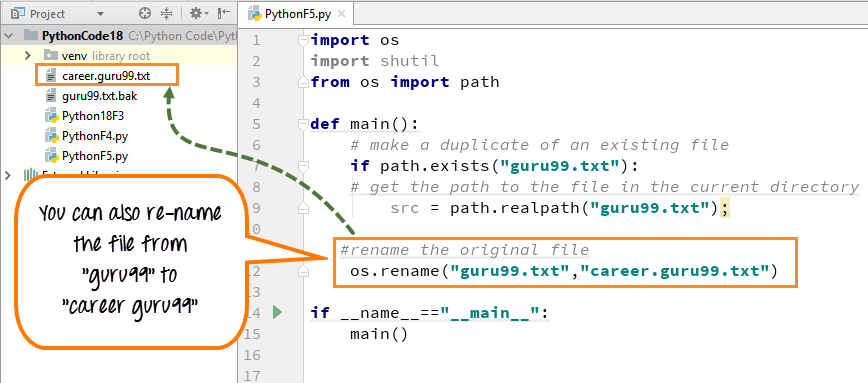
**Example:**

import os

os.rename('guru99.txt','career.guru99.txt')

Let's look at example in detail

You can rename the original file, we have changed the file name from "Guru99.txt" to "Career.guru99.txt."



* To rename "guru99.txt" file, we going to use "rename function" in the OS module
* So when the code is executed, you can observe that a new file "career.guru99.txt" is created on the right side of the panel, which we renamed for our original file.

Here is the complete code

import os

import shutil

from os import path

def main():

# make a duplicate of an existing file

if path.exists("guru99.txt"):

# get the path to the file in the current directory

src = path.realpath("guru99.txt");

# rename the original file

os.rename('guru99.txt','career.guru99.txt')

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Python ZIP file with Example**

Python allows you to quickly create zip/tar archives.

Following command will zip entire directory

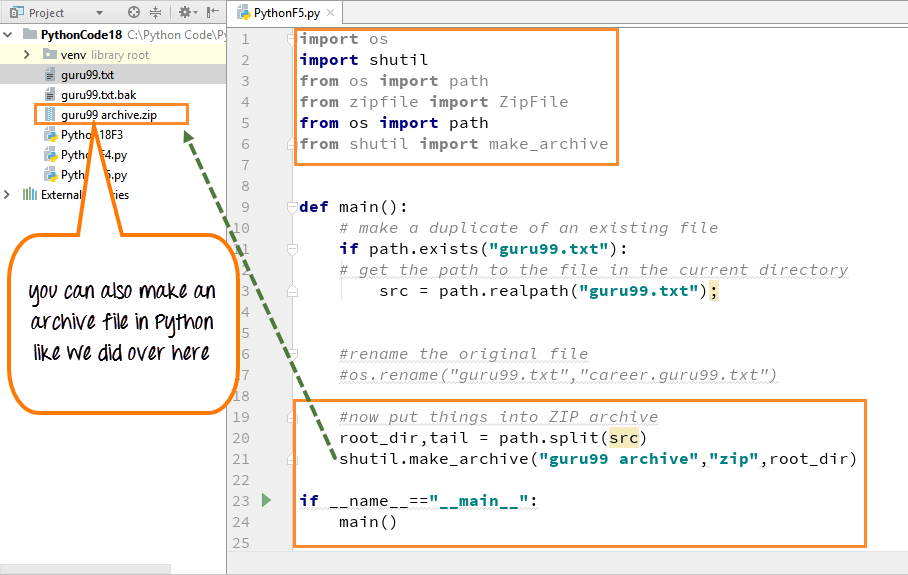
shutil.make\_archive(output\_filename, 'zip', dir\_name)

Following command gives you control on the files you want to archive

ZipFile.write(filename)

Here are the steps to create Zip File in Python

**Step 1)** To create an archive file from Python, make sure you have your import statement correct and in order. Here the import statement for the archive is from shutil import make\_archive

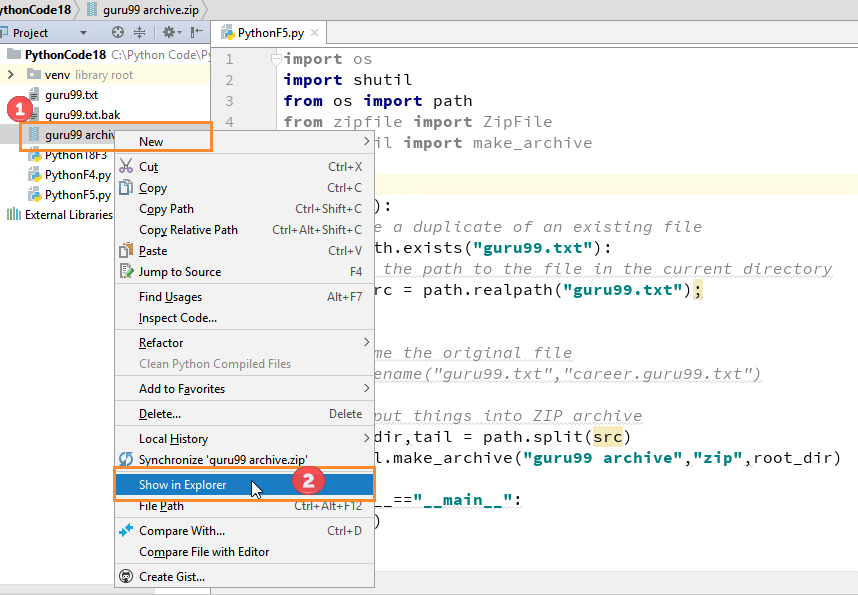


**Code Explanation**

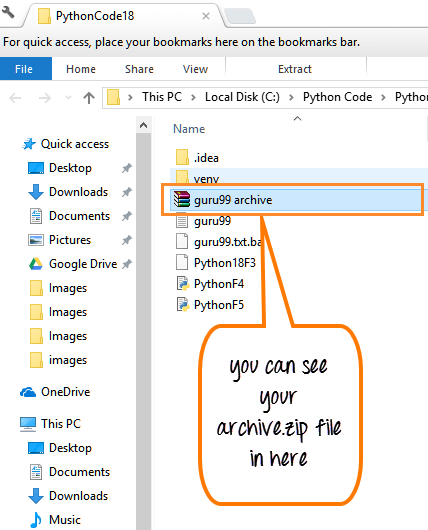
* Import make\_archive class from module shutil
* Use the split function to split out the directory and the file name from the path to the location of the text file (guru99)
* Then we call the module "shutil.make\_archive("guru99 archive, "zip", root\_dir)" to create archive file, which will be in zip format
* After then we pass in the root directory of things we want to be zipped up. So everything in the directory will be zipped
* When you run the code, you can see the archive zip file is created on the right side of the panel.

**Step 2)**

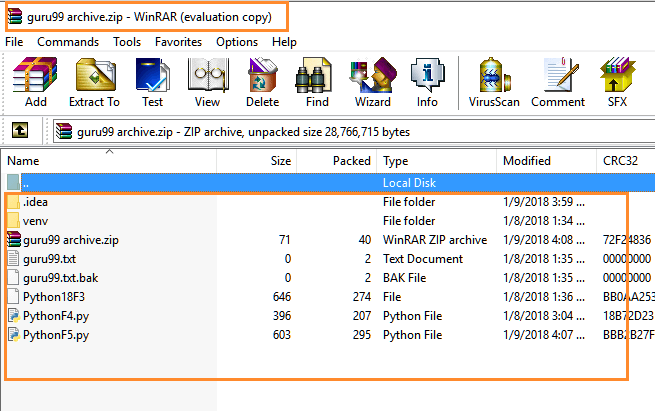
 Once your archive file is made, you can right-click on the file and select the O.S, and it will show your archive files in it as shown below



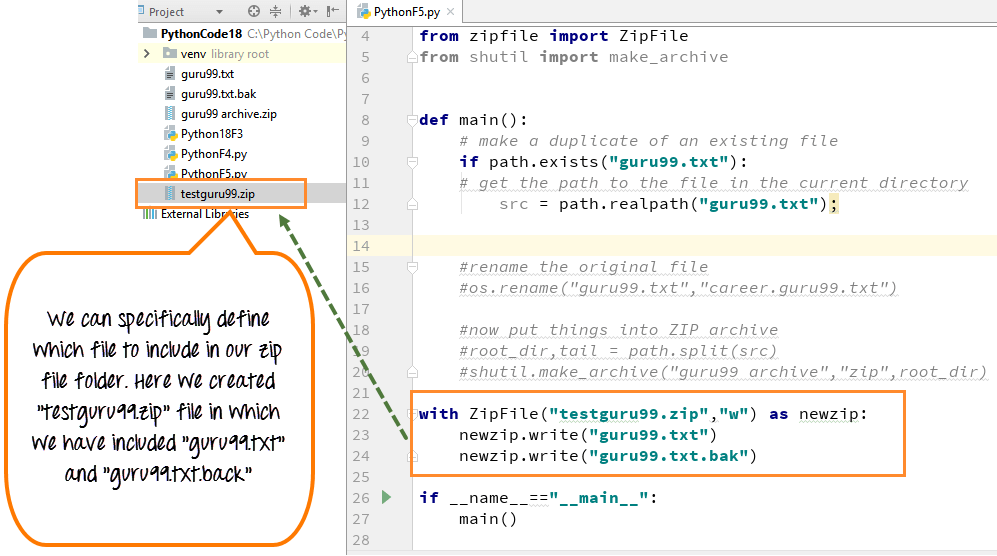
Now your archive.zip file will appear on your O.S (Windows Explorer)



**Step 3)** When you double-click on the file, you will see the list all the files in there.



**Step 4)** In Python we can have more control over archive since we can define which specific file to include under archive. In our case, we will include two files under archive **"guru99.txt"** and **"guru99.txt.bak".**



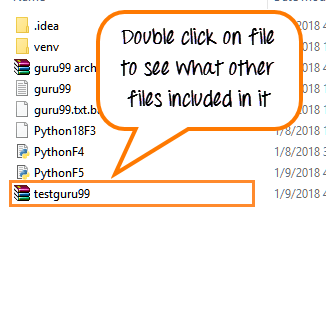
**Code Explanation**

* Import Zipfile class from zip file Python module. This module gives full control over creating zip files
* We create a new Zipfile with name ( "testguru99.zip, "w")
* Creating a new Zipfile class, requires to pass in permission because it's a file, so you need to write information into the file as newzip
* We used variable "newzip" to refer to the zip file we created
* Using the write function on the "newzip" variable, we add the files "guru99.txt" and "guru99.txt.bak" to the archive

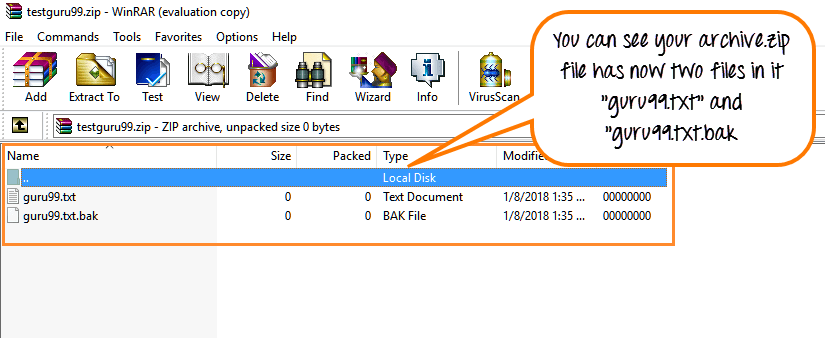
When you execute the code you can see the file is created on the right side of the panel with name "guru99.zip"

**Note**: Here we don't give any command to "close" the file like "newzip.close" because we use "With" scope lock, so when program falls outside of this scope the file will be cleaned up and is closed automatically.

**Step 5)** When you **-> right click on file (testguru99.zip) and -> select your O.S (Windows Explorer)**, it will show the archive files in the folder as shown below.

[](https://www.guru99.com/images/Pythonnew/Python18.15.png)

When you double click on file "testguru99.zip", it will open another window, and this will show the files included in it.



Here is the complete code

**Python 2 Example**

import os

import shutil

from zipfile import ZipFile

from os import path

from shutil import make\_archive

def main():

# Check if file exists

if path.exists("guru99.txt"):

# get the path to the file in the current directory

src = path.realpath("guru99.txt");

# rename the original file

os.rename("career.guru99.txt","guru99.txt")

# now put things into a ZIP archive

root\_dir,tail = path.split(src)

shutil.make\_archive("guru99 archive", "zip", root\_dir)

# more fine-grained control over ZIP files

with ZipFile("testguru99.zip","w") as newzip:

newzip.write("guru99.txt")

newzip.write("guru99.txt.bak")

if \_\_name\_\_== "\_\_main\_\_":

main()

**Python 3 Example**

import os

import shutil

from zipfile import ZipFile

from os import path

from shutil import make\_archive

# Check if file exists

if path.exists("guru99.txt"):

# get the path to the file in the current directory

src = path.realpath("guru99.txt");

# rename the original file

os.rename("career.guru99.txt","guru99.txt")

# now put things into a ZIP archive

root\_dir,tail = path.split(src)

shutil.make\_archive("guru99 archive","zip",root\_dir)

# more fine-grained control over ZIP files

with ZipFile("testguru99.zip", "w") as newzip:

newzip.write("guru99.txt")

newzip.write("guru99.txt.bak")

**Summary**

* To zip entire directory use command "shutil.make\_archive("name","zip", root\_dir)
* To select the files to zip use command "ZipFile.write(filename)"

# Python Internet Access using Urllib.Request and urlopen()

### What is urllib?

urllib is a Python module that can be used for opening URLs. It defines functions and classes to help in URL actions.

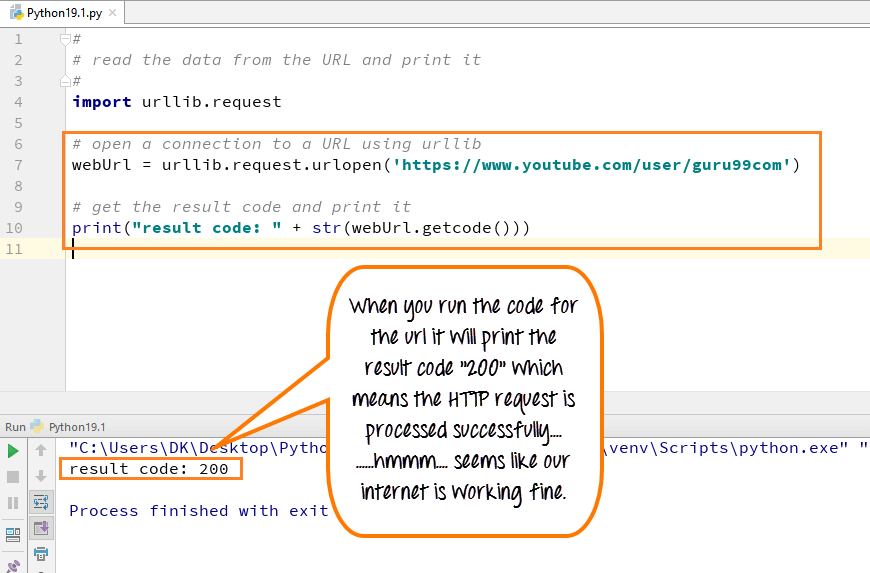
With Python you can also access and retrieve data from the internet like XML, HTML, JSON, etc. You can also use Python to work with this data directly. In this tutorial we are going to see how we can retrieve data from the web. For example, here we used a guru99 video URL, and we are going to access this video URL using Python as well as print HTML file of this URL.

In this tutorial we will learn

* [How to Open URL using Urllib](https://www.guru99.com/accessing-internet-data-with-python.html#1)
* [How to read HTML file for your URL in Python](https://www.guru99.com/accessing-internet-data-with-python.html#2)

## How to Open URL using Urllib

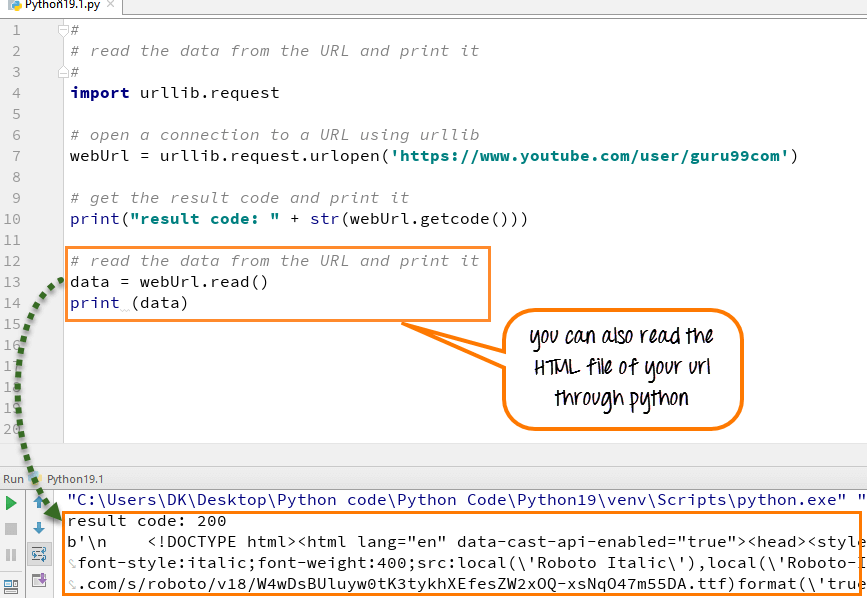
Before we run the code to connect to Internet data, we need to import statement for URL library module or "urllib".



* Import urllib
* Define your main function
* Declare the variable webUrl
* Then call the urlopen function on the URL lib library
* The URL we are opening is guru99 tutorial on youtube
* Next, we going to print the result code
* Result code is retrieved by calling the getcode function on the webUrl variable we have created
* We going to convert that to a string, so that it can be concatenated with our string "result code"
* This will be a regular HTTP code "200", indicating http request is processed successfully

## How to get HTML file form URL in Python

You can also read the HTML file by using the "read function" in Python, and when you run the code, the HTML file will appear in the console.



* Call the read function on the webURL variable
* Read variable allows to read the contents of data files
* Read the entire content of the URL into a variable called data
* Run the code- It will print the data into HTML format

Here is the complete code

**Python 2 Example**

#

# read the data from the URL and print it

#

import urllib2

def main():

# open a connection to a URL using urllib2

webUrl = urllib2.urlopen("https://www.youtube.com/user/guru99com")

#get the result code and print it

print "result code: " + str(webUrl.getcode())

# read the data from the URL and print it

data = webUrl.read()

print data

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Python 3 Example**

#

# read the data from the URL and print it

#

import urllib.request

# open a connection to a URL using urllib

webUrl = urllib.request.urlopen('https://www.youtube.com/user/guru99com')

#get the result code and print it

print ("result code: " + str(webUrl.getcode()))

# read the data from the URL and print it

data = webUrl.read()

print (data)

# Python XML Parser Tutorial: Read xml file example(Minidom, ElementTree)

### What is XML?

XML stands for eXtensible Markup Language. It was designed to store and transport small to medium amounts of data and is widely used for sharing structured information.

Python enables you to parse and modify XML document. In order to parse XML document you need to have the entire XML document in memory. In this tutorial, we will see how we can use XML minidom class in Python to load and parse XML file.

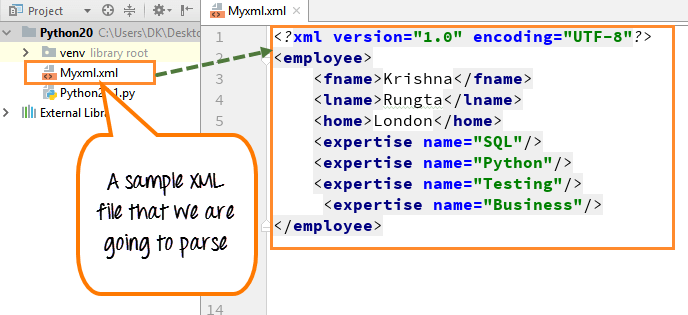
In this tutorial, we will learn-

* [How to Parse XML using minidom](https://www.guru99.com/manipulating-xml-with-python.html#1)
* [How to Create XML Node](https://www.guru99.com/manipulating-xml-with-python.html#2)
* [How to Parse XML using ElementTree](https://www.guru99.com/manipulating-xml-with-python.html#3)

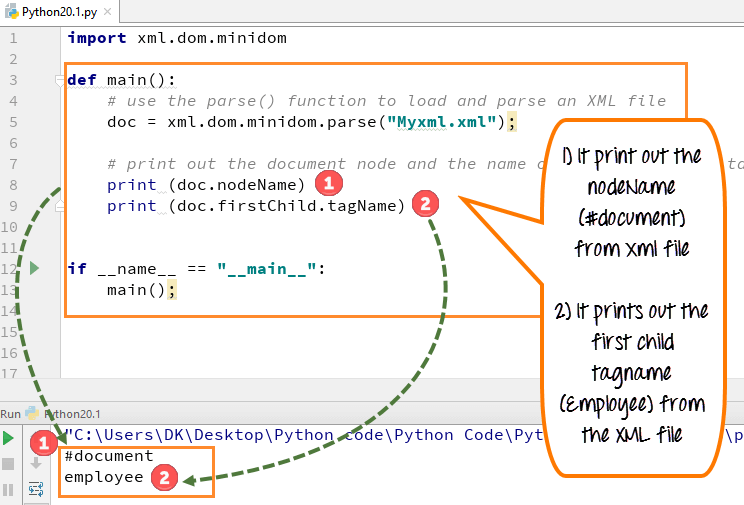
## How to Parse XML using minidom

We have created a sample XML file that we are going to parse.

**Step 1)** Inside file, we can see first name, last name, home and the area of expertise (SQL, Python, Testing and Business)



**Step 2)** Once we have parsed the document, we will print out the **"node name"** of the root of the document and the "**firstchild tagname"**. Tagname and nodename are the standard properties of the XML file.

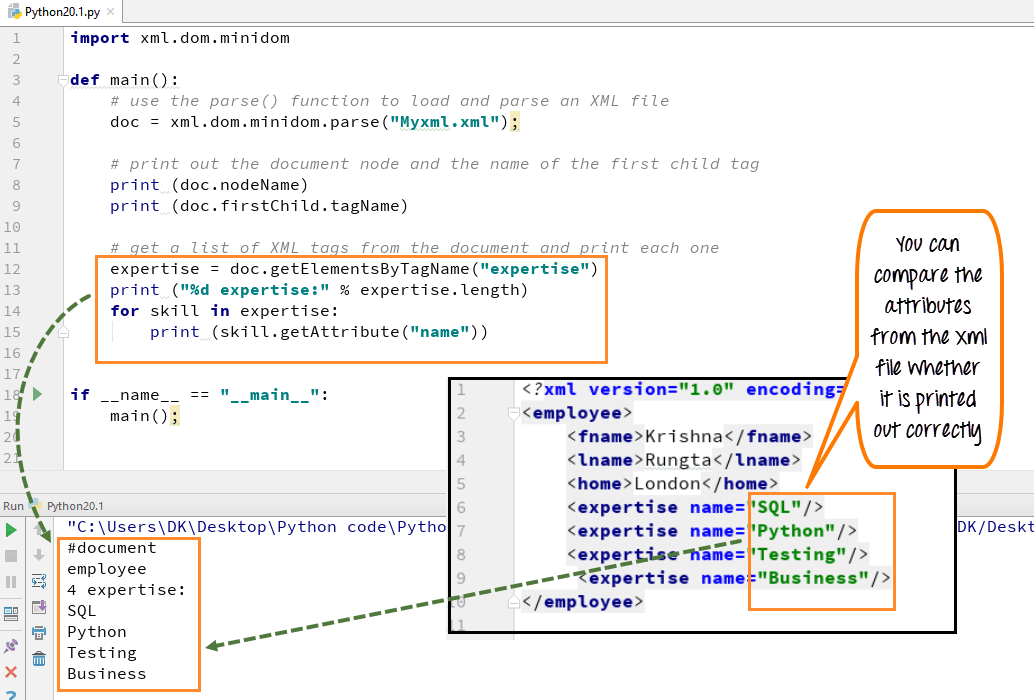


* Import the xml.dom.minidom module and declare file that has to be parsed (myxml.xml)
* This file carries some basic information about employee like first name, last name, home, expertise, etc.
* We use the parse function on the XML minidom to load and parse the XML file
* We have variable doc and doc gets the result of the parse function
* We want to print the nodename and child tagname from the file, so we declare it in print function
* Run the code- It prints out the nodename (#document) from the XML file and the first child tagname (employee) from the XML file

**Note**:

Nodename and child tagname are the standard names or properties of an XML dom. In case if you are not familiar with these type of naming conventions.

**Step 3)** We can also call the list of XML tags from the XML document and printed out. Here we printed out the set of skills like SQL, Python,[Testing](https://www.guru99.com/software-testing.html)and Business.

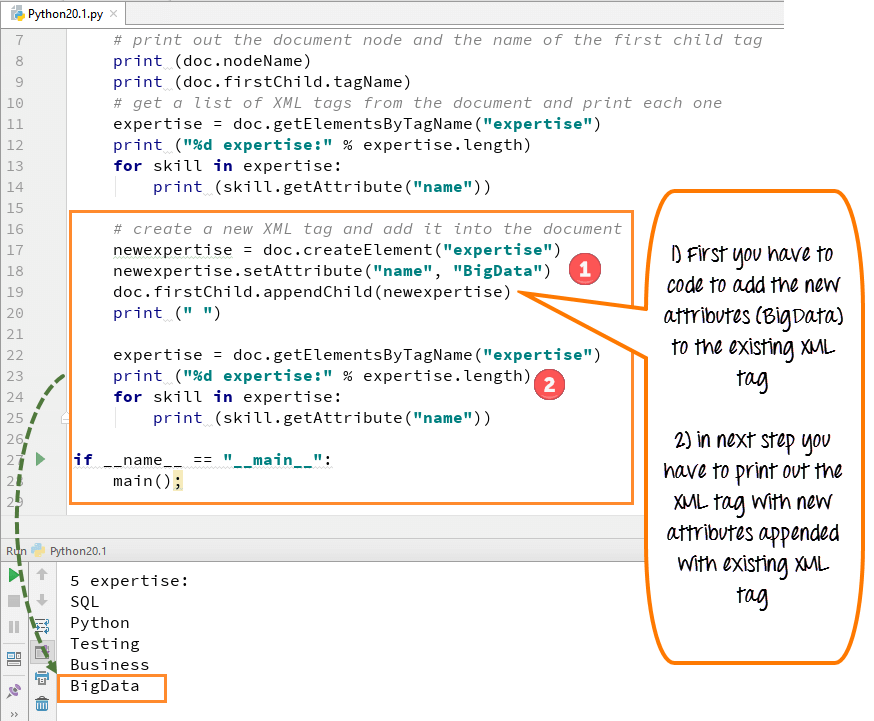


* Declare the variable expertise, from which we going to extract all the expertise name employee is having
* Use the dom standard function called "getElementsByTagName"
* This will get all the elements named skill
* Declare loop over each one of the skill tags
* Run the code- It will give list of four skills

## How to Create XML Node

We can create a new attribute by using "createElement" function and then append this new attribute or tag to the existing XML tags. We added a new tag "BigData" in our XML file.

1. You have to code to add the new attribute (BigData) to the existing XML tag
2. Then you have to print out the XML tag with new attributes appended with existing XML tag



* To add a new XML and add it to the document, we use code "doc.create elements"
* This code will create a new skill tag for our new attribute "Big-data"
* Add this skill tag into the document first child (employee)
* Run the code- the new tag "big data" will appear with the other list of expertise

### XML Parser Example

**Python 2 Example**

import xml.dom.minidom

def main():

# use the parse() function to load and parse an XML file

doc = xml.dom.minidom.parse("Myxml.xml");

# print out the document node and the name of the first child tag

print doc.nodeName

print doc.firstChild.tagName

# get a list of XML tags from the document and print each one

expertise = doc.getElementsByTagName("expertise")

print "%d expertise:" % expertise.length

for skill in expertise:

print skill.getAttribute("name")

# create a new XML tag and add it into the document

newexpertise = doc.createElement("expertise")

newexpertise.setAttribute("name", "BigData")

doc.firstChild.appendChild(newexpertise)

print " "

expertise = doc.getElementsByTagName("expertise")

print "%d expertise:" % expertise.length

for skill in expertise:

print skill.getAttribute("name")

if name == "\_\_main\_\_":

main();

**Python 3 Example**

import xml.dom.minidom

def main():

# use the parse() function to load and parse an XML file

doc = xml.dom.minidom.parse("Myxml.xml");

# print out the document node and the name of the first child tag

print (doc.nodeName)

print (doc.firstChild.tagName)

# get a list of XML tags from the document and print each one

expertise = doc.getElementsByTagName("expertise")

print ("%d expertise:" % expertise.length)

for skill in expertise:

print (skill.getAttribute("name"))

# create a new XML tag and add it into the document

newexpertise = doc.createElement("expertise")

newexpertise.setAttribute("name", "BigData")

doc.firstChild.appendChild(newexpertise)

print (" ")

expertise = doc.getElementsByTagName("expertise")

print ("%d expertise:" % expertise.length)

for skill in expertise:

print (skill.getAttribute("name"))

if \_\_name\_\_ == "\_\_main\_\_":

main();

## How to Parse XML using ElementTree

ElementTree is an API for manipulating XML. ElementTree is the easy way to process XML files.

We are using the following XML document as the sample data:

<data>

<items>

<item name="expertise1">SQL</item>

<item name="expertise2">Python</item>

</items>

</data>

**Reading XML using ElementTree:**

we must first import the xml.etree.ElementTree module.

import xml.etree.ElementTree as ET

Now let's fetch the root element:

root = tree.getroot()

Following is the complete code for reading above xml data

import xml.etree.ElementTree as ET

tree = ET.parse('items.xml')

root = tree.getroot()

# all items data

print('Expertise Data:')

for elem in root:

for subelem in elem:

print(subelem.text)

**output:**

Expertise Data:

SQL

Python

### Summary:

Python enables you to parse the entire XML document at one go and not just one line at a time. In order to parse XML document you need to have the entire document in memory.

* To parse XML document
  + Import xml.dom.minidom
  + Use the function "parse" to parse the document ( doc=xml.dom.minidom.parse (file name);
  + Call the list of XML tags from the XML document using code (=doc.getElementsByTagName( "name of xml tags")
* To create and add new attribute in XML document
  + Use function "createElement"

# Python vs JAVA vs PHP vs PERL vs Ruby vs JavaScript vs C++ vs TCL

Python is one of the most popular programming languages. Currently, each of the following six languages are being used by programmers for developing both desktop and web applications. That is why, it is important for programmers to compare Python with JAVA, RUBY, PHP, TCL and Perl to pick the right language for their projects.

|  |  |
| --- | --- |
| **Programming Languages** | **Difference between Python and other languages** |
| [Java](https://www.guru99.com/java-tutorial.html) | * Python takes less time to develop although it is expected to run slower than Java programs * Due to Python high-level data types and its dynamic typing programs are shorter than Java programs |
| Perl | * Although both these languages are considered similar in many ways,[Perl](https://www.guru99.com/perl-tutorials.html)supports common application-oriented task like report generating, file scanning, etc.while Python supports for common programming methodologies like data structure design and OOPs programming for programmers making language to readable code |
| Tcl | * Tcl is a standalone programming language it is considered weak on data structures * It is slower in executing code compared to Python |
| PHP | * Python has easy to read code while[PHP](https://www.guru99.com/php-tutorials.html)has more syntax from C/C++ and Perl * In Python, classes are used extensively in the standard library while PHP has SPL which is fully class based * Python supports structured exception handling while most PHP functions do not use exceptions for reporting errors * Development features in Python are provided by an add-on while in PHP it is built in * Python supports major GUI frameworks |
| RUBY | * Compare to Ruby, Python has a rich set of data structures, internal functions, better namespace handling and use of modules and iterators * Python supports multiple inheritance while Ruby does not * To close all its scopes, Ruby requires "end" or "}" while Python uses the white space only. * RUBY is preferred for Web development and functional programming while Python for Academic and scientific programming. * Ruby work on “Do more with less.” Philosophy while PYTHON “One right way to do things” |
| C++ | * Python code is quite shorter than most other programming languages like C or C++. * Python is a dynamically typed language, while C++ is a statically typed language. |
| JavaScript | * Python is server side. JavaScript is client side. * Python uses indentation and whitespace. JavaScript uses curly brackets to designate blocks of code. |

# PyQt Tutorial: Python GUI Designer

## What is PyQt?

PyQt is a python binding of the open-source widget-toolkit Qt, which also functions as a cross-platform application development framework. Qt is a popular C++ framework for writing GUI applications for all major desktop, mobile, and embedded platforms (supports Linux, Windows, MacOS, Android, iOS, Raspberry Pi, and more).

PyQt is developed and maintained by Riverbank Computing, a company based in England, whereas Qt is developed by a Finnish firm called The Qt Company.

In this PyQT tutorial, you will learn:

* [What is PyQt?](https://www.guru99.com/pyqt-tutorial.html#1)
* [Features of PyQT](https://www.guru99.com/pyqt-tutorial.html#2)
* [PyQt Versions](https://www.guru99.com/pyqt-tutorial.html#3)
* [How to install PyQt](https://www.guru99.com/pyqt-tutorial.html#4)
* [Basic PyQt Concepts and Programs](https://www.guru99.com/pyqt-tutorial.html#5)
* [Components and Widgets](https://www.guru99.com/pyqt-tutorial.html#6)
* [Themes](https://www.guru99.com/pyqt-tutorial.html#7)

## Features of PyQT

Here, are Important features of PyQt:

PyQt consists of more than six hundred classes covering a range of features such as

* Graphical User Interfaces
* SQL Databases
* Web toolkits
* XML processing
* Networking

These features can be combined to create advanced UIs as well as standalone applications. A lot of major companies across all industries use Qt. Some examples are LG, Mercedes, AMD, Panasonic, Harman, etc.

## PyQt Versions

PyQt is available in two editions, PyQt4 and PyQt5. PyQt4 provides glue code for binding 4.x and 5.x versions of the Qt framework while PyQt5 provides a binding for only the 5.x versions. As a result, PyQt5 is not backward compatible with the deprecated modules of the older version. In this tutorial, PyQt5 will be used for the demonstration of examples. Apart from these two versions,

Riverbank Computing also provides PyQt3D—the python bindings for the Qt3D framework. Qt3D is an application framework used to create real-time simulation systems with 2D/3D rendering.

## How to install PyQt

There are two ways of installing PyQt.

* Using Wheel files
* Building and Installing from Source

Qt (pronounced cute) is a complex system, and the PyQt codebase contains compiled C++ and Python code under the hood. As a result, it is a complicated process to build and install it from the source compared to other python libraries. However, you can easily install PyQt5 using wheels.

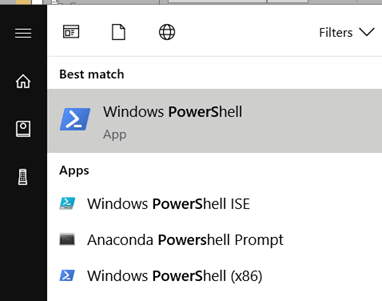
### Installation with wheels

Wheels are the new standard Python packaging and distribution format. Simply speaking, a wheel is a ZIP archive with a special name and .**whl**file extension. Wheels can be installed using pip (Python's package manager), which is included by default in the recent versions of Python.

So, if you have Python 3.4 or later installed, you already have pip. If, however, you are using an older version of Python, you must download and install pip before going forward. You can search for instructions for that at this link: <https://pypi.org/project/pip/>.

To install PyQt5,

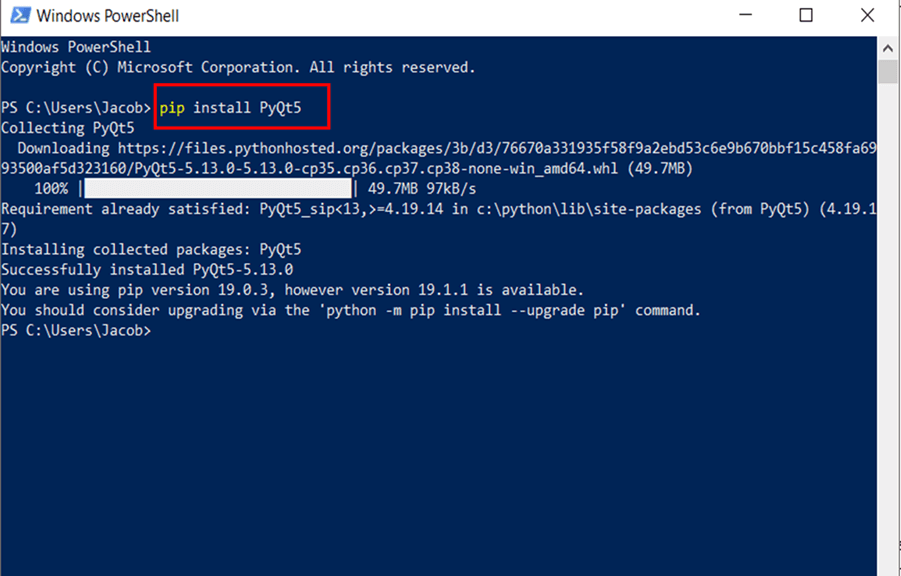
**Step 1)**Open the Command Prompt or PowerShell in your Windows machine.

[](https://www.guru99.com/images/1/070919_1308_PyQtTutoria1.png)

**Step 2)** Type in the following

pip install PyQt5

This step will download the PyQt5 whl package (about 50 MB) and install it on your system.



**Step 1)** Alternatively, you can also download a Windows binary for the version of python installed on your computer.

Once it is complete, proceed to the next section to write your first GUI app.

## Basic PyQt Concepts and Programs

Now that you have successfully installed PyQt5 in your computer, you are ready to write GUI applications in python.

Let's start with a simple app which will display an empty window on your screen.

Fire up your python IDLE and type in the following:

Program 1

import sys

from PyQt5.QtWidgets import QApplication, QWidget

if \_\_name\_\_ == "\_\_main\_\_":

app = QApplication(sys.argv)

w = QWidget()

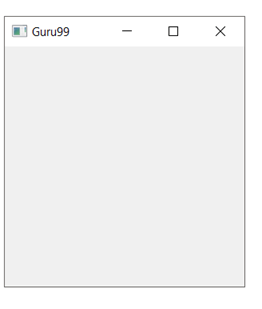
w.resize(300,300)

w.setWindowTitle(‘Guru99’)

w.show()

sys.exit(app.exec\_())

Save it as app.py (the name does not matter) and hit F5 to run the program. Alternatively, just double click your saved file to launch the application. If you have done everything right, a new window will open up with the title Guru99 as shown below.



Great! It's working. It is not much, but it's enough to grasp the basics. Now, let's see in detail what each of the lines in your program does.

from PyQt5.QtWidgets import QApplication, QWidget

This statement imports all the modules you need to create a GUI into the current namespace. The QtWidgets module contains all the major widgets that you will be using in this tutorial.

app = QApplication(sys.argv)

Here, you are creating an object of the QApplication class. This step is a necessity for PyQt5; every UI app must create an instance of QApplication, as a sort of entry point into the app. If you do not create it, errors will be shown.

sys.argv is the list of command-line parameters that you can pass to the application when launching it through the shell or while automating the interface.

In this example, you did not pass any arguments to QApplications. Therefore, you can also replace it with the code below and not even have to import the sys module.

app = QApplication([])

w = QWidget()

Next, we make an object of the QWidget class. QWidget is the base class of all UI objects in Qt, and virtually everything you see in an app is a widget. That includes dialogs, texts, buttons, bars, and so on. The feature that allows you to design complex user interfaces is that the widgets can be nested, i.e., you can have a widget inside a widget, which is inside yet another widget. You will see this in action in the next section.

w.resize(300,300)

The resize method of the QWidget class allows you to set the dimensions of any widget. In this case, you have resized the window to 300px by 300px.

Here, you should remember that widgets could be nested together, the outermost widget (i.e., the widget with no parent) is called a Window.

w.setWindowTitle('Guru99')

The setWindowTitle() method allows you to pass a string as an argument which will set the title of the window to the string you pass. In the example, the title bar will display Guru99.

w.show()

show() simply displays the widget on the monitor screen.

sys.exit(app.exec\_())

The app.exec\_() method starts the Qt/C++ event loop. As you know, PyQt is largely written in C++ and uses the event loop mechanism to implement parallel execution. app.exec\_() passes the control over to Qt which will exit the application only when the user closes it from the GUI. That is why ctrl+c will not exit the application as in other python programs. Since Qt has control over the app, python events are not processed unless we set them up inside the application. Also, note that the exec method has an underscore in its name; this is because exec() was already a keyword in python and the underscore resolves naming conflict.

### Beyond empty windows

In the previous section, you saw how to make a basic widget in Qt. It's now time to make some more involved interfaces with which the users can truly interact. Again, fire up your IDLE and write the following.

import sys

from PyQt5.QtWidgets import QApplication, QWidget, QLabel, QPushButton, QMessageBox

def dialog():

mbox = QMessageBox()

mbox.setText("Your allegiance has been noted")

mbox.setDetailedText("You are now a disciple and subject of the all-knowing Guru")

mbox.setStandardButtons(QMessageBox.Ok | QMessageBox.Cancel)

mbox.exec\_()

if \_\_name\_\_ == "\_\_main\_\_":

app = QApplication(sys.argv)

w = QWidget()

w.resize(300,300)

w.setWindowTitle('Guru99')

label = QLabel(w)

label.setText("Behold the Guru, Guru99")

label.move(100,130)

label.show()

btn = QPushButton(w)

btn.setText('Beheld')

btn.move(110,150)

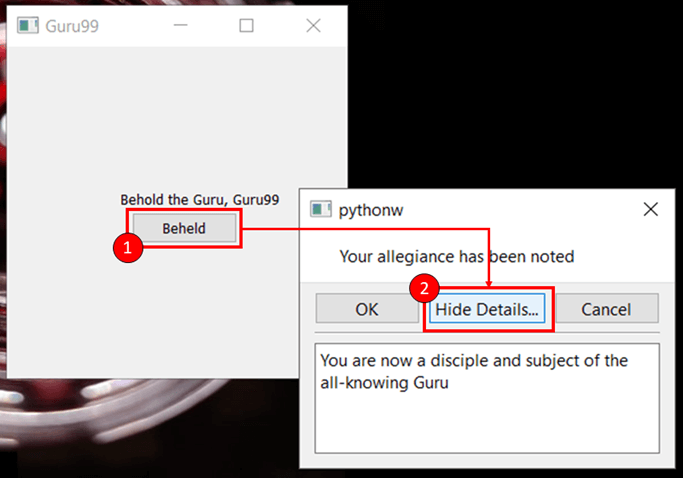
btn.show()

btn.clicked.connect(dialog)

w.show()

sys.exit(app.exec\_())

Save the file as appone.py or anything you like and press F5 to run the program. If you haven't made any mistakes, the IDLE will open a new window with some text and a button as shown below.



1. Once you click on the button in the first window, a new message box will pop open with the text that you had written.
2. You can now click on the Hide Details/Show Details button to toggle the visibility of additional text.

As you can see, since we had not set any window title in the message box, a default title was provided by python itself.

Now that it's working let's take a look at the extra code that you have added to the previous example.

from PyQt5.QtWidgets import QApplication, QWidget, QLabel, QPushButton, QMessageBox

This imports some more widgets that you have used in this example, namely QLabel, QPushButton, and QMessageBox.

def dialog():

mbox = QMessageBox()

mbox.setText("Your allegiance has been noted")

mbox.setDetailedText("You are now a disciple and subject of the all-knowing Guru")

mbox.setStandardButtons(QMessageBox.Ok | QMessageBox.Cancel)

mbox.exec\_()

Here, you have defined a method named dialog which creates a message box widget and sets some text to the buttons and other fields.

The dialog method is called from the main block of the program when a button is pressed in a specific widget (in this case, the btn PushButton). The click event triggered on that button causes this function to execute. Such a function is called a slot in Qt, and you will learn more about **signals** and **slots** in the upcoming paragraphs.

if \_\_name\_\_ == "\_\_main\_\_":

app = QApplication(sys.argv)

w = QWidget()

w.resize(300,300)

w.setWindowTitle('Guru99')

This is the main section of the app and as in the previous example, you start by creating an instance of QApplication followed by a simple widget, i.e. an instance of QWidget.

label = QLabel(w)

btn = QPushButton(w)

You have added two new widgets in this application: QLabel and QPushButton. QLabel is used to print non-editable text or placeholders inside a widget, whereas QPushButton is used to create a clickable button.

The critical thing to notice here is that when you create the label and btn objects, you are passing the window object (w) to the constructors of QLabel and QPushButton. This is how nesting works in PyQt5. To create a widget inside another widget, you pass the reference of the parent widget to the child's constructor.

label.move(100,130)

btn.move(110,150)

move() is used to set the position of a widget with respect to its parent widget. In the first case, the label will be moved 100px from the left and 130px from the top of the window.

Similarly, the button will be placed 110px from the left and 150px from the top of the window. This example is a crude way of achieving layouts and is generally not used in production; it is only included here for learning purposes. Qt supports different layouts that you will see in detail in the upcoming sections of this tutorial.

btn.clicked.connect(dialog)

Finally, this is an example of signals and slots in Qt. In GUI based applications, functions are executed based on the actions performed by the user, like hovering over an element or clicking a button. These actions are called **events**. Recall that the app.exec\_() method transfers control to the Qt **event-**loop. This is what the event loop is there for: to listen for events and perform actions in response.

Whenever an event occurs, like a user clicking a button, the corresponding Qt widget raises a **signal**. These signals can be connected to python functions (like the dialog function in this example) so that the function is executed when a signal is triggered. These functions are called **slots** in Qt lingo.

Subsequently, the basic syntax to trigger a slot function in response to the signal from an event is as follows

widget.signal.connect(slot)

Which means that whenever a **signal** is triggered by a **widget**, the connected **slot** function will be executed. In summary, signals and slots are used by Qt to communicate between objects and facilitate component reusability and interactivity.

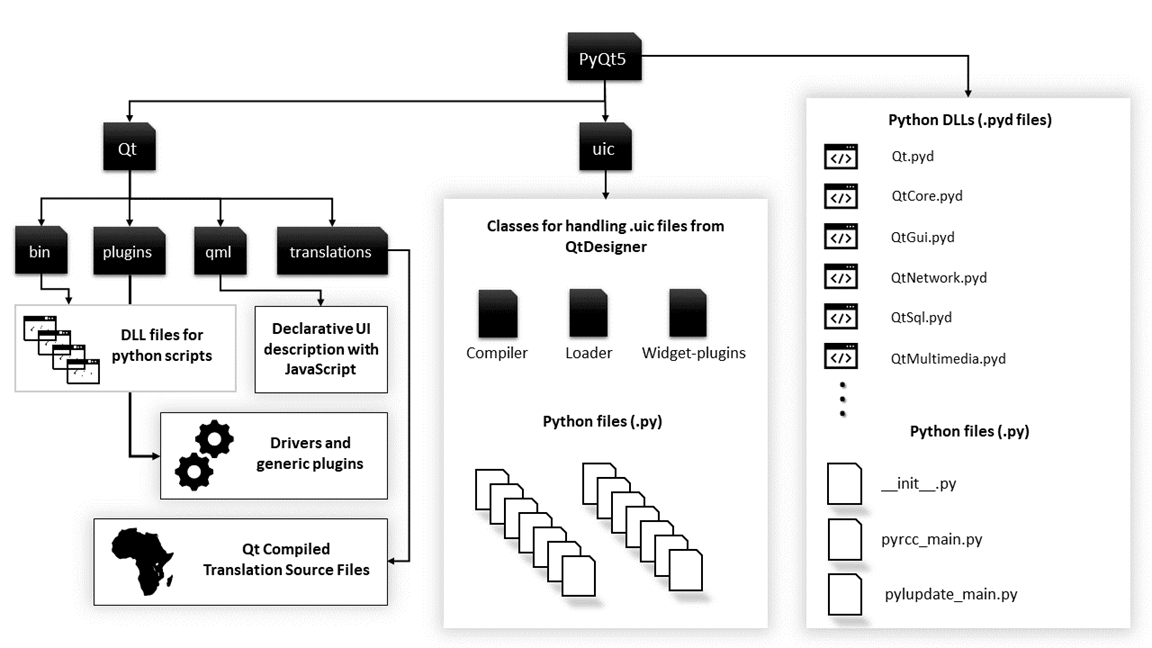
Now that you know how to nest widgets and implement interactions using signals and slots, here is a list of useful widgets and other classes that you can use in your PyQt apps.

## Components and Widgets

There is a large number of widgets available in PyQt for creating GUI apps. However, with PyQt5, there has been a reshuffling of classes into different modules and revisions in the licenses.

Therefore, it's crucial to have a high-level view of the structure of PyQt5. In this section, you will see how PyQt5 is organized internally and learn about the different modules, libraries, and API classes provided by PyQt5.

### PyQt5 Directory Structure



These are the fundamental modules used by Python's Qt binding, specifically PyQt5.

* **Qt**: It combines all the classes/modules mentioned below into a single module. It considerably increases the memory used by the application. However, it's easier to manage the framework by only importing one module.
* **QtCore**: Contains the core non-graphical classes used by other modules. This is where the Qt event loop, signals, and slot-connectivity, etc. are implemented.
* **QtWidgets**: Contains most of the widgets available in Pyqt5.
* **QtGui**: Contains GUI components and extends the QtCore module.
* **QtNetwork**: Contains classes used to implement network programming through Qt. It supports TCP servers, TCP sockets, UDP sockets, SSL handling, network sessions, and DNS lookups.
* **QtMultimedia** provides low-level multimedia functionality.
* **QtSql**: implements database integration for SQL databases. Supports ODBC, MySQL, Oracle, SQLite, and PostgreSQL.

### PyQt5 Widgets

Here is a list of the most frequently used widgets in PyQt5

* **QLineEdit**: This is an input field which allows one line of text to be entered by the user.

line = QLineEdit()

* **QRadioButton**: This is an input field with a selectable button, similar to the radio buttons in html.
* rad = QRadioButton("button title")
* rad.setChecked(True) #to select the button by default.
* **QComboBox**: It is used to display a dropdown menu with a list of selectable items.
* drop = QComboBox(w)
* drop.addItems(["item one", "item two", "item three"])
* **QCheckBox**: Displays a selectable square box in front of the label that is ticked if selected, similar to radio buttons.
* check = QCheckBox("button title")
* **QMenuBar**: it displays a horizontal menu bar at the top of a window. You can only add objects of the QMenu class to this bar. Those QMenu objects can further contain strings, QAction objects or other QMenu objects.
* **QToolBar**: It is a horizontal bar or pane which can be moved within the window. It may contain buttons and other widgets.
* **QTab**: it is used to break down the contents of a window into multiple pages that can be accessed through different tabs on top of the widget. It consists of two sections: the tab bar and the tab page.
* **QScrollBar**: It is used to create scroll bars which allow the user to scroll up and down within a window. It consists of a movable slider, a slider track, and two buttons to scroll the slider up or down.

scroll = QScrollBar()

* **QSplitter**: Splitters are used to separate the contents of a window so that the widgets are grouped properly and do not appear cluttered. QSplitter is one of the primary layout handlers available in PyQt5 and is used to split the content both horizontally and vertically.
* **QDock**: A dock widget is a sub-window with two properties:
  + It can be moved within the main window and
  + It can be docked outside the parent window to another location on the screen.

### Layouts and Themes

In the previous examples, you have only been using the move() and resize() methods to set the positions of widgets in your GUI.

However, PyQt has a robust layout management engine which can be used to create advanced user interfaces for applications. In this section, you will learn about two important classes that are used in Qt to create and manage layouts.

1. **QBoxLayout**
2. **QGridLayout**

### QBoxLayout

QBoxLayout is used to align the child widgets of the layout in a horizontal or vertical row. The two classes of interest that inherit from QBoxLayout are:

* QHBoxLayout: used to line the child widgets horizontally.
* QVBoxLayout: used to line the child widgets vertically.

For example, this is how three buttons aligned with QHBoxLayout will look.

[](https://www.guru99.com/images/1/070919_1308_PyQtTutoria6.png)

import sys

from PyQt5.QtWidgets import \*

if \_\_name\_\_ == "\_\_main\_\_":

app = QApplication([])

w = QWidget()

w.setWindowTitle("Musketeers")

btn1 = QPushButton("Athos")

btn2 = QPushButton("Porthos")

btn3 = QPushButton("Aramis")

hbox = QHBoxLayout(w)

hbox.addWidget(btn1)

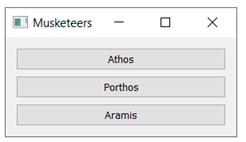
hbox.addWidget(btn2)

hbox.addWidget(btn3)

w.show()

sys.exit(app.exec\_())

And this is how they will look in QVBoxLayout.

[](https://www.guru99.com/images/1/070919_1308_PyQtTutoria7.png)

import sys

from PyQt5.QtWidgets import \*

if \_\_name\_\_ == "\_\_main\_\_":

app = QApplication([])

w = QWidget()

w.setWindowTitle("Musketeers")

btn1 = QPushButton("Athos")

btn2 = QPushButton("Porthos")

btn3 = QPushButton("Aramis")

vb = QVBoxLayout(w)

vb.addWidget(btn1)

vb.addWidget(btn2)

vb.addWidget(btn3)

w.show()

sys.exit(app.exec\_())

The only function that needs any explanation at this point is the addWidget() method. It is used to insert widgets into the HBox or VBox layout. It is also used in other layouts where it takes a different number of parameters as you will see in the next section. The widgets will appear inside the layout in the order in which you insert them.

### QGridLayout

QGridLayout is used to create interfaces in which the widgets are laid out in the form of a grid (like a matrix or 2D array). To insert elements in a grid layout, you can use the matrix representation to define the number of rows and columns in the grid as well as the position of those elements.

For example, to create a 3\*3 grid (i.e., a grid with three rows and three columns), you will write the following code:

Import sys

from PyQt5.QtWidgets import \*

if \_\_name\_\_ == "\_\_main\_\_":

app = QApplication([])

w = QWidget()

grid = QGridLayout(w)

for i in range(3):

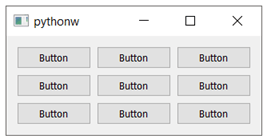
for j in range(3):

grid.addWidget(QPushButton("Button"),i,j)

w.show()

sys.exit(app.exec\_())

This will be the output:

[](https://www.guru99.com/images/1/070919_1308_PyQtTutoria8.png)

The addWidget() method In the grid layout takes these arguments:

* The widget object that you want to add to the grid
* The x-coordinate of the object
* The y-coordinate of the object
* The row-span (default =0)
* The col-span (default=0)

To understand it better, you can manually insert each widget as shown below

import sys

from PyQt5.QtWidgets import \*

if \_\_name\_\_ == "\_\_main\_\_":

app = QApplication([])

w = QWidget()

grid = QGridLayout(w)

grid.addWidget(QPushButton("Button one"),0,0)

grid.addWidget(QPushButton("Button two"),0,1)

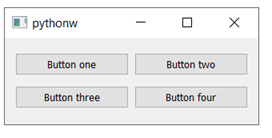
grid.addWidget(QPushButton("Button three"),1,0)

grid.addWidget(QPushButton("Button four"),1,1)

w.show()

sys.exit(app.exec\_())

This is what the grid will look like:

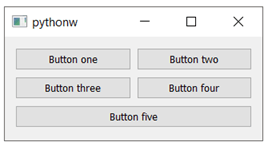
[](https://www.guru99.com/images/1/070919_1308_PyQtTutoria9.png)

You can also pass the rowspan and colspan parameters to addWidget() in order to span more than one row or column.

For example,

grid.addWidget(QPushButton("Button five"),2,0,1,0)

This will create a button that stretches over both columns.

[](https://www.guru99.com/images/1/070919_1308_PyQtTutoria10.png)

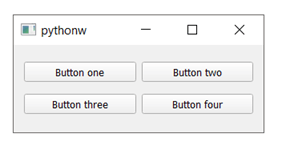
## Themes

PyQt5 comes with some built-in themes that you can use in your apps. The **setStyle()** method called on the QApplication instance is used to set a particular theme to your application.

For example, adding the following line of code will change the theme of your application from default to Fusion

app.setStyle("Fusion")

This is how the previous example will look in Fusion Theme

[](https://www.guru99.com/images/1/070919_1308_PyQtTutoria11.png)

Another useful function for theming your apps is the setPalette() method. Here's the code for changing the colour of different widgets using setPalette().

import sys

from PyQt5.QtCore import Qt

from PyQt5.QtWidgets import \*

from PyQt5.QtGui import QPalette

if \_\_name\_\_ == "\_\_main\_\_":

app = QApplication([])

app.setStyle("Fusion")

qp = QPalette()

qp.setColor(QPalette.ButtonText, Qt.black)

qp.setColor(QPalette.Window, Qt.black)

qp.setColor(QPalette.Button, Qt.gray)

app.setPalette(qp)

w = QWidget()

grid = QGridLayout(w)

grid.addWidget(QPushButton("Button one"),0,0)

grid.addWidget(QPushButton("Button two"),0,1)

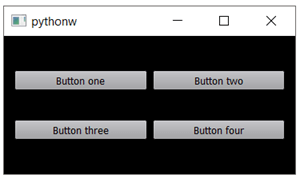
grid.addWidget(QPushButton("Button three"),1,0)

grid.addWidget(QPushButton("Button four"),1,1)

w.show()

sys.exit(app.exec\_())

Here's the result.

[](https://www.guru99.com/images/1/070919_1308_PyQtTutoria12.png)

To use the setPalette() method, you first need to define a palette. This is done by creating an object of the QPalette class.

qp = QPalette()

Notice that the QPalette class belongs to the QtGui module and you will need to import it for this to work. Once you've created the QPalette object, use the setColor() method to pass the name of a widget whose colour you want to change and the colour you want to be set.

qp.setColor(QPalette.Window, Qt.black)

This will change the colour of the window to black. After you have defined your color-scheme, use the setPalette() function to apply the palette to your application.

app.setPalette(qp)

That is all you need to do if you want to create some basic themes for your app. PyQt also allows you to use stylesheets to define the look of your widgets. If you are familiar with CSS, you can easily define advanced styles for your app using Qt Style Sheets.

## Summary

* PyQt is the python binding for the C++ UI framework, Qt.
* PyQt4 and PyQt5 are the two major versions developed by Riverbank Computing.
* The main modules of the PyQt framework are:
  1. Qt
  2. QtCore
  3. QtWidgets
  4. QtGui
  5. QtSql
  6. QtNetwork
* PyQt supports various widgets like:
  1. Buttons
  2. Text labels
  3. Text fields
  4. Radio buttons and Checkboxes
  5. Toolbars and Menu bars
  6. Webkit
  7. Tabs
  8. Docks
* In PyQt, interactivity is implemented using **signals** and **slots.**
* An **event**is an action performed by a user in the GUI (like clicking a button).
* A **signal**is raised by the corresponding widget when an event occurs on it.
* A **slot**is a function that is connected to the signal and executes when the signal is raised.
* PyQt has a robust layout engine and supports advanced layout design and management. These are two frequently used layout schemes in PyQt:
  1. Box Layout
  2. Grid Layout
* PyQt allows you to create custom themes for GUI application and has inbuilt support for style sheets.
* PyQt can be used to create user interfaces as well as standalone applications.

# Multithreading in Python with Global Interpreter Lock (GIL) Example

The python programming language allows you to use multiprocessing or multithreading.In this tutorial, you will learn how to write multithreaded applications in Python.

## What is a Thread?

A thread is a unit of exection on concurrent programming. Multithreading is a technique which allows a CPU to execute many tasks of one process at the same time. These threads can execute individually while sharing their process resources.

## What is a Process?

A process is basically the program in execution. When you start an application in your computer (like a browser or text editor), the operating system creates a **process.**

## What is Multithreading?

Multithreading is a technique which allows a CPU to execute multiple threads at the same time. These threads can execute individually while sharing their process resources.

## What is Multiprocessing?

Multiprocessing allows you to run multiple unrelated processes simultaneously. These processes do not share their resources and communicate through IPC.

## Python Multithreading vs Multiprocessing

To understand processes and threads, consider this scenario: An .exe file on your computer is a program. When you open it, the OS loads it into memory, and the CPU executes it. The instance of the program which is now running is called the process.

Every process will have 2 fundamental components:

* The Code
* The Data

Now, a process can contain one or more sub-parts called **threads.**Thisdepends on the OS architecture,.You can think about a thread as a section of the process which can be executed separately by the operating system.

In other words, it is a stream of instructions which can be run independently by the OS. Threads within a single process share the data of that process and are designed to work together for facilitating parallelism.

In this tutorial, you will learn,

* [What is a Thread?](https://www.guru99.com/python-multithreading-gil-example.html#1)
* [What is a Process?](https://www.guru99.com/python-multithreading-gil-example.html#2)
* [What is Multithreading?](https://www.guru99.com/python-multithreading-gil-example.html#3)
* [What is Multiprocessing?](https://www.guru99.com/python-multithreading-gil-example.html#4)
* [Python Multithreading vs Multiprocessing](https://www.guru99.com/python-multithreading-gil-example.html#5)
* [Why use Multithreading?](https://www.guru99.com/python-multithreading-gil-example.html#6)
* [Python MultiThreading](https://www.guru99.com/python-multithreading-gil-example.html#7)
* [The Thread and Threading modules](https://www.guru99.com/python-multithreading-gil-example.html#8)
* [The Thread Module](https://www.guru99.com/python-multithreading-gil-example.html#9)
* [The Threading Module](https://www.guru99.com/python-multithreading-gil-example.html#10)
* [Deadlocks and Race conditions](https://www.guru99.com/python-multithreading-gil-example.html#11)
* [Synchronizing threads](https://www.guru99.com/python-multithreading-gil-example.html#12)
* [What is GIL?](https://www.guru99.com/python-multithreading-gil-example.html#13)
* [Why was GIL needed?](https://www.guru99.com/python-multithreading-gil-example.html#14)

## Why use Multithreading?

Multithreading allows you to break down an application into multiple sub-tasks and run these tasks simultaneously. If you use multithreading properly, your application speed, performance, and rendering can all be improved.

## Python MultiThreading

Python supports constructs for both multiprocessing as well as multithreading. In this tutorial, you will primarily be focusing on implementing **multithreaded** applications with python. There are two main modules which can be used to handle threads in Python:

1. The **thread** module, and
2. The **threading** module

However, in python, there is also something called a global interpreter lock (GIL). It doesn't allow for much performance gain and may even **reduce** the performance of some multithreaded applications. You will learn all about it in the upcoming sections of this tutorial.

## The Thread and Threading modules

The two modules that you will learn about in this tutorial are the **thread module** and the **threading module**.

However, the thread module has long been deprecated. Starting with Python 3, it has been designated as obsolete and is only accessible as **\_\_thread** for backward compatibility.

You should use the higher-level **threading** module for applications which you intend to deploy. The thread module has only been covered here for educational purposes.

## The Thread Module

The syntax to create a new thread using this module is as follows:

thread.start\_new\_thread(function\_name, arguments)

Alright, now you have covered the basic theory to start coding. So, open your IDLE or a notepad and type in the following:

import time

import \_thread

def thread\_test(name, wait):

i = 0

while i <= 3:

time.sleep(wait)

print("Running %s\n" %name)

i = i + 1

print("%s has finished execution" %name)

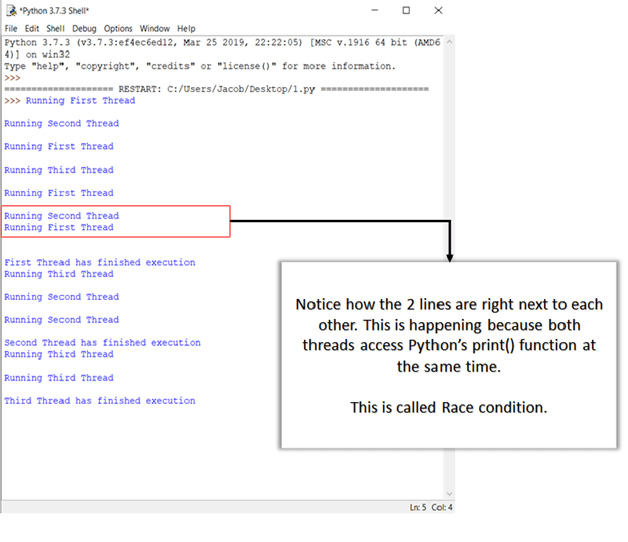
if \_\_name\_\_ == "\_\_main\_\_":

\_thread.start\_new\_thread(thread\_test, ("First Thread", 1))

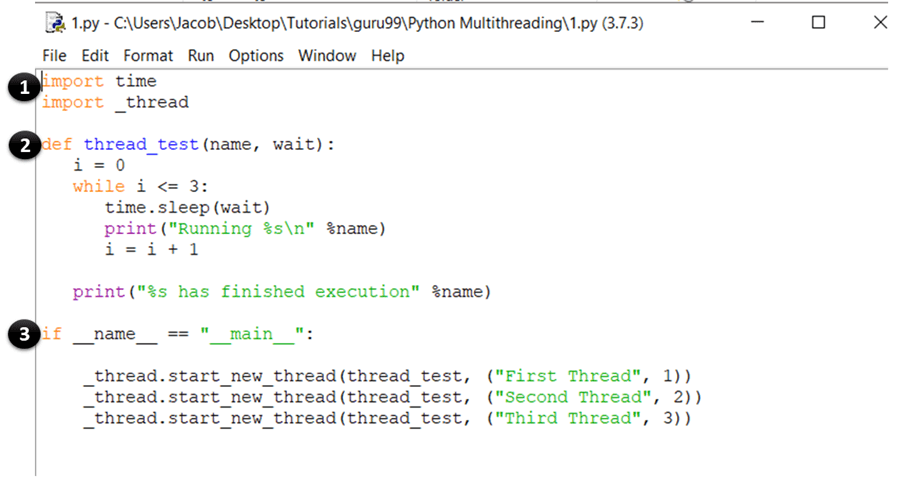
\_thread.start\_new\_thread(thread\_test, ("Second Thread", 2))

\_thread.start\_new\_thread(thread\_test, ("Third Thread", 3))

Save the file and hit F5 to run the program. If everything was done correctly, this is the output that you should see:



You will learn more about race conditions and how to handle them in the upcoming sections



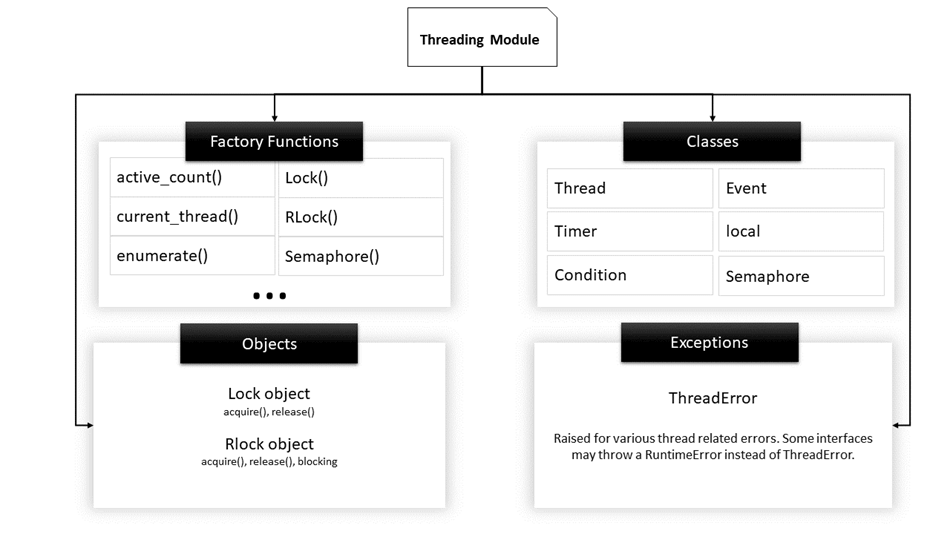
**CODE EXPLANATION**

1. These statements import the time and thread module which are used to handle the execution and delaying of the Python threads.
2. Here, you have defined a function called **thread\_test,** which will be called by the **start\_new\_thread**method. The function runs a while loop for four iterations and prints the name of the thread which called it. Once the iteration is complete, it prints a message saying that the thread has finished execution.
3. This is the main section of your program. Here, you simply call the **start\_new\_thread**method with the **thread\_test**function as an argument.

This will create a new thread for the function you pass as argument and start executing it. Note that you can replace this (thread**\_**test) with any other function which you want to run as a thread.

## The Threading Module

This module is the high-level implementation of threading in python and the de facto standard for managing multithreaded applications. It provides a wide range of features when compared to the thread module.



Structure of Threading module

Here is a list of some useful functions defined in this module:

|  |  |
| --- | --- |
| **Function Name** | **Description** |
| **activeCount()** | Returns the count of **Thread** objects which are still alive |
| **currentThread()** | Returns the current object of the Thread class. |
| **enumerate()** | Lists all active Thread objects. |
| **isDaemon()** | Returns true if the thread is a daemon. |
| **isAlive()** | Returns true if the thread is still alive. |
|  | **Thread Class methods** |
| **start()** | Starts the activity of a thread. It must be called only once for each thread because it will throw a runtime error if called multiple times. |
| **run()** | This method denotes the activity of a thread and can be overridden by a class that extends the Thread class. |
| **join()** | It blocks the execution of other code until the thread on which the join() method was called gets terminated. |

### Backstory: The Thread Class

Before you start coding multithreaded programs using the threading module, it is crucial to understand about the Thread class.The thread class is the primary class which defines the template and the operations of a thread in python.

The most common way to create a multithreaded python application is to declare a class which extends the Thread class and overrides it's run() method.

The Thread class, in summary, signifies a code sequence that runs in a separate **thread** of control.

So, when writing a multithreaded app, you will do the following:

1. define a class which extends the Thread class
2. Override the **\_\_init\_\_** constructor
3. Override the **run()** method

Once a thread object has been made, the **start()** method can be used to begin the execution of this activity and the **join()** method can be used to block all other code till the current activity finishes.

Now, let's try using the threading module to implement your previous example. Again, fire up your IDLE and type in the following:

import time

import threading

class threadtester (threading.Thread):

def \_\_init\_\_(self, id, name, i):

threading.Thread.\_\_init\_\_(self)

self.id = id

self.name = name

self.i = i

def run(self):

thread\_test(self.name, self.i, 5)

print ("%s has finished execution " %self.name)

def thread\_test(name, wait, i):

while i:

time.sleep(wait)

print ("Running %s \n" %name)

i = i - 1

if \_\_name\_\_=="\_\_main\_\_":

thread1 = threadtester(1, "First Thread", 1)

thread2 = threadtester(2, "Second Thread", 2)

thread3 = threadtester(3, "Third Thread", 3)

thread1.start()

thread2.start()

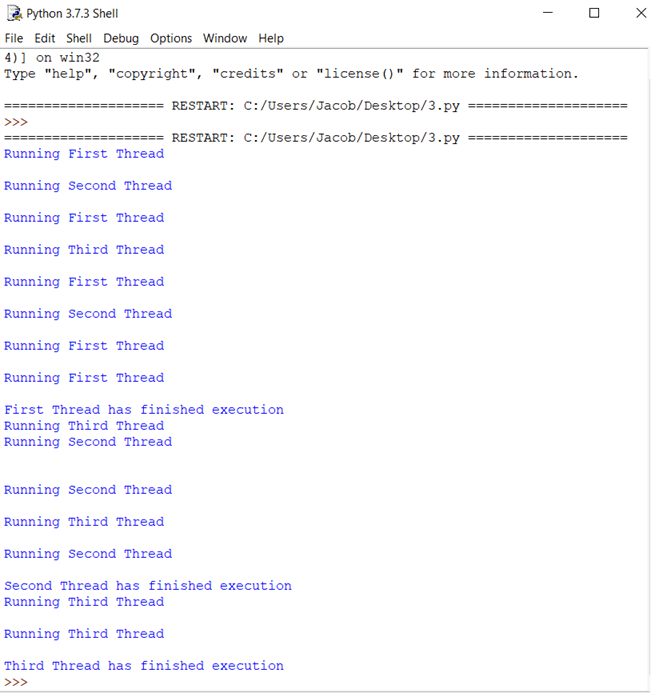
thread3.start()

thread1.join()

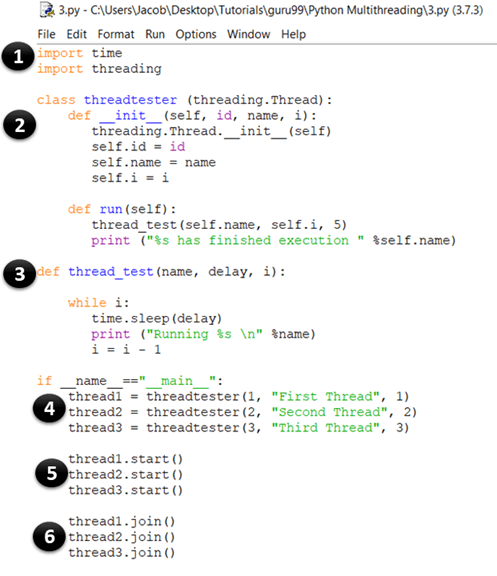
thread2.join()

thread3.join()

This will be the output when you execute the above code:



**CODE EXPLANATION**

[](https://www.guru99.com/images/1/080219_0505_Multithread5.png)

1. This part is the same as our previous example. Here, you import the time and thread module which are used to handle the execution and delays of the Python threads.
2. In this bit, you are creating a class called threadtester, which inherits or extends the **Thread** class of the threading module. This is one of the most common ways of creating threads in python. However, you should only override the constructor and the **run()** method in your app. As you can see in the above code sample, the **\_\_init\_\_** method (constructor) has been overridden.

Similarly, you have also overridden the **run()**method. It contains the code that you want to execute inside a thread. In this example, you have called the thread\_test() function.

1. This is the thread\_test() method which takes the value of **i**as an argument, decreases it by 1 at each iterationand loops through the rest of the code until i becomes 0. In each iteration, it prints the name of the currently executing thread and sleeps for wait seconds (which is also taken as an argument).
2. thread1 = threadtester(1, "First Thread", 1)

Here, we are creating a thread and passing the three parameters that we declared in \_\_init\_\_. The first parameter is the id of the thread, the second parameter is the thread's name, and the third parameter is the counter, which determines how many times the while loop should run.

1. thread2.start()

The start method is used to start the execution of a thread. Internally, the start() function calls the run() method of your class.

1. thread3.join()

The join() method blocks the execution of other code and waits until the thread on which it was called finishes.

As you already know, the threads which are in the same process have access to the memory and data of that process. As a result, if more than one thread tries to change or access the data simultaneously, errors may creep in.

In the next section, you will see the different kinds of complications that can show up when threads access data and critical-section without checking for existing access transactions.

## Deadlocks and Race conditions

Before learning about deadlocks and race conditions, it'll be helpful to understand a few basic definitions related to concurrent programming:

* Critical Section

It is a fragment of code that accesses or modifies shared variables and must be performed as an atomic transaction.

* Context Switch

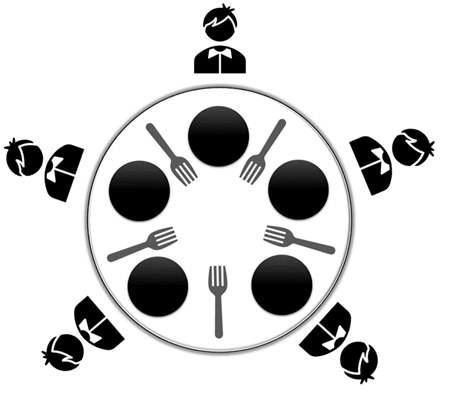
It is the process that a CPU follows to store the state of a thread before changing from one task to another so that it can be resumed from the same point later.

### Deadlocks

Deadlocks are the most feared issue that developers face when writing concurrent/multithreaded applications in python. The best way to understand deadlocks is by using the classic computer science example problem known as the **Dining Philosophers Problem.**

The problem statement for dining philosophers is as follows:

Five philosophers are seated on a round table with five plates of spaghetti (a type of pasta) and five forks, as shown in the diagram.

[](https://www.guru99.com/images/1/080219_0505_Multithread6.png)Dining Philosophers Problem

At any given time, a philosopher must either be eating or thinking.

Moreover, a philosopher must take the two forks adjacent to him (i.e., the left and right forks) before he can eat the spaghetti. The problem of deadlock occurs when all five philosophers pick up their right forks simultaneously.

Since each of the philosophers has one fork, they will all wait for the others to put their fork down. As a result, none of them will be able to eat spaghetti.

Similarly, in a concurrent system, a deadlock occurs when different threads or processes (philosophers) try to acquire the shared system resources (forks) at the same time. As a result, none of the processes get a chance to execute as they are waiting for another resource held by some other process.

### Race Conditions

A race condition is an unwanted state of a program which occurs when a system performs two or more operations simultaneously. For example, consider this simple for loop:

i=0; # a global variable

for x in range(100):

print(i)

i+=1;

If you create **n**number of threads which run this code at once, you cannot determine the value of i (which is shared by the threads) when the program finishes execution. This is because in a real multithreading environment, the threads can overlap, and the value of i which was retrieved and modified by a thread can change in between when some other thread accesses it.

These are the two main classes of problems that can occur in a multithreaded or distributed python application. In the next section, you will learn how to overcome this problem by synchronizing threads.

## Synchronizing threads

To deal with race conditions, deadlocks, and other thread-based issues, the threading module provides the **Lock**object. The idea is that when a thread wants access to a specific resource, it acquires a lock for that resource. Once a thread locks a particular resource, no other thread can access it until the lock is released. As a result, the changes to the resource will be atomic, and race conditions will be averted.

A lock is a low-level synchronization primitive implemented by the **\_\_thread**module. At any given time, a lock can be in one of 2 states: **locked**or **unlocked.**It supports two methods:

1. **acquire()**

When the lock-state is unlocked, calling the acquire() method will change the state to locked and return. However, If the state is locked, the call to acquire() is blocked until the release() method is called by some other thread.

1. **release()**

The release() method is used to set the state to unlocked, i.e., to release a lock. It can be called by any thread, not necessarily the one that acquired the lock.

Here's an example of using locks in your apps. Fire up your IDLE and type the following:

import threading

lock = threading.Lock()

def first\_function():

for i in range(5):

lock.acquire()

print ('lock acquired')

print ('Executing the first funcion')

lock.release()

def second\_function():

for i in range(5):

lock.acquire()

print ('lock acquired')

print ('Executing the second funcion')

lock.release()

if \_\_name\_\_=="\_\_main\_\_":

thread\_one = threading.Thread(target=first\_function)

thread\_two = threading.Thread(target=second\_function)

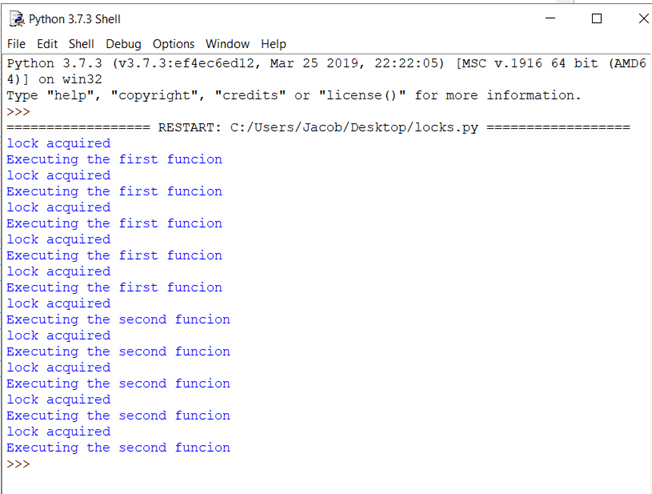
thread\_one.start()

thread\_two.start()

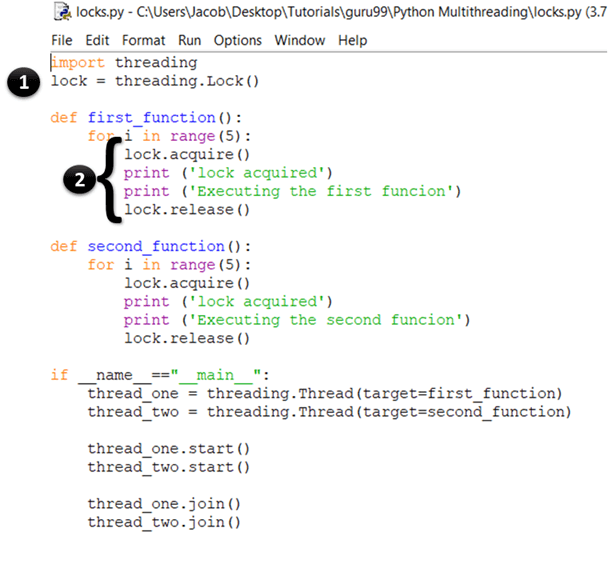
thread\_one.join()

thread\_two.join()

Now, hit F5. You should see an output like this:



**CODE EXPLANATION**



1. Here, you are simply creating a new lock by calling the **threading.Lock()** factory function. Internally, Lock() returns an instance of the most effective concrete Lock class that is maintained by the platform.
2. In the first statement, you acquire the lock by calling the acquire() method. When the lock has been granted, you print**"lock acquired"**to the console. Once all the code that you want the thread to run has finished execution, you release the lock by calling the release() method.

The theory is fine, but how do you know that the lock really worked? If you look at the output, you will see that each of the print statements is printing exactly one line at a time. Recall that, in an earlier example, the outputs from print where haphazard because multiple threads were accessing the print() method at the same time. Here, the print function is called only after the lock is acquired. So, the outputs are displayed one at a time and line by line.

Apart from locks, python also supports some other mechanisms to handle thread synchronization as listed below:

1. RLocks
2. Semaphores
3. Conditions
4. Events, and
5. Barriers

### Global Interpreter Lock (and how to deal with it)

Before getting into the details of python's GIL, let's define a few terms that will be useful in understanding the upcoming section:

1. CPU-bound code: this refers to any piece of code which will be directly executed by the CPU.
2. I/O-bound code: this can be any code that accesses the file system thru' the OS
3. CPython: it is the reference **implementation** of Python and can be described as the interpreter written in C and Python (programming language).

## What is GIL?

A lock can be used to make sure that only one thread has access to a particular resource at a given time.

One of the features of Python is that it uses a global lock on each interpreter process, which means that every process treats the python interpreter itself as a resource.

For example, suppose you have written a python program which uses two threads to perform both CPU and 'I/O' operations. When you execute this program, this is what happens:

1. The python interpreter creates a new process and spawns the threads
2. When thread-1 starts running, it will first acquire the GIL and lock it.
3. If thread-2 wants to execute now, it will have to wait for the GIL to be released even if another processor is free.
4. Now, suppose thread-1 is waiting for an I/O operation. At this time, it will release the GIL, and thread-2 will acquire it.
5. After completing the I/O ops, if thread-1 wants to execute now, it will again have to wait for the GIL to be released by thread-2.

Due to this, only one thread can access the interpreter at any time, meaning that there will be only one thread executing python code at a given point of time.

This is alright in a single-core processor because it would be using time slicing (see the first section of this tutorial) to handle the threads. However, in case of multi-core processors, a CPU-bound function executing on multiple threads will have a considerable impact on the program's efficiency since it won't actually be using all the available cores at the same time.

## Why was GIL needed?

The CPython garbage collector uses an efficient memory management technique known as reference counting. Here's how it works: Every object in python has a reference count, which is increased when it is assigned to a new variable name or added to a container (like tuples, lists, etc.). Likewise, the reference count is decreased when the reference goes out of scope or when the del statement is called. When the reference count of an object reaches 0, it is garbage collected, and the allotted memory is freed.

But the problem is that the reference count variable is prone to race conditions like any other global variable. To solve this problem, the developers of python decided to use the global interpreter lock. The other option was to add a lock to each object which would have resulted in deadlocks and increased overhead from acquire() and release() calls.

Therefore, GIL is a significant restriction for multithreaded python programs running heavy CPU-bound operations (effectively making them single-threaded). If you want to make use of multiple CPU cores in your application, use the **multiprocessing** module instead.

#### Summary

* Python supports 2 modules for multithreading:
  1. **\_\_thread** module: It provides a low-level implementation for threading and is obsolete.
  2. **threading module**: It provides a high-level implementation for multithreading and is the current standard.
* To create a thread using the threading module, you must do the following:
  1. Create a class which extends the **Thread** class.
  2. Override its constructor (\_\_init\_\_).
  3. Override its **run()** method.
  4. Create an object of this class.
* A thread can be executed by calling the **start()** method.
* The **join()** method can be used to block other threads until this thread (the one on which join was called) finishes execution.
* A race condition occurs when multiple threads access or modify a shared resource at the same time.
* It can be avoided by Synchronizing threads.
* Python supports 6 ways to synchronize threads:
  1. Locks
  2. RLocks
  3. Semaphores
  4. Conditions
  5. Events, and
  6. Barriers
* Locks allow only a particular thread which has acquired the lock to enter the critical section.
* A Lock has 2 primary methods:
  1. **acquire()**: It sets the lock state to **locked.**If called on a locked object, it blocks until the resource is free.
  2. **release()**: It sets the lock state to **unlocked**and returns. If called on an unlocked object, it returns false.
* The global interpreter lock is a mechanism through which only 1 CPython interpreter process can execute at a time.
* It was used to facilitate the reference counting functionality of CPythons's garbage collector.
* To make Python apps with heavy CPU-bound operations, you should use the multiprocessing module.

# Python Lambda Functions with EXAMPLES

## What is Lambda?

Lambdas, also known as anonymous functions, are small, restricted functions which do not need a name (i.e., an identifier). Lambda functions were first introduced to the field of mathematics by Alonzo Church in the 1930s.

Today, many modern programming languages like Java, Python, C#, and C++ support lambda functions to add functionality to the languages.

In this Lambda tutorial, you will learn:

* [What is Lambda?](https://www.guru99.com/python-lambda-function.html#1)
* [Lambdas in Python](https://www.guru99.com/python-lambda-function.html#2)
* [Syntax and Examples](https://www.guru99.com/python-lambda-function.html#3)
* [Using lambdas with Python built-ins](https://www.guru99.com/python-lambda-function.html#4)
* [lambdas in filter()](https://www.guru99.com/python-lambda-function.html#5)
* [lambdas in map()](https://www.guru99.com/python-lambda-function.html#6)
* [lambdas in reduce()](https://www.guru99.com/python-lambda-function.html#7)
* [Why (and why not) use lambda functions?](https://www.guru99.com/python-lambda-function.html#8)
* [Lambdas vs. Regular functions](https://www.guru99.com/python-lambda-function.html#9)

## Lambdas in Python

In Python, lambda expressions (or lambda forms) are utilized to construct anonymous functions. To do so, you will use the **lambda**keyword (just as you use **def**to define normal functions).

Every anonymous function you define in Python will have 3 essential parts:

* The lambda keyword.
* The parameters (or bound variables), and
* The function body.

A lambda function can have any number of parameters, but the function body can only contain **one** expression.

Moreover, a lambda is written in a single line of code and can also be invoked immediately. You will see all this in action in the upcoming examples.

## Syntax and Examples

The formal syntax to write a lambda function is as given below:

lambda p1, p2: expression

Here, p1 and p2 are the parameters which are passed to the lambda function. You can add as many or few parameters as you need.

However, notice that we do not use brackets around the parameters as we do with regular functions. The last part (expression) is any valid python expression that operates on the parameters you provide to the function.

### Example 1

Now that you know about lambdas let's try it with an example. So, open your IDLE and type in the following:

adder = lambda x, y: x + y

print (adder (1, 2))

**Here is the output:**

3

### Code Explanation

Here, we define a variable that will hold the result returned by the lambda function.

**1.** The lambda keyword used to define an anonymous function.

**2.** x and y are the parameters that we pass to the lambda function.

**3.** This is the body of the function, which adds the 2 parameters we passed. Notice that it is a single expression. You cannot write multiple statements in the body of a lambda function.

**4.** We call the function and print the returned value.

### Example 2

That was a basic example to understand the fundamentals and syntax of lambda. Let's now try to print out a lambda and see the result. Again, open your IDLE and type in the following:

#What a lambda returns

string='some kind of a useless lambda'

print(lambda string : print(string))

Now save your file and hit F5 to run the program. This is the output you should get.

**Output:**

<function <lambda> at 0x00000185C3BF81E0>

What's happening here? Let's look at the code to understand further.

Code Explanation

1. Here, we define a string that you'll pass as a parameter to the lambda.
2. We declare a lambda that calls a print statement and prints the result.

But why doesn't the program print the string we pass? This is because the lambda itself returns a function object. In this example, the lambda is not being **called** by the print function but simply **returning** the function object and the memory location where it is stored. That's what gets printed at the console.

### Example 3

However, if you write a program like this:

#What a lambda returns #2

x="some kind of a useless lambda"

(lambda x : print(x))(x)

And run it by hitting F5, you'll see an output like this.

**Output:**

some kind of a useless lambda

Now, the lambda is being called, and the string we pass gets printed at the console. But what is that weird syntax, and why is the lambda definition covered in brackets? Let's understand that now.

Code Explanation

1. Here is the same string we defined in the previous example.
2. In this part, we are defining a lambda and calling it immediately by passing the string as an argument. This is something called an IIFE, and you'll learn more about it in the upcoming sections of this tutorial.

### Example 4

Let's look at a final example to understand how lambdas and regular functions are executed. So, open your IDLE and in a new file, type in the following:

#A REGULAR FUNCTION

def guru( funct, \*args ):

funct( \*args )

def printer\_one( arg ):

return print (arg)

def printer\_two( arg ):

print(arg)

#CALL A REGULAR FUNCTION

guru( printer\_one, 'printer 1 REGULAR CALL' )

guru( printer\_two, 'printer 2 REGULAR CALL \n' )

#CALL A REGULAR FUNCTION THRU A LAMBDA

guru(lambda: printer\_one('printer 1 LAMBDA CALL'))

guru(lambda: printer\_two('printer 2 LAMBDA CALL'))

Now, save the file and hit F5 to run the program. If you didn't make any mistakes, the output should be something like this.

**Output:**

printer 1 REGULAR CALL

printer 2 REGULAR CALL

printer 1 LAMBDA CALL

printer 2 LAMBDA CALL

Code Explanation

1. A function called guru that takes another function as the first parameter and any other arguments following it.
2. printer\_one is a simple function which prints the parameter passed to it and returns it.
3. printer\_two is similar to printer\_one but without the return statement.
4. In this part, we are calling the guru function and passing the printer functions and a string as parameters.
5. This is the syntax to achieve the fourth step (i.e., calling the guru function) but using lambdas.

In the next section, you will learn how to use lambda functions with **map(),** **reduce(),** and **filter()** in Python.

## Using lambdas with Python built-ins

Lambda functions provide an elegant and powerful way to perform operations using built-in methods in Python. It is possible because lambdas can be invoked immediately and passed as an argument to these functions.

### IIFE in Python Lambda

**IIFE** stands for **immediately invoked function execution.**It means that a lambda function is callable as soon as it is defined. Let's understand this with an example; fire up your IDLE and type in the following:

**Here is the output and code explanation:**

This ability of lambdas to be invoked immediately allows you to use them inside functions like map() and reduce(). It is useful because you may not want to use these functions again.

## lambdas in filter()

The filter function is used to select some particular elements from a sequence of elements. The sequence can be any iterator like lists, sets, tuples, etc.

The elements which will be selected is based on some pre-defined constraint. It takes 2 parameters:

* A function that defines the filtering constraint
* A sequence (any iterator like lists, tuples, etc.)

For example,

sequences = [10,2,8,7,5,4,3,11,0, 1]

filtered\_result = filter (lambda x: x > 4, sequences)

print(list(filtered\_result))

**Here's the output:**

[10, 8, 7, 5, 11]

**Code Explanation:**

**1.** In the first statement, we define a list called sequences which contains some numbers.

**2.** Here, we declare a variable called filtered\_result, which will store the filtered values returned by the filter() function.

**3.** A lambda function which runs on each element of the list and returns true if it is greater than 4.

**4.**Print the result returned by the filter function.

## lambdas in map()

the map function is used to apply a particular operation to every element in a sequence. Like filter(), it also takes 2 parameters:

1. A function that defines the op to perform on the elements
2. One or more sequences

For example, here is a program that prints the squares of numbers in a given list:

sequences = [10,2,8,7,5,4,3,11,0, 1]

filtered\_result = map (lambda x: x\*x, sequences)

print(list(filtered\_result))

**Output:**

[100, 4, 64, 49, 25, 16, 121, 0, 1]

[KR1]

Code Explanation:

1. Here, we define a list called sequences which contains some numbers.
2. We declare a variable called filtered\_result which will store the mapped values
3. A lambda function which runs on each element of the list and returns the square of that number.
4. Print the result returned by the map function.

## lambdas in reduce[vV2][J3]()

The reduce function, like map(), is used to apply an operation to every element in a sequence. However, it differs from the map in its working. These are the steps followed by the reduce() function to compute an output:

**Step 1)**Perform the defined operation on the first 2 elements of the sequence.

**Step 2)**Save this result

**Step 3)**Perform the operation with the saved result and the next element in the sequence.

**Step 4)**Repeat until no more elements are left.

It also takes two parameters:

1. A function that defines the operation to be performed
2. A sequence (any iterator like lists, tuples, etc.)

For example, here is a program that returns the product of all elements in a list:

from functools import reduce

sequences = [1,2,3,4,5]

product = reduce (lambda x, y: x\*y, sequences)

print(product)

**Here is the output:**

120

Code Explanation:

1. Import reduce from the functools module
2. Here, we define a list called sequences which contains some numbers.
3. We declare a variable called product which will store the reduced value
4. A lambda function that runs on each element of the list. It will return the product of that number as per the previous result.
5. Print the result returned by the reduce function.

## Why (and why not) use lambda functions?

As you will see in the next section, lambdas are treated the same as regular functions at the interpreter level. In a way, you could say that lambdas provide compact syntax for writing functions which return a single expression.

However, you should know when it is a good idea to use lambdas and when to avoid them. In this section, you will learn some of the design principles used by python developers when writing lambdas.

One of the most common use cases for lambdas is in functional programming as Python supports a paradigm (or style) of programming known as functional programming.

It allows you to provide a function as a parameter to another function (for example, in map, filter, etc.). In such cases, using lambdas offer an elegant way to create a one-time function and pass it as the parameter.

### When should you not use Lambda?

You should never write complicated lambda functions in a production environment. It will be very difficult for coders who maintain your code to decrypt it. If you find yourself making complex one-liner expressions, it would be a much superior practice to define a proper function. As a best practice, you need to remember that simple code is always better than complex code.

## Lambdas vs. Regular functions

As previously stated, lambdas are[vV4][J5] just functions which do not have an identifier bound to them. In simpler words, they are functions with no names (hence, anonymous). Here is a table to illustrate the difference between lambdas and regular functions in python.

**Lambdas**

**Regular Functions**

Syntax:

lambda x : x + x

Syntax:

def (x) :

return x + x

Lambda functions can only have one expression in their body.

Regular functions can have multiple expressions and statements in their body.

Lambdas do not have a name associated with them. That's why they are also known as anonymous functions.

Regular functions must have a name and signature.

Lambdas do not contain a return statement because the body is automatically returned.

Functions which need to return value should include a return statement.

### Explanation of the differences?

The primary difference between a lambda and a regular function is that the lambda function evaluates only a single expression and yields a function object. Consequently, we can name the result of the lambda function and use it in our program as we did in the previous example.

A regular function for the above example would look like this:

def adder (x, y):

return x + y

print (adder (1, 2))

Here, we have to define a **name** for the function which **returns**the result when we **call** it. A lambda function doesn't contain a return statement because it will have only a single expression which is always returned by default. You don't even have to assign a lambda either as it can be immediately invoked (see the next section). As you will see in the following example, lambdas become particularly powerful when we use them with Python's built-in functions.

However, you may still be wondering how lambdas are any different from a function that returns a single expression (like the one above). At the interpreter level, there is not much difference. It may sound surprising, but any lambda function that you define in Python is treated as a normal function by the interpreter.

As you can see in the diagram, the two definitions are handled in the same way by the python interpreter when converted to bytecode. Now, you cannot name a function **lambda** because it is reserved by Python, but any other function name will yield the same bytecode[KR6].

### Summary

* Lambdas, also known as anonymous functions, are small, restricted functions which do not need a name (i.e., an identifier).
* Every lambda function in Python has 3 essential parts:
* The lambda keyword.
* The parameters (or bound variables), and
* The function body.
* The syntax for writing a lambda is: lambda parameter: expression
* Lambdas can have any number of parameters, but they are not enclosed in braces
* A lambda can have only 1 expression in its function body, which is returned by default.
* At the bytecode level, there is not much difference between how lambdas and regular functions are handled by the interpreter.
* Lambdas support IIFE thru this syntax: (lambda parameter: expression)(argument)
* Lambdas are commonly used with the following python built-ins:
* Filter: filter (lambda parameter: expression, iterable-sequence)
* Map: map (lambda parameter: expression, iterable-sequences)
* Reduce: reduce (lambda parameter1, parameter2: expression, iterable-sequence)
* Do not write complicated lambda functions in a production environment because it will be difficult for code-maintainers.

[J5]I've added a table, but the explanation is necessary to understand the differences.

[vV4]Put this section in the table fomat. See this link: https://www.guru99.com/call-by-value-vs-call-by-reference.html add more points in this section.

[J3]It is below the explanation

[vV2]Image is missing in this section.

[KR1]Do not want images of code… just paste the output on pre tag… do it everywhere

[KR6]Add more examples as we want the word count to be 2300+

# Python Vs PHP: What's the Difference?

## What is Python?

Python is a high level object-oriented, programming language. It has built-in data structures, combined with dynamic typing & binding which makes it an ideal choice for rapid application development. Python also offers support for modules and packages, which allows system modularity and code reuse.

It is one of the fastest programming languages as it requires very few lines of code. Its emphasis is on readability and simplicity, which make it a great choice for beginners.

In this tutorial, you will learn:

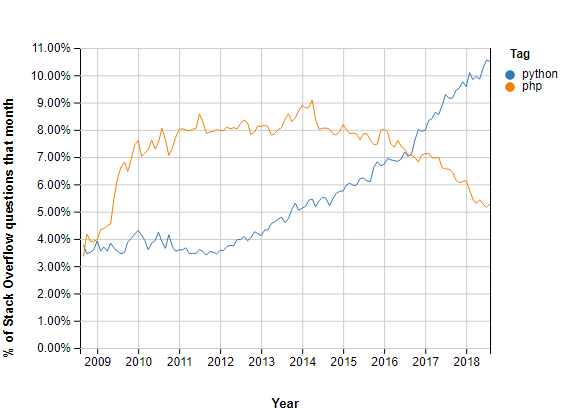
* [What is Python?](https://www.guru99.com/python-vs-php.html#1)
* [What is PHP?](https://www.guru99.com/python-vs-php.html#2)
* [Why Python?](https://www.guru99.com/python-vs-php.html#3)
* [Why PHP?](https://www.guru99.com/python-vs-php.html#4)
* [History of Python](https://www.guru99.com/python-vs-php.html#5)
* [History of PHP](https://www.guru99.com/python-vs-php.html#6)
* [Python vs. PHP](https://www.guru99.com/python-vs-php.html#7)
* [Features of Python](https://www.guru99.com/python-vs-php.html#8)
* [Feature of PHP](https://www.guru99.com/python-vs-php.html#9)

## What is PHP?

PHP stands for Hypertext Pre-processor. It is a server-side scripting language. It is used to develop a dynamic website or web application. PHP can easily integrate with all major web servers on all major operating systems.

PHP is the widely-used, free, and efficient option to competitors such as Microsoft's ASP. Non-technical users can easily learn a few useful tricks to make their web pages easier to manage, and more useful.

## Why Python?



Stackoverflow Python vs. PHP tags

Here, are certain reasons why you should select Python:

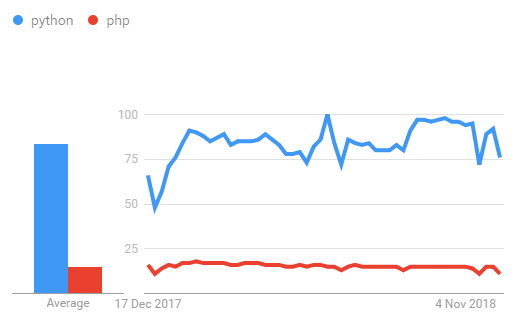
* Compared with the code of other language python code is easy to write and debug. Therefore, its source code is relatively easy to maintain.
* Python is a portable language so it can run on a wide variety of Operating systems and platforms.
* Python comes with many prebuilt libraries which makes your development task easily.
* Python helps you to make complex programming simpler. As it internally deals with memory addresses, garbage collection.
* Python provides an interactive shell which helps you to test the things before it's actual implementation.
* Python offers database interfaces to all major commercial DBMS system.
* Python supports GUI applications and has as a framework for Web. Example: tkmter, WXPython, Django.

## Why PHP?

Here, are prime reasons why you should select PHP:

* PHP runs on various platforms like Windows, Unix, Linux, Mac OS X, etc.
* PHP is compatible with almost Apaches, IIS servers
* PHP is easy to learn which runs efficiently on the server side
* PHP is free to download from its official website www.php.net

## History of Python



Google Trends Python Vs. PHP

* Python s first conceived in the late 1980s, but its implementation started only in December 1989
* It was invented in the Netherlands, by Guido van Rossum.
* Rossum published the first version of Python (0.9.0) in February 199
* t was derived from ABC programming language which is a general-purpose programming language.
* Nowadays Python is maintained by a core development team although Rossum still plays a vital role in directing it's progress

## History of PHP

* PHP was started as a Perl hack in 1995 by Rasmus Lerdorf
* It was rewritten again in 1997 and released as PHP2.0 with a new passer engine by Zeev Suraski and Gutmans.
* PHP 2 transformed the language into a Server-side scripting language.
* PHP 3 (1998) added support for ODBC(Open Database Connectivity) multiple platform support and email protocols are added
* PHP 4 version which was released in the year 2000 has become an independent component of the web server for added efficiency.
* PHP 5 which was released in the year 2004 adds Zend Engine II with object-oriented programming, robust XML support.

## Python vs. PHP

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Python** | **PHP** |
| Frameworks | Python has a smaller number of frameworks | PHP users have access to mature frameworks. |
| Popularity | Widely used in AI, data science and scientific community | Language of choice for web development |
| Database Connectivity | Does not support database connectivity as broadly as PHP | It is possible to access more than 20 different databases. |
| Community Support | Currently growing at fast pace. | Wider community support. |
| Learning curve | Python is better in the long-term projects. | PHP has a very low learning curve, and it is straightforward to get started with. |
| Readability | Python uses indentation enforcements that are quite strict. This makes it more readable than PHP. | PHP language is highly documented and follows a classic approach |
| Type of language | General purpose programming language | Specialized for Web development programming language. |
| Syntax | A very clear and concise, syntax of codes | The built-in library has a wide range of naming conventions |
| A famous company using this technology | Uber, Pinterest, Mozilla | Hootsuite, Viber, Appcelerator |
| Git hub stars | 30.4 K | 30 K |
| Fork | 8.23 K | 8.04 K |
| TIOBE rating | 4th position | 7th position |
| Key Features | Rapid development, Dynamic typing and beautiful code. | Open source, Easy Deployment, Continual Improvements. |
| Best frameworks | Flask, Django, jam.py. | Slim, Laravel, Fast-free framework. |
| Salaries | The average salary for a professional Python Developer is $120,024 per year in the United States. | The avg salary for a PHP Developer is $86,017 per year in the United States. |

## Features of Python

* Easy to learn, read and maintain
* It can run on various hardware platforms & using the same interface.
* You can include low-level modules to the Python interpreter.
* Python offers an ideal structure and support for large programs.
* Python offers support for automatic garbage collection.
* It supports for an interactive mode of testing and debugging.
* It offers high-level dynamic data types and also supports dynamic type checking.
* Python language can be integrated with Java, C, and C++ programming code.

## Features of PHP

* Database Integration Supports many databases, such as Oracle, MySQL, etc.
* It is easy to use, compare to another scripting language
* It is faster than another scripting language, e.g., ASP and JSP
* Open source means you don't need to pay for using PHP; you can freely download and use it
* PHP has some predefined error reporting constants to generate an error notice or warning
* PHP offers access logging by creating the summary of recent accesses of the user.

## What is better?

Even after comparing both of these languages, selecting one is an undoubtedly difficult task because PHP and Python both are good on their way. Python is the gateway to machine learning codes. On the other side, PHP is used for server-side scripting and web development.

## KEY DIFFERENCE

* Python is a high-level object-oriented programming language used for rapid application development whereas PHP stands for Hypertext pre-processor used to develop a dynamic website or web application.
* Python is widely used in AI, data science, and scientific community whereas PHP is the language of choice for web development.
* Python has a small number of frameworks whereas PHP has many frameworks.
* Python offers very clear and concise syntax of codes whereas PHP has a wide range of naming conventions and syntax.
* Key features of Python are Rapid development, Dynamic typing and beautiful code whereas Key features of PHP are Open source, Easy Deployment, Continual Improvements.

# Find AVERAGE of a List in Python with Example

The formula to calculate average is done by calculating the sum of the numbers in the list divided by the count of numbers in the list.

The average of a list can be done in many ways listed below:

* Python Average by using the loop
* By using sum() and len() built-in functions from python
* Using mean() function to calculate the average from the statistics module.
* Using mean() from numpy library

In this Python tutorial, you will learn:

* [Python Average via Loop](https://www.guru99.com/find-average-list-python.html#1)
* [Using sum() and len() built-in functions](https://www.guru99.com/find-average-list-python.html#2)
* [Using mean function from statistics module](https://www.guru99.com/find-average-list-python.html#3)
* [Using mean() from numpy library](https://www.guru99.com/find-average-list-python.html#4)

## Python Average via Loop

In this example, we have initialized the variable sum\_num to zero and used for loop. The for-loop will loop through the elements present in the list, and each number is added and saved inside the sum\_num variable. The average is calculated by using the sum\_num divided by the count of the numbers in the list using len() built-in function.

### Code Example:

def cal\_average(num):

sum\_num = 0

for t in num:

sum\_num = sum\_num + t

avg = sum\_num / len(num)

return avg

print("The average is", cal\_average([18,25,3,41,5]))

Output:

The average is 18.4

## Using sum() and len() built-in functions

In this example the sum() and len() built-in functions are used. It is a straight forward way to calculate the average as you don't have to loop through the elements, and also, the code size is reduced. The average can be calculated with just one line of code as shown below.

### Program Example:

# Example to find average of list

number\_list = [45, 34, 10, 36, 12, 6, 80]

avg = sum(number\_list)/len(number\_list)

print("The average is ", round(avg,2))

Output:

The average is 31.86

## Using mean function from statistics module

You can easily calculate the "average" using the mean function from the statistics module. Example shown below

# Example to find the average of the list

from statistics import mean

number\_list = [45, 34, 10, 36, 12, 6, 80]

avg = mean(number\_list)

print("The average is ", round(avg,2))

Output:

The average is 31.86

## Using mean() from numpy library

Numpy library is commonly used library to work on large multi-dimensional arrays. It also has a large collection of mathematical functions to be used on arrays to perform various tasks. One important one is the mean() function that will give us the average for the list given.

### Code Example:

# Example to find avearge of list

from numpy import mean

number\_list = [45, 34, 10, 36, 12, 6, 80]

avg = mean(number\_list)

print("The average is ", round(avg,2))

Output:

C:\pythontest>python testavg.py

The average is 31.86

### Summary:

* The formula to calculate average is done by calculating the sum of the numbers in the list divided by the count of numbers in the list.
* The average of a list can be done in many ways i.e
  + Python Average by using the loop
  + By using sum() and len() built-in functions from python
  + Using mean() function to calculate the average from the statistics module.
  + Using mean() from numpy library

# Python abs() Function: Absolute Value Examples

## abs()

abs() is a built-in function available with python, and it will return you the absolute value for the input given.

## Syntax:

abs(value)

## Parameters: (value)

The input value to be given to abs() to get the absolute value. It can be an integer, a float, or a complex number.

## Return Value:

It will return the absolute value for the given number.

* If the input is an integer, the return value also will be an integer.
* If the input is a float, the return value will also be float.
* If the input is a complex number, the return value will be the magnitude of the input.

## Examples:

#### Code Example 1: Integer and Float number

To get the absolute value of an integer and float number check this code:

# testing abs() for an integer and float

int\_num = -25

float\_num = -10.50

print("The absolute value of an integer number is:", abs(int\_num))

print("The absolute value of a float number is:", abs(float\_num))

**Output:**

The absolute value of an integer number is: 25

The absolute value of a float number is: 10.5

## Example 2: Complex Number

To get absolute value of complex number

# testing abs() for a complex number

complex\_num = (3+10j)

print("The magnitude of the complex number is:", abs(complex\_num))

Output:

The magnitude of the complex number is: 10.44030650891055

### Summary:

* Abs() is a built-in function available with python, and it will return you the absolute value for the input given.
* Value is an input value to be given to abs() to get the absolute value. It can be an integer, a float, or a complex number.
* The abs() method takes one argument, i.e. the value you want to get the absolute.
* The abs function returns the absolute value for the given number.

**Python round() function with EXAMPLES**

## Round()

Round() is a built-in function available with python. It will return you a float number that will be rounded to the decimal places which are given as input.

If the decimal places to be rounded are not specified, it is considered as 0, and it will round to the nearest integer.

In this Python Tutorial, you will learn:

* [Round()](https://www.guru99.com/round-function-python.html#1)
* [Syntax:](https://www.guru99.com/round-function-python.html#2)
* [How much Impact can Rounding Have? (Rounding vs Truncation)](https://www.guru99.com/round-function-python.html#3)
* [Example: Rounding Float Numbers](https://www.guru99.com/round-function-python.html#4)
* [Example: Rounding Integer Values](https://www.guru99.com/round-function-python.html#5)
* [Example: Rounding on Negative Numbers](https://www.guru99.com/round-function-python.html#6)
* [Example: Round Numpy Arrays](https://www.guru99.com/round-function-python.html#7)
* [Example: Decimal Module](https://www.guru99.com/round-function-python.html#8)

## Syntax:

round(float\_num, num\_of\_decimals)

#### Parameters

* float\_num: the float number to be rounded.
* num\_of\_decimals: (optional) The number of decimals to be considered while rounding. It is optional, and if not specified, it defaults to 0, and the rounding is done to the nearest integer.

#### Description

The round() method takes two argument

* the number to be rounded and
* the decimal places it should consider while rounding.

The second argument is optional and defaults to 0 when not specified, and in such case, it will round to the nearest integer, and the return type will also be an integer.

When the decimal places, i.e. the second argument, is present, it will round to the number of places given. The return type will be a float.

If the number after the decimal place given

* >=5 than + 1 will be added to the final value
* <5 than the final value will return as it is up to the decimal places mentioned.

#### Return value

It will return an integer value if the num\_of\_decimals is not given and a float value if the num\_of\_decimals is given. Please note the value will be rounded to +1 if the value after the decimal point is >=5 else it will return the value as it is up to the decimal places mentioned.

## How much Impact can Rounding Have? (Rounding vs Truncation)

The best example to show the impact of rounding is for the stock exchange market. In the past i.e in the year 1982, the [Vancouver Stock Exchange](https://en.wikipedia.org/wiki/Vancouver_Stock_Exchange) (VSE): used to truncate the stock values to three decimal places on each trade.

It was done almost 3000 times every day. The accumulated truncations lead to a loss of around 25 points per month.

An example of truncating the values versus rounding is shown below.

Consider the floating-point numbers generated below as stock values. Right now I am generating it for a range of

1,000,000 seconds between 0.01 and 0.05.

#### Examples:

arr = [random.uniform(0.01, 0.05) for \_ in range(1000000)]

To show the impact of rounding, I have written a small piece of code wherein at first, you need to use the numbers up to only 3 decimal places, i.e. truncating the number after 3 decimal places.

I have the original total value, the total coming from truncated values and the difference between original and truncated value.

On the same set of numbers, I have been using round() method up to 3 decimal places and calculating the sum and the difference between the original value and the rounded value.

Here are the example and the output

### Example 1

import random

def truncate(num):

return int(num \* 1000) / 1000

arr = [random.uniform(0.01, 0.05) for \_ in range(1000000)]

sum\_num = 0

sum\_truncate = 0

for i in arr:

sum\_num = sum\_num + i

sum\_truncate = truncate(sum\_truncate + i)

print("Testing by using truncating upto 3 decimal places")

print("The original sum is = ", sum\_num)

print("The total using truncate = ", sum\_truncate)

print("The difference from original - truncate = ", sum\_num - sum\_truncate)

print("\n\n")

print("Testing by using round() upto 3 decimal places")

sum\_num1 = 0

sum\_truncate1 = 0

for i in arr:

sum\_num1 = sum\_num1 + i

sum\_truncate1 = round(sum\_truncate1 + i, 3)

print("The original sum is =", sum\_num1)

print("The total using round = ", sum\_truncate1)

print("The difference from original - round =", sum\_num1 - sum\_truncate1)

### Output:

Testing by using truncating upto 3 decimal places

The original sum is = 29985.958619386867

The total using truncate = 29486.057

The difference from original - truncate = 499.9016193868665

Testing by using round() up to 3 decimal places

The original sum is = 29985.958619386867

The total using round = 29985.912

The difference from original - round = 0.04661938686695066

The difference between original and after truncating is 499.9016193868665, and from round, it is 0.04661938686695066

The difference seems to be very big, and the example shows how to round() method helps in calculating close to accuracy.

## Example: Rounding Float Numbers

In this program, we will see how rounding words on floating numbers

# testing round()

float\_num1 = 10.60 # here the value will be rounded to 11 as after the decimal point the number is 6 that is >=5

float\_num2 = 10.40 # here the value will be rounded to 10 as after the decimal point the number is 4 that is <5

float\_num3 = 10.3456 # here the value will be 10.35 as after the 2 decimal points the value >=5

float\_num4 = 10.3445 #here the value will be 10.34 as after the 2 decimal points the value is <5

print("The rounded value without num\_of\_decimals is :", round(float\_num1))

print("The rounded value without num\_of\_decimals is :", round(float\_num2))

print("The rounded value with num\_of\_decimals as 2 is :", round(float\_num3, 2))

print("The rounded value with num\_of\_decimals as 2 is :", round(float\_num4, 2))

Output:

The rounded value without num\_of\_decimals is : 11

The rounded value without num\_of\_decimals is : 10

The rounded value with num\_of\_decimals as 2 is : 10.35

The rounded value with num\_of\_decimals as 2 is : 10.34

## Example: Rounding Integer Values

If you happen to use round() on an integer value, it will just return you the number back without any changes.

# testing round() on a integer

num = 15

print("The output is", round(num))

Output:

The output is 15

## Example: Rounding on Negative Numbers

Let us see few examples of how rounding works on negative numbers

# testing round()

num = -2.8

num1 = -1.5

print("The value after rounding is", round(num))

print("The value after rounding is", round(num1))

Output:

C:\pythontest>python testround.py

The value after rounding is -3

The value after rounding is -2

## Example: Round Numpy Arrays

How to round numpy arrays in python?

To solve this, we can make use of numpy module and use numpy.round() or numpy.around() method, as shown in the example below.

**Using numpy.round()**

# testing round()

import numpy as np

arr = [-0.341111, 1.455098989, 4.232323, -0.3432326, 7.626632, 5.122323]

arr1 = np.round(arr, 2)

print(arr1)

Output:

C:\pythontest>python testround.py

[-0.34 1.46 4.23 -0.34 7.63 5.12]

We can also use numpy.around(), which gives you the same result as shown in the example below.

## Example: Decimal Module

In addition to the round() function, python has a decimal module that helps in handling decimal numbers more accurately.

The Decimal module comes with rounding types, as shown below :

* ROUND\_CEILING: it will round towards Infinity,
* ROUND\_DOWN: it will round the value towards zero,
* ROUND\_FLOOR: it will round towards -Infinity,
* ROUND\_HALF\_DOWN: it will round to nearest value going towards zero,
* ROUND\_HALF\_EVEN: it will round to nearest with value going to nearest even integer,
* ROUND\_HALF\_UP: it will round to nearest with value going away from zero
* ROUND\_UP: it will round where the value will go away from zero.

In decimal, the quantize() method helps to round to a fixed number of decimal places, and you can specify the rounding to be used, as shown in the example below.

#### Example:

Using round() and decimal methods

import decimal

round\_num = 15.456

final\_val = round(round\_num, 2)

#Using decimal module

final\_val1 = decimal.Decimal(round\_num).quantize(decimal.Decimal('0.00'), rounding=decimal.ROUND\_CEILING)

final\_val2 = decimal.Decimal(round\_num).quantize(decimal.Decimal('0.00'), rounding=decimal.ROUND\_DOWN)

final\_val3 = decimal.Decimal(round\_num).quantize(decimal.Decimal('0.00'), rounding=decimal.ROUND\_FLOOR)

final\_val4 = decimal.Decimal(round\_num).quantize(decimal.Decimal('0.00'), rounding=decimal.ROUND\_HALF\_DOWN)

final\_val5 = decimal.Decimal(round\_num).quantize(decimal.Decimal('0.00'), rounding=decimal.ROUND\_HALF\_EVEN)

final\_val6 = decimal.Decimal(round\_num).quantize(decimal.Decimal('0.00'), rounding=decimal.ROUND\_HALF\_UP)

final\_val7 = decimal.Decimal(round\_num).quantize(decimal.Decimal('0.00'), rounding=decimal.ROUND\_UP)

print("Using round()", final\_val)

print("Using Decimal - ROUND\_CEILING ",final\_val1)

print("Using Decimal - ROUND\_DOWN ",final\_val2)

print("Using Decimal - ROUND\_FLOOR ",final\_val3)

print("Using Decimal - ROUND\_HALF\_DOWN ",final\_val4)

print("Using Decimal - ROUND\_HALF\_EVEN ",final\_val5)

print("Using Decimal - ROUND\_HALF\_UP ",final\_val6)

print("Using Decimal - ROUND\_UP ",final\_val7)

Output:

Using round() 15.46

Using Decimal - ROUND\_CEILING 15.46

Using Decimal - ROUND\_DOWN 15.45

Using Decimal - ROUND\_FLOOR 15.45

Using Decimal - ROUND\_HALF\_DOWN 15.46

Using Decimal - ROUND\_HALF\_EVEN 15.46

Using Decimal - ROUND\_HALF\_UP 15.46

Using Decimal - ROUND\_UP 15.46

### Summary:

* Round(float\_num, Num\_of\_decimals) is a built-in function available with python. It will return you the float number that will be rounded to the decimal places which are given as input.
* float\_num: the float number to be rounded.
* Num\_of\_decimals: It is the number of decimals to be considered while rounding.
* It will return an integer value if the num\_of\_decimals is not given and a float value if the num\_of\_decimals is given.

# Python Timeit() with Examples

**timeit()** method is available with python library timeit. It is used to get the execution time taken for the small code given. The library runs the code statement 1 million times and provides the minimum time taken from the set. It is a useful method that helps in checking the performance of the code.

## Syntax:

timeit.timeit(stmt, setup,timer, number)

#### Parameters

* **stmt**: This will take the code for which you want to measure the execution time. The default value is "pass".
* **setup**: This will have setup details that need to be executed before stmt. The default value is "pass."
* **timer**: This will have the timer value, timeit() already has a default value set, and we can ignore it.
* **number**: The stmt will execute as per the number is given here. The default value is 1000000.

To work with timeit(), we need to import the module, as shown below:

import timeit

## First Example

Here is a simple example of timeit() function

### Code Example 1:

# testing timeit()

import timeit

print(timeit.timeit('output = 10\*5'))

**Output:**

0.06127880399999999

We have seen a simple example that gives us the execution time of the simple code statement output = 10\*5, and the time is taken to execute it is 0.06127880399999999.

## Timing Multiple lines in python code

There are two you can execute multiple lines of code in timeit.timeit(), using a semicolon or by saving the code enclosed as a string with triple quotes.

Here are examples that show the working of it.

### Example 1: Using semicolon

import timeit

print("The time taken is ",timeit.timeit(stmt='a=10;b=10;sum=a+b'))

Output:

The time taken is 0.137031482

### Example 2: Using triple quotes

import timeit

import\_module = "import random"

testcode = '''

def test():

return random.randint(10, 100)

'''

print(timeit.repeat(stmt=testcode, setup=import\_module))

Output:

C:\pythontest>python testtimeit.py

The time taken is 0.182619178

## timeit - Methods:

Here, are 2 important timeit methods

**timeit.default\_timer()** : This will return the default time when executed.

**timeit.repeat(stmt, setup, timer, repeat, number)** : same as timeit() , but with repeat the timeit() is called the number of times repeat is given.

### Program Example 1:

# testing timeit()

import timeit

import\_module = "import random"

testcode = '''

def test():

return random.randint(10, 100)

'''

print(timeit.timeit(stmt=testcode, setup=import\_module))

Output:

0.46715912400000004

### Example 2:

default\_timer() Example

# testing timeit()

import timeit

import random

def test():

return random.randint(10, 100)

starttime = timeit.default\_timer()

print("The start time is :",starttime)

test()

print("The time difference is :", timeit.default\_timer() - starttime)

Output:

The start time is : 0.220261875

The time difference is : 0.0004737320000000045

### Example 3: timeit.repeat()

# testing timeit()

import timeit

import\_module = "import random"

testcode = '''

def test():

return random.randint(10, 100)

'''

print(timeit.repeat(stmt=testcode, setup=import\_module, repeat=5))

Output:

[0.43638873, 0.5040939680000001, 0.5069179909999999, 0.3943449330000002, 0.3546886979999999]

timeit.repeat() works similar to timeit.timeit() function, with the only difference it takes in the repeat argument and gives back the execution time in array format with values as per the repeat number.

## Executing timing function timeit.timeit() inside command-line interface

The syntax to execute your function inside timeit() on the command line is as follows:

python -m timeit [-n N] [-r N] [-s S] [-t] [-c] [-h] [code statement ...]

Command line parameters:

* -n N: the number of times you want the code to execute.
* -r N: the number of times you want the timeit() function to repeat
* -s S: this will have setup details that will get executed before code execution.
* -t: for this, you can make use of time.time()
* -c: for this, you can make use of time.clock()
* -h: for help
* code statement: The code details.

### Example:

C:\pythontest>python -m timeit -s 'text="hello world"'

20000000 loops, best of 5: 13.1 nsec per loop

Another way you can execute inside command line is as shown below:

### Example :

>>> import timeit

>>> print("The time taken is ",timeit.timeit(stmt='a=10;b=10;sum=a+b'))

The time taken is 0.15048536300000137

>>>

## Why is timeit() the best way to measure the execution time of Python code?

Here are a few reasons why we consider timeit() is the best way to measure execution time.

* It runs the code statement 1 million times that is the default value, and from that, it will return you the minimum time taken. You can also increase/decrease the 1 million by setting the argument number in time () function.
* While executing the test, the garbage collection is disabled every time by time () function.
* timeit() internally takes the accurate time as per your operating system being used. For example, it will use time.clock() for Windows operating system and time.time() for mac and Linux.

#### ****Summary****

Timeit()is used to get the execution time taken for the small code given

Parameters used with timeit()

* stmt: This will take the code you want to measure the execution time
* setup: This will have setup details that need to be executed before stmt
* timer: This will have the timer value, timeit() already has a default value set, and we can ignore it.
* number: The stmt will execute as per the number is given here.

# Python string length | len() method Example

len() is a built-in function in python. You can use the len() to get the length of the given string, array, list, tuple, dictionary, etc.

You can use len function to optimize the performance of the program. The number of elements stored in the object is never calculated, so len helps provide the number of elements.

## Syntax:

len(value)

## Parameters:

**Value**: the given value you want the length of.

## Return value

It will return an integer value i.e. the length of the given string, or array, or list or collections.

Various types of Return values:

### Strings:

It returns the number of characters in a string, which includes punctuation, space, and all type of special characters. However, you should be very careful while using len of a Null variable.

### Empty:

Empty is a second return call which has zero characters, but it is always None.

### Collections:

The len built-in returns the number of elements in a collection.

### TypeError:

Len function depends on the type of the variable passed to it. A Non-Type does not have any built-in support.

### Dictionary:

For the dictionary, each pair is counted as one unit. However, values and keys are not independent.

### Example 1: How find the length of the given string?

# testing len()

str1 = "Welcome to Guru99 Python Tutorials"

print("The length of the string is :", len(str1))

Output:

C:\pythontest>python teststrlen.py

The length of the string is : 35

### Example 2: How to find the length of the list in python?

# to find the length of the list

list1 = ["Tim","Charlie","Tiffany","Robert"]

print("The length of the list is", len(list1))

Output:

C:\pythontest>python testlen.py

The length of the list is 4

### Example 3: How to find the length of a tuple in python

# to find the length of the tuple

Tup = ('Jan','feb','march')

print("The length of the tuple is", len(Tup))

Output:

C:\pythontest>python testlen.py

The length of the tuple is 3

### Example 4: How to find the length of the dictionary in Python?

# to find the length of the Dictionary

Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}

print("The length of the Dictionary is", len(Dict))

Output:

C:\pythontest>python testlen.py

The length of the Dictionary is 3

### Example 5: How to find the length of the array in python

# to find the length of the array

arr1 = ['Tim','Charlie','Tiffany','Robert']

print("The length of the Array is", len(arr1))

Output:

C:\pythontest>python testlen.py

The length of the Array is 4

### Summary:

* len() is a built-in function in python.You can use the len() to get the length of the given string, array, list, tuple, dictionary, etc.
* Value: the given value you want the length of.
* Return value a return an integer value i.e. the length of the given string, or array, or list, or collections.

# Python SciPy Tutorial: Learn with Example

## What is SciPy?

SciPy is an Open Source Python-based library, which is used in mathematics, scientific computing, Engineering, and technical computing.

SciPy also pronounced as "Sigh Pi."

**Sub-packages of SciPy:**

* File input/output - [**scipy.io**](https://docs.scipy.org/doc/scipy/reference/io.html)
* Special Function - [**scipy.special**](https://docs.scipy.org/doc/scipy/reference/special.html)
* Linear Algebra Operation - [**scipy.linalg**](https://docs.scipy.org/doc/scipy/reference/linalg.html)
* Interpolation - [**scipy.interpolate**](https://docs.scipy.org/doc/scipy/reference/interpolate.html)
* Optimization and fit - [**scipy.optimize**](https://docs.scipy.org/doc/scipy/reference/optimize.html)
* Statistics and random numbers - [**scipy.stats**](https://docs.scipy.org/doc/scipy/reference/stats.html)
* Numerical Integration - [**scipy.integrate**](https://docs.scipy.org/doc/scipy/reference/integrate.html)
* Fast Fourier transforms - [**scipy.fftpack**](https://docs.scipy.org/doc/scipy/reference/fftpack.html)
* Signal Processing - [**scipy.signal**](https://docs.scipy.org/doc/scipy/reference/signal.html)
* Image manipulation – [**scipy.ndimage**](https://docs.scipy.org/doc/scipy/reference/ndimage.html)

In this tutorial, you will learn:

* [What is SciPy?](https://www.guru99.com/scipy-tutorial.html#1)
* [Why use SciPy](https://www.guru99.com/scipy-tutorial.html#2)
* [Numpy VS SciPy](https://www.guru99.com/scipy-tutorial.html#3)
* [SciPy - Installation and Environment Setup](https://www.guru99.com/scipy-tutorial.html#4)
* [File Input / Output package:](https://www.guru99.com/scipy-tutorial.html#5)
* [Special Function package:](https://www.guru99.com/scipy-tutorial.html#6)
* [Linear Algebra with SciPy:](https://www.guru99.com/scipy-tutorial.html#7)
* [Discrete Fourier Transform – scipy.fftpack](https://www.guru99.com/scipy-tutorial.html#8)
* [Optimization and Fit in SciPy – scipy.optimize](https://www.guru99.com/scipy-tutorial.html#9)
* [Nelder –Mead Algorithm:](https://www.guru99.com/scipy-tutorial.html#10)
* [Image Processing with SciPy – scipy.ndimage](https://www.guru99.com/scipy-tutorial.html#11)

## Why use SciPy

* SciPy contains varieties of sub packages which help to solve the most common issue related to Scientific Computation.
* SciPy is the most used Scientific library only second to GNU Scientific Library for C/C++ or Matlab's.
* Easy to use and understand as well as fast computational power.
* It can operate on an array of NumPy library.

## Numpy VS SciPy

**Numpy:**

* Numpy is written in C and use for mathematical or numeric calculation.
* It is faster than other Python Libraries
* Numpy is the most useful library for Data Science to perform basic calculations.
* Numpy contains nothing but array data type which performs the most basic operation like sorting, shaping, indexing, etc.

**SciPy:**

* SciPy is built in top of the NumPy
* SciPy is a fully-featured version of Linear Algebra while Numpy contains only a few features.
* Most new Data Science features are available in Scipy rather than Numpy.

## SciPy - Installation and Environment Setup

You can also install SciPy in Windows via pip

Python3 -m pip install --user numpy scipy

Install Scipy on Linux

sudo apt-get install python-scipy python-numpy

Install SciPy in Mac

sudo port install py35-scipy py35-numpy

Before start to learning SciPy, you need to know basic functionality as well as different types of an array of [NumPy](https://www.guru99.com/numpy-tutorial.html)

The standard way of import infSciPy modules and Numpy:

from scipy import special #same for other modules

import numpy as np

## File Input / Output package:

Scipy, I/O package, has a wide range of functions for work with different files format which are Matlab, Arff, Wave, Matrix Market, IDL, NetCDF, TXT, CSV and binary format.

Let's we take one file format example as which are regularly use of MatLab:

import numpy as np

from scipy import io as sio

array = np.ones((4, 4))

sio.savemat('example.mat', {'ar': array})

data = sio.loadmat(‘example.mat', struct\_as\_record=True)

data['array']

Output:

array([[ 1., 1., 1., 1.],

[ 1., 1., 1., 1.],

[ 1., 1., 1., 1.],

[ 1., 1., 1., 1.]])

Code Explanation

* **Line 1 & 2:**Import the essential library scipy with i/o package and Numpy.
* **Line 3**: Create 4 x 4, dimensional one's array
* **Line 4**: Store array in **example.mat** file.
* **Line 5:**Get data from **example.mat** file
* **Line 6**: Print output.

## Special Function package

* **scipy.special** package contains numerous functions of mathematical physics.
* SciPy special function includes Cubic Root, Exponential, Log sum Exponential, Lambert, Permutation and Combinations, Gamma, Bessel, hypergeometric, Kelvin, beta, parabolic cylinder, Relative Error Exponential, etc..
* For one line description all of these function, type in Python console:

help(scipy.special)

Output :

NAME

scipy.special

DESCRIPTION

========================================

Special functions (:mod:`scipy.special`)

========================================

.. module:: scipy.special

Nearly all of the functions below are universal functions and follow

broadcasting and automatic array-looping rules. Exceptions are noted.

### Cubic Root Function:

Cubic Root function finds the cube root of values.

Syntax:

scipy.special.cbrt(x)

Example:

from scipy.special import cbrt

#Find cubic root of 27 & 64 using cbrt() function

cb = cbrt([27, 64])

#print value of cb

print(cb)

Output: array([3., 4.])

### Exponential Function:

Exponential function computes the 10\*\*x element-wise.

Example:

from scipy.special import exp10

#define exp10 function and pass value in its

exp = exp10([1,10])

print(exp)

Output: [1.e+01 1.e+10]

### Permutations & Combinations:

SciPy also gives functionality to calculate Permutations and Combinations.

**Combinations -**scipy.special.comb(N,k**)**

Example:

from scipy.special import comb

#find combinations of 5, 2 values using comb(N, k)

com = comb(5, 2, exact = False, repetition=True)

print(com)

Output: 15.0

**Permutations –**

scipy.special.perm(N,k)

Example:

from scipy.special import perm

#find permutation of 5, 2 using perm (N, k) function

per = perm(5, 2, exact = True)

print(per)

Output: 20

### Log Sum Exponential Function

Log Sum Exponential computes the log of sum exponential input element.

Syntax :

scipy.special.logsumexp(x)

### Bessel Function

Nth integer order calculation function

Syntax :

scipy.special.jn()

## Linear Algebra with SciPy

* Linear Algebra of SciPy is an implementation of BLAS and ATLAS LAPACK libraries.
* Performance of Linear Algebra is very fast compared to BLAS and LAPACK.
* Linear algebra routine accepts two-dimensional array object and output is also a two-dimensional array.

Now let's do some test with **scipy.linalg,**

Calculating **determinant** of a two-dimensional matrix,

from scipy import linalg

import numpy as np

#define square matrix

two\_d\_array = np.array([ [4,5], [3,2] ])

#pass values to det() function

linalg.det( two\_d\_array )

Output: -7.0

**Inverse Matrix –**

scipy.linalg.inv()

Inverse Matrix of Scipy calculates the inverse of any square matrix.

Let's see,

from scipy import linalg

import numpy as np

# define square matrix

two\_d\_array = np.array([ [4,5], [3,2] ])

#pass value to function inv()

linalg.inv( two\_d\_array )

Output:

array( [[-0.28571429, 0.71428571],

[ 0.42857143, -0.57142857]] )

**Eigenvalues and Eigenvector –**scipy.linalg.eig()

* The most common problem in linear algebra is eigenvalues and eigenvector which can be easily solved using **eig()** function.
* Now lets we find the Eigenvalue of (**X**) and correspond eigenvector of a two-dimensional square matrix.

Example,

from scipy import linalg

import numpy as np

#define two dimensional array

arr = np.array([[5,4],[6,3]])

#pass value into function

eg\_val, eg\_vect = linalg.eig(arr)

#get eigenvalues

print(eg\_val)

#get eigenvectors

print(eg\_vect)

Output:

[ 9.+0.j -1.+0.j] #eigenvalues

[ [ 0.70710678 -0.5547002 ] #eigenvectors

[ 0.70710678 0.83205029] ]

## Discrete Fourier Transform – scipy.fftpack

* DFT is a mathematical technique which is used in converting spatial data into frequency data.
* FFT (Fast Fourier Transformation) is an algorithm for computing DFT
* FFT is applied to a multidimensional array.
* Frequency defines the number of signal or wavelength in particular time period.

Example: Take a wave and show using Matplotlib library. we take simple periodic function example of sin(20 × 2πt)

%matplotlib inline

from matplotlib import pyplot as plt

import numpy as np

#Frequency in terms of Hertz

fre = 5

#Sample rate

fre\_samp = 50

t = np.linspace(0, 2, 2 \* fre\_samp, endpoint = False )

a = np.sin(fre \* 2 \* np.pi \* t)

figure, axis = plt.subplots()

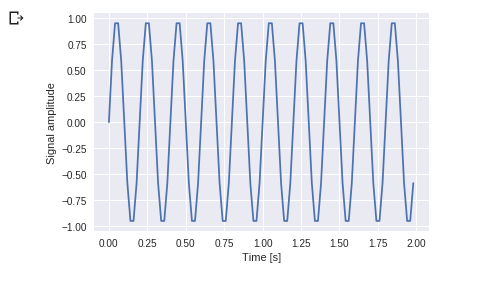
axis.plot(t, a)

axis.set\_xlabel ('Time (s)')

axis.set\_ylabel ('Signal amplitude')

plt.show()

Output :



You can see this. Frequency is 5 Hz and its signal repeats in 1/5 seconds – it's call as a particular time period.

Now let us use this sinusoid wave with the help of DFT application.

from scipy import fftpack

A = fftpack.fft(a)

frequency = fftpack.fftfreq(len(a)) \* fre\_samp

figure, axis = plt.subplots()

axis.stem(frequency, np.abs(A))

axis.set\_xlabel('Frequency in Hz')

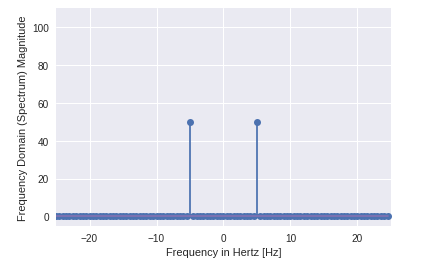
axis.set\_ylabel('Frequency Spectrum Magnitude')

axis.set\_xlim(-fre\_samp / 2, fre\_samp/ 2)

axis.set\_ylim(-5, 110)

plt.show()

Output:

[](https://www.guru99.com/images/1/120518_0746_PythonSciPy2.png)

* You can clearly see that output is a one-dimensional array.
* Input containing complex values are zero except two points.
* In DFT example we visualize the magnitude of the signal.

## Optimization and Fit in SciPy – scipy.optimize

* Optimization provides a useful algorithm for minimization of curve fitting, multidimensional or scalar and root fitting.
* Let's take an example of **a Scalar Function,**to find minimum scalar function**.**

%matplotlib inline

import matplotlib.pyplot as plt

from scipy import optimize

import numpy as np

def function(a):

return a\*2 + 20 \* np.sin(a)

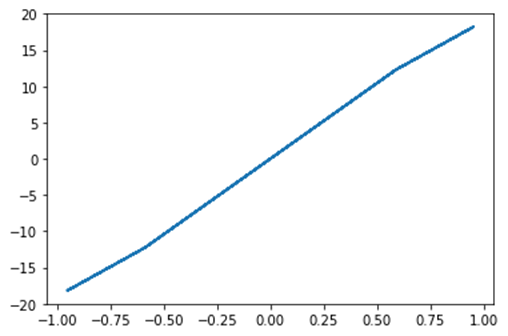
plt.plot(a, function(a))

plt.show()

#use BFGS algorithm for optimization

optimize.fmin\_bfgs(function, 0)

Output:

[](https://www.guru99.com/images/1/120518_0746_PythonSciPy3.png)

Optimization terminated successfully.

Current function value: -23.241676

Iterations: 4

Function evaluations: 18

Gradient evaluations: 6

array([-1.67096375])

* In this example, optimization is done with the help of the gradient descent algorithm from the initial point
* But the possible issue is local minima instead of global minima. If we don't find a neighbor of global minima, then we need to apply global optimization and find global minima function used as **basinhopping()** which combines local optimizer.

**optimize.basinhopping(function, 0)**

Output:

fun: -23.241676238045315

lowest\_optimization\_result:

fun: -23.241676238045315

hess\_inv: array([[0.05023331]])

jac: array([4.76837158e-07])

message: 'Optimization terminated successfully.'

nfev: 15

nit: 3

njev: 5

status: 0

success: True

x: array([-1.67096375])

message: ['requested number of basinhopping iterations completed successfully']

minimization\_failures: 0

nfev: 1530

nit: 100

njev: 510

x: array([-1.67096375])

## Nelder –Mead Algorithm:

* Nelder-Mead algorithm selects through method parameter.
* It provides the most straightforward way of minimization for fair behaved function.
* Nelder – Mead algorithm is not used for gradient evaluations because it may take a longer time to find the solution.

import numpy as np

from scipy.optimize import minimize

#define function f(x)

def f(x):

return .4\*(1 - x[0])\*\*2

optimize.minimize(f, [2, -1], method="Nelder-Mead")

Output:

final\_simplex: (array([[ 1. , -1.27109375],

[ 1. , -1.27118835],

[ 1. , -1.27113762]]), array([0., 0., 0.]))

fun: 0.0

message: 'Optimization terminated successfully.'

nfev: 147

nit: 69

status: 0

success: True

x: array([ 1. , -1.27109375])

## Image Processing with SciPy – scipy.ndimage

* scipy.ndimage is a submodule of SciPy which is mostly used for performing an image related operation
* ndimage means the "n" dimensional image.
* SciPy Image Processing provides Geometrics transformation (rotate, crop, flip), image filtering (sharp and de nosing), display image, image segmentation, classification and features extraction.
* **MISC Package** in SciPy contains prebuilt images which can be used to perform image manipulation task

Example: Let's take a geometric transformation example of images

from scipy import misc

from matplotlib import pyplot as plt

import numpy as np

#get face image of panda from misc package

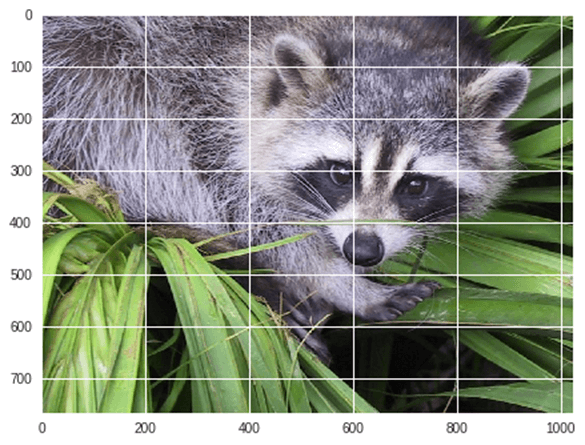
panda = misc.face()

#plot or show image of face

plt.imshow( panda )

plt.show()

Output:



Now we **Flip-down** current image:

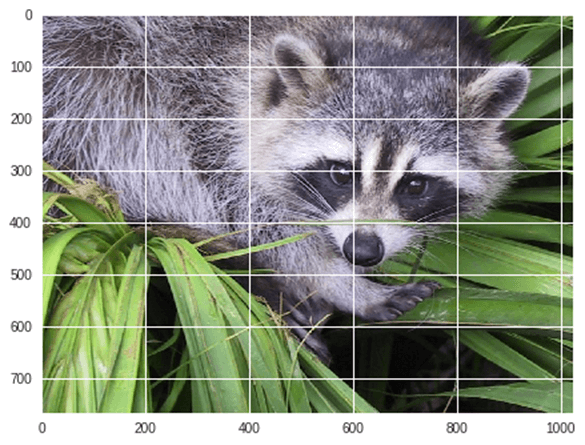
#Flip Down using scipy misc.face image

flip\_down = np.flipud(misc.face())

plt.imshow(flip\_down)

plt.show()

Output:



Example**: Rotation of Image using Scipy,**

from scipy import ndimage, misc

from matplotlib import pyplot as plt

panda = misc.face()

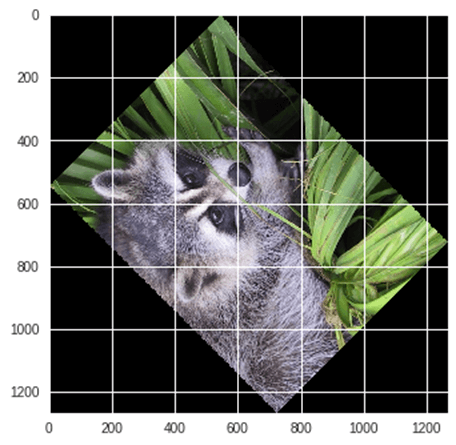
#rotatation function of scipy for image – image rotated 135 degree

panda\_rotate = ndimage.rotate(panda, 135)

plt.imshow(panda\_rotate)

plt.show()

Output:



### Integration with Scipy – Numerical Integration

* When we integrate any function where analytically integrate is not possible, we need to turn for numerical integration
* SciPy provides functionality to integrate function with numerical integration.
* **scipy.integrate** library has single integration, double, triple, multiple, Gaussian quadrate, Romberg, Trapezoidal and Simpson's rules.

Example: Now take an example of **Single Integration**

[](https://www.guru99.com/images/1/120518_0746_PythonSciPy7.jpg)

Here **a** is the upper limit and **b** is the lower limit

from scipy import integrate

# take f(x) function as f

f = lambda x : x\*\*2

#single integration with a = 0 & b = 1

integration = integrate.quad(f, 0 , 1)

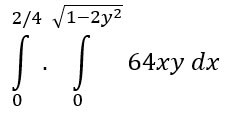
print(integration)

Output:

(0.33333333333333337, 3.700743415417189e-15)

Here function returns two values, in which the first value is integration and second value is estimated error in integral.

Example: Now take an example of **double integration.**We find the double integration of the following equation,

[](https://www.guru99.com/images/1/120518_0746_PythonSciPy8.jpg)

from scipy import integrate

import numpy as np

#import square root function from math lib

from math import sqrt

# set fuction f(x)

f = lambda x, y : 64 \*x\*y

# lower limit of second integral

p = lambda x : 0

# upper limit of first integral

q = lambda y : sqrt(1 - 2\*y\*\*2)

# perform double integration

integration = integrate.dblquad(f , 0 , 2/4, p, q)

print(integration)

Output:

(3.0, 9.657432734515774e-14)

You have seen that above output as same previous one.

## Summary

* SciPy(pronounced as "Sigh Pi") is an Open Source Python-based library, which is used in mathematics, scientific computing, Engineering, and technical computing.
* SciPy contains varieties of sub packages which help to solve the most common issue related to Scientific Computation.
* SciPy is built in top of the NumPy

|  |  |
| --- | --- |
| **Package Name** | **Description** |
| scipy.io | * File input/output |
| scipy.special | * Special Function |
| scipy.linalg | * Linear Algebra Operation |
| scipy.interpolate | * Interpolation |
| scipy.optimize | * Optimization and fit |
| scipy.stats | * Statistics and random numbers |
| scipy.integrate | * Numerical Integration |
| scipy.fftpack | * Fast Fourier transforms |
| scipy.signal | * Signal Processing |
| scipy.ndimage | * Image manipulation – |

**Reading and Writing CSV Files in Python using CSV Module & Pandas**

## What is a CSV file?

A CSV file is a type of plain text file that uses specific structuring to arrange tabular data. CSV is a common format for data interchange as it's compact, simple and general. Many online services allow its users to export tabular data from the website into a CSV file. Files of CSV will open into Excel, and nearly all databases have a tool to allow import from CSV file. The standard format is defined by rows and columns data. Moreover, each row is terminated by a newline to begin the next row. Also within the row, each column is separated by a comma.

In this tutorial, you will learn:

* [What is a CSV file?](https://www.guru99.com/python-csv.html#1)
* [CSV Sample File.](https://www.guru99.com/python-csv.html#2)
* [Python CSV Module](https://www.guru99.com/python-csv.html#3)
* [CSV Module Functions](https://www.guru99.com/python-csv.html#4)
* [Reading CSV Files](https://www.guru99.com/python-csv.html#5)
* [Reading as a Dictionary](https://www.guru99.com/python-csv.html#6)
* [Writing to CSV Files](https://www.guru99.com/python-csv.html#7)
* [Reading CSV Files with Pandas](https://www.guru99.com/python-csv.html#8)
* [Writing to CSV Files with Pandas](https://www.guru99.com/python-csv.html#9)

## CSV Sample File.

Data in the form of tables is also called CSV (comma separated values) - literally "comma-separated values." This is a text format intended for the presentation of tabular data. Each line of the file is one line of the table. The values of individual columns are separated by a separator symbol - a comma (,), a semicolon (;) or another symbol. CSV can be easily read and processed by Python.

Consider the following Tabe

**Table Data**

|  |  |  |  |
| --- | --- | --- | --- |
| Programming language | Designed by | Appeared | Extension |
| Python | Guido van Rossum | 1991 | .py |
| Java | James Gosling | 1995 | .java |
| C++ | Bjarne Stroustrup | 1983 | .cpp |

You can represent this table in csv as below.

**CSV Data**

Programming language, Designed by, Appeared, Extension

Python, Guido van Rossum, 1991, .py

Java, James Gosling, 1995, .java

C++, Bjarne Stroustrup,1983,.cpp

As you can see each row is a new line, and each column is separated with a comma. This is an example of how a CSV file looks like.

## Python CSV Module

Python provides a CSV module to handle CSV files. To read/write data, you need to loop through rows of the CSV. You need to use the split method to get data from specified columns.

## CSV Module Functions

In CSV module documentation you can find following functions:

* csv.field\_size\_limit – return maximum field size
* csv.get\_dialect – get the dialect which is associated with the name
* csv.list\_dialects – show all registered dialects
* csv.reader – read data from a csv file
* csv.register\_dialect - associate dialect with name
* csv.writer – write data to a csv file
* csv.unregister\_dialect - delete the dialect associated with the name the dialect registry
* **csv.QUOTE\_ALL** - Quote everything, regardless of type.
* **csv.QUOTE\_MINIMAL** - Quote fields with special characters
* **csv.QUOTE\_NONNUMERIC** - Quote all fields that aren't numbers value
* **csv.QUOTE\_NONE** – Don't quote anything in output

In this tutorial, we are going to focus only on the reader and writer functions which allow you to edit, modify, and manipulate the data in a CSV file.

## How to Read a CSV File

To read data from CSV files, you must use the reader function to generate a reader object.

The reader function is developed to take each row of the file and make a list of all columns. Then, you have to choose the column you want the variable data for.

It sounds a lot more intricate than it is. Let's take a look at this example, and we will find out that working with csv file isn't so hard.

#import necessary modules

import csv

with open('X:\data.csv','rt')as f:

data = csv.reader(f)

for row in data:

print(row)

When you execute the program above, the output will be:

['Programming language; Designed by; Appeared; Extension']

['Python; Guido van Rossum; 1991; .py']

['Java; James Gosling; 1995; .java']

['C++; Bjarne Stroustrup;1983;.cpp']

## How to Read a CSV as a Dictionary

You can also you use DictReader to read CSV files. The results are interpreted as a dictionary where the header row is the key, and other rows are values.

Consider the following code

#import necessary modules

import csv

reader = csv.DictReader(open("file2.csv"))

for raw in reader:

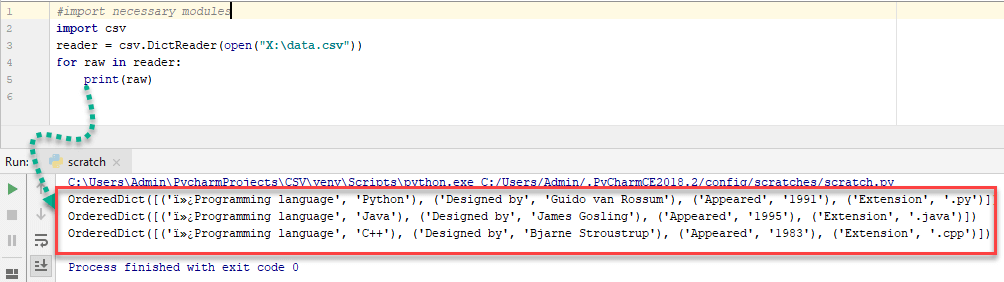
print(raw)

The result of this code is:

OrderedDict([('Programming language', 'Python'), ('Designed by', 'Guido van Rossum'), (' Appeared', ' 1991'), (' Extension', ' .py')])

OrderedDict([('Programming language', 'Java'), ('Designed by', 'James Gosling'), (' Appeared', ' 1995'), (' Extension', ' .java')])

OrderedDict([('Programming language', 'C++'), ('Designed by', ' Bjarne Stroustrup'), (' Appeared', ' 1985'), (' Extension', ' .cpp')])

[](https://www.guru99.com/images/1/122118_0714_ReadingandW1.png)

And this way to read data from CSV file is much easier than earlier method. However, this is not isn't the best way to read data.

## How to write CSV File

When you have a set of data that you would like to store in a CSV file you have to use writer() function. To iterate the data over the rows(lines), you have to use the writerow() function.

Consider the following example. We write data into a file "writeData.csv" where the delimiter is an apostrophe.

#import necessary modules

import csv

with open('X:\writeData.csv', mode='w') as file:

writer = csv.writer(file, delimiter=',', quotechar='"', quoting=csv.QUOTE\_MINIMAL)

#way to write to csv file

writer.writerow(['Programming language', 'Designed by', 'Appeared', 'Extension'])

writer.writerow(['Python', 'Guido van Rossum', '1991', '.py'])

writer.writerow(['Java', 'James Gosling', '1995', '.java'])

writer.writerow(['C++', 'Bjarne Stroustrup', '1985', '.cpp'])

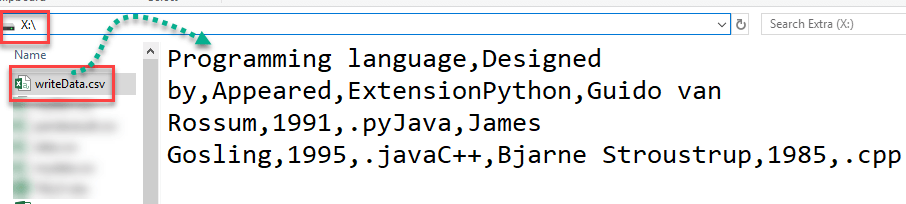
Result in csv file is:

Programming language, Designed by, Appeared, Extension

Python, Guido van Rossum, 1991, .py

Java, James Gosling, 1995, .java

C++, Bjarne Stroustrup,1983,.cpp



## Reading CSV Files with Pandas

Pandas is an opensource library that allows to you perform data manipulation in Python. Pandas provide an easy way to create, manipulate and delete the data.

You must install pandas library with command <code>pip install pandas</code>. In windows, you will execute this command in Command Prompt while in Linux in the Terminal.

Reading the CSV into a pandas DataFrame is very quick and easy:

#import necessary modules

import pandas

result = pandas.read\_csv('X:\data.csv')

print(result)

Result:

Programming language, Designed by, Appeared, Extension

0 Python, Guido van Rossum, 1991, .py

1 Java, James Gosling, 1995, .java

2 C++, Bjarne Stroustrup,1983,.cpp

Very useful library. In just three lines of code you the same result as earlier. Pandas know that the first line of the CSV contained column names, and it will use them automatically.

## Writing to CSV Files with Pandas

Writing to CSV file with Pandas is as easy as reading. Here you can convince in it. First you must create DataFrame based on the following code.

from pandas import DataFrame

C = {'Programming language': ['Python','Java', 'C++'],

'Designed by': ['Guido van Rossum', 'James Gosling', 'Bjarne Stroustrup'],

'Appeared': ['1991', '1995', '1985'],

'Extension': ['.py', '.java', '.cpp'],

}

df = DataFrame(C, columns= ['Programming language', 'Designed by', 'Appeared', 'Extension'])

export\_csv = df.to\_csv (r'X:\pandaresult.csv', index = None, header=True) # here you have to write path, where result file will be stored

print (df)

Here is the output

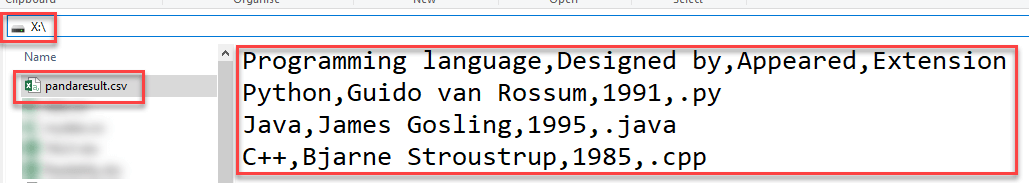
Programming language, Designed by, Appeared, Extension

0 Python, Guido van Rossum, 1991, .py

1 Java, James Gosling, 1995, .java

2 C++, Bjarne Stroustrup,1983,.cpp

And CSV file is created at the specified location.



**Conclusion**

So, now you know how use method 'csv' and also read and write data in CSV format. CSV files are widely used in software applications because they are easy to read and manage, and their small size makes them relatively fast for processing and transmission.

The csv module provides various functions and classes which allow you to read and write easily. You can look at the official Python documentation and find some more interesting tips and modules. CSV is the best way for saving, viewing, and sending data. Actually, it isn't so hard to learn as it seems at the beginning. But with a little practice, you'll master it.

Pandas is a great alternative to read CSV files.

Also, there are other ways to parse text files with libraries like ANTLR, PLY, and PlyPlus. They can all handle heavy-duty parsing, and if simple String manipulation doesn't work, there are regular expressions which you can use.

**Python JSON: Encode(dump), Decode(load) json Data & File (Example)**

## What is JSON?

**JSON**is a standard format for data exchange, which is inspired by JavaScript. Generally, JSON is in string or text format. **JSON**stands for**J**ava**S**cript **O**bject **N**otation.

The syntax of JSON: JSON is written as key and value pair.

{

"Key": "Value",

"Key": "Value",

}

JSON is very similar toPython dictionary. Python supports JSON, and it has an inbuilt library as a JSON.

## JSON Library in Python

'**marshal**' and '**pickle'**external modules of Python maintain a version of **JSON** library. To perform JSON related operations like encoding and decoding in Python you need first to **import**JSON library and for that in your **.py** file,

import json

Following methods are available in the JSON module

|  |  |
| --- | --- |
| **Method** | **Description** |
| dumps() | encoding to JSON objects |
| dump() | encoded string writing on file |
| loads() | Decode the JSON string |
| load() | Decode while JSON file read |

**What You Will Learn:** [hide]

* [What is JSON?](https://www.guru99.com/python-json.html#1)
* [JSON Library in Python](https://www.guru99.com/python-json.html#2)
* [Python to JSON (Encoding)](https://www.guru99.com/python-json.html#3)
* [JSON to Python (Decoding)](https://www.guru99.com/python-json.html#4)
  + [Decoding JSON File or Parsing JSON file in Python](https://www.guru99.com/python-json.html#5)
  + [Compact Encoding in Python](https://www.guru99.com/python-json.html#6)
  + [Format JSON code (Pretty print)](https://www.guru99.com/python-json.html#7)
* [Complex Object encoding of Python](https://www.guru99.com/python-json.html#8)
* [Complex JSON object decoding in Python](https://www.guru99.com/python-json.html#9)
* [Overview of JSON Serialization class JSONEncoder](https://www.guru99.com/python-json.html#10)
* [Overview of JSON Deserialization class JSONDecoder](https://www.guru99.com/python-json.html#11)
  + [Decoding JSON data from URL: Real Life Example.](https://www.guru99.com/python-json.html#12)
* [Exceptions Related to JSON Library in Python](https://www.guru99.com/python-json.html#13)
* [Infinite and NaN Numbers in Python](https://www.guru99.com/python-json.html#14)
* [Repeated key in JSON String](https://www.guru99.com/python-json.html#15)
* [CLI (Command Line Interface) with JSON in Python](https://www.guru99.com/python-json.html#16)
* [Advantages of JSON in Python](https://www.guru99.com/python-json.html#17)
* [Implementation limitation of JSON in Python](https://www.guru99.com/python-json.html#18)
* [Cheat Code](https://www.guru99.com/python-json.html#19)

## Python to JSON (Encoding)

JSON Library of Python performs following translation of Python objects into JSON objects by default

|  |  |
| --- | --- |
| **Python** | **JSON** |
| dict | Object |
| list | Array |
| unicode | String |
| number - int, long | number – int |
| float | number – real |
| True | True |
| False | False |
| None | Null |

Converting Python data to JSON is called an Encoding operation. Encoding is done with the help of JSON library method – **dumps()**

**dumps()**method converts dictionary object of python into JSON string data format.

Now lets we perform our first encoding example with Python.

import json

x = {

"name": "Ken",

"age": 45,

"married": True,

"children": ("Alice","Bob"),

"pets": ['Dog'],

"cars": [

{"model": "Audi A1", "mpg": 15.1},

{"model": "Zeep Compass", "mpg": 18.1}

]

}

# sorting result in asscending order by keys:

sorted\_string = json.dumps(x, indent=4, sort\_keys=True)

print(sorted\_string)

Output:

{"person": {"name": "Kenn", "sex": "male", "age": 28}})

Let's create a JSON file of the dictionary using the same function **dump()**

# here we create new data\_file.json file with write mode using file i/o operation

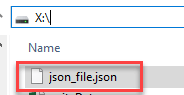
with open('json\_file.json', "w") as file\_write:

# write json data into file

json.dump(person\_data, file\_write)

Output:

Nothing to show…In your system json\_file.json is created you can check that file.

[](https://www.guru99.com/images/1/122818_1014_PythonJSONE1.png)

## JSON to Python (Decoding)

JSON string decoding is done with the help of inbuilt method **loads()** & **load()**of JSON library in Python. Here translation table show example of JSON objects to Python objects which are helpful to perform decoding in Python of JSON string.

|  |  |
| --- | --- |
| **JSON** | **Python** |
| Object | dict |
| Array | list |
| String | unicode |
| number – int | number - int, long |
| number – real | float |
| True | True |
| False | False |
| Null | None |

Let's see a basic example of decoding in Python with the help of **json.loads()** function,

import json # json library imported

# json data string

person\_data = '{ "person": { "name": "Kenn", "sex": "male", "age": 28}}'

# Decoding or converting JSON format in dictionary using loads()

dict\_obj = json.loads(person\_data)

print(dict\_obj)

# check type of dict\_obj

print("Type of dict\_obj", type(dict\_obj))

# get human object details

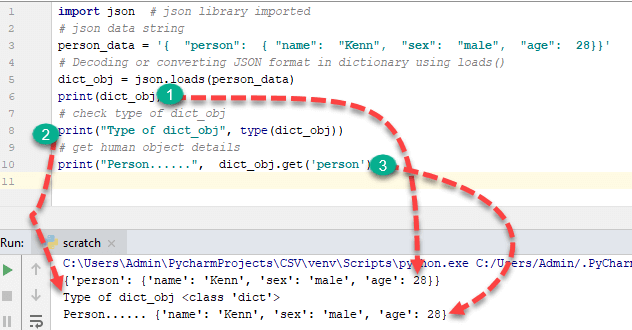
print("Person......", dict\_obj.get('person'))

**Output:**

{'person': {'name': 'Kenn', 'sex': 'male', 'age': 28}}

Type of dict\_obj <class 'dict'>

Person...... {'name': 'John', 'sex': 'male'}



### Decoding JSON File or Parsing JSON file in Python

**NOTE:**Decoding JSON file is File Input /Output (I/O) related operation. The JSON file must exist on your system at specified the location that you mention in your program.

Example,

import json

#File I/O Open function for read data from JSON File

with open('X:/json\_file.json') as file\_object:

# store file data in object

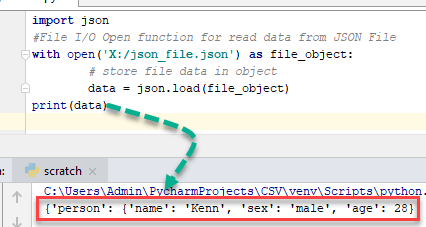
data = json.load(file\_object)

print(data)

**Here data** is a dictionary object of Python.

Output:

{'person': {'name': 'Kenn', 'sex': 'male', 'age': 28}}

[](https://www.guru99.com/images/1/122818_1014_PythonJSONE3.png)

### Compact Encoding in Python

When you need to reduce the size of your JSON file, you can use compact encoding in Python.

Example,

import json

# Create a List that contains dictionary

lst = ['a', 'b', 'c',{'4': 5, '6': 7}]

# separator used for compact representation of JSON.

# Use of ',' to identify list items

# Use of ':' to identify key and value in dictionary

compact\_obj = json.dumps(lst, separators=(',', ':'))

print(compact\_obj)

**Output:**

'["a", "b", "c", {"4": 5, "6": 7}]'

\*\* Here output of JSON is represented in a single line which is the most compact representation by removing the space character from compact\_obj \*\*

### Format JSON code (Pretty print)

* The aim is to write well-formatted code for human understanding. With the help of pretty printing, anyone can easily understand the code.
* Example,

import json

dic = { 'a': 4, 'b': 5 }

''' To format the code use of indent and 4 shows number of space and use of separator is not necessary but standard way to write code of particular function. '''

formatted\_obj = json.dumps(dic, indent=4, separators=(',', ': '))

print(formatted\_obj)

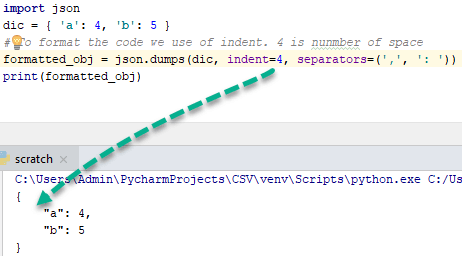
Output:

{

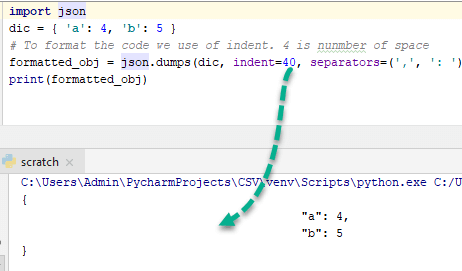
"a" : 4,

"b" : 5

}



To better understand this, change indent to 40 and observe the output-

[](https://www.guru99.com/images/1/122818_1014_PythonJSONE5.png)

**Ordering the JSON code:**

**sort\_keys** attribute in dumps() function's argument will sort the key in JSON in ascending order. The sort\_keys argument is a Boolean attribute. When it's true sorting is allowed otherwise not

Example,

import json

x = {

"name": "Ken",

"age": 45,

"married": True,

"children": ("Alice", "Bob"),

"pets": [ 'Dog' ],

"cars": [

{"model": "Audi A1", "mpg": 15.1},

{"model": "Zeep Compass", "mpg": 18.1}

],

}

# sorting result in asscending order by keys:

sorted\_string = json.dumps(x, indent=4, sort\_keys=True)

print(sorted\_string)

**Output:**

{

"age": 45,

"cars": [ {

"model": "Audi A1",

"mpg": 15.1

},

{

"model": "Zeep Compass",

"mpg": 18.1

}

],

"children": [ "Alice",

"Bob"

],

"married": true,

"name": "Ken",

"pets": [

"Dog"

]

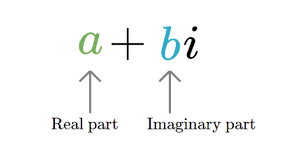
}

As you may observe the keys age, cars, children, etc are arranged in ascending order.

## Complex Object encoding of Python

A Complex object has two different parts that is

1. Real part
2. Imaginary part

[](https://www.guru99.com/images/1/122818_1014_PythonJSONE6.png)

**Example: 3 +2i**

Before performing encoding of a complex object, you need to check a variable is complex or not. You need to create a function which checks the value stored in a variable by using an instance method.

Let's create the specific function for check object is complex or eligible for encoding.

import json

# create function to check instance is complex or not

def complex\_encode(object):

# check using isinstance method

if isinstance(object, complex):

return [object.real, object.imag]

# raised error using exception handling if object is not complex

raise TypeError(repr(object) + " is not JSON serialized")

# perform json encoding by passing parameter

complex\_obj = json.dumps(4 + 5j, default=complex\_encode)

print(complex\_obj)

**Output:**

'[4.0, 5.0]'

## Complex JSON object decoding in Python

To decode complex object in JSON, use an object\_hook parameter which checks JSON string contains the complex object or not. Example,

import json

# function check JSON string contains complex object

def is\_complex(objct):

if '\_\_complex\_\_' in objct:

return complex(objct['real'], objct['img'])

return objct

# use of json loads method with object\_hook for check object complex or not

complex\_object =json.loads('{"\_\_complex\_\_": true, "real": 4, "img": 5}', object\_hook = is\_complex)

#here we not passed complex object so it's convert into dictionary

simple\_object =json.loads('{"real": 6, "img": 7}', object\_hook = is\_complex)

print("Complex\_object......",complex\_object)

print("Without\_complex\_object......",simple\_object)

Output:

Complex\_object...... (4+5j)

Without\_complex\_object...... {'real': 6, 'img': 7}

## Overview of JSON Serialization class JSONEncoder

JSONEncoder class is used for serialization of any Python object while performing encoding. It contains three different methods of encoding which are

* **default(o)** – Implemented in the subclass and return serialize object for **o** object.
* **encode(o)** – Same as json.dumps() method return JSON string of Python data structure.
* **iterencode(o)**– Represent string one by one and encode object o.

With the help of encode() method of JSONEncoder class, we can also encode any Python object.

# import JSONEncoder class from json

from json.encoder import JSONEncoder

colour\_dict = { "colour": ["red", "yellow", "green" ]}

# directly called encode method of JSON

JSONEncoder().encode(colour\_dict)

**Output:**

'{"colour": ["red", "yellow", "green"]}'

## Overview of JSON Deserialization class JSONDecoder

JSONDecoder class is used for deserialization of any Python object while performing decoding. It contains three different methods of decoding which are

* **default(o)** – Implemented in the subclass and return deserialized object **o** object.
* **decode(o)** – Same as json.loads() method return Python data structure of JSON string or data.
* **raw\_decode(o)**– Represent Python dictionary one by one and decode object o.

With the help of decode() method of JSONDecoder class, we can also decode JSON string.

import json

# import JSONDecoder class from json

from json.decoder import JSONDecoder

colour\_string = '{ "colour": ["red", "yellow"]}'

# directly called decode method of JSON

JSONDecoder().decode(colour\_string)

**Output:**

{'colour': ['red', 'yellow']}

### Decoding JSON data from URL: Real Life Example

We will fetch data of CityBike NYC (Bike Sharing System) from specified URL(<https://feeds.citibikenyc.com/stations/stations.json>) and convert into dictionary format.

Example,

NOTE:- Make sure requests library is already installed in your Python, If not then open Terminal or CMD and type

* (For Python 3 or above) **pip3 install requests**

import json

import requests

# get JSON string data from CityBike NYC using web requests library

json\_response= requests.get("https://feeds.citibikenyc.com/stations/stations.json")

# check type of json\_response object

print(type(json\_response.text))

# load data in loads() function of json library

bike\_dict = json.loads(json\_response.text)

#check type of news\_dict

print(type(bike\_dict))

# now get stationBeanList key data from dict

print(bike\_dict['stationBeanList'][0])

**Output:**

<class 'str'>

<class 'dict'>

{

'id': 487,

'stationName': 'E 20 St & FDR Drive',

'availableDocks': 24,

'totalDocks': 34,

'latitude': 40.73314259,

'longitude': -73.97573881,

'statusValue': 'In Service',

'statusKey': 1,

'availableBikes': 9,

'stAddress1': 'E 20 St & FDR Drive',

'stAddress2': '',

'city': '',

'postalCode': '',

'location': '',

'altitude': '',

'testStation': False,

'lastCommunicationTime': '2018-12-11 10:59:09 PM', 'landMark': ''

}

## Exceptions Related to JSON Library in Python:

* Class **json.JSONDecoderError** handles the exception related to decoding operation. and it's a subclass of **ValueError.**
* Exception - **json.JSONDecoderError(msg, doc)**
* Parameters of Exception are,
  + msg – Unformatted Error message
  + doc – JSON docs parsed
  + pos – start index of doc when it's failed
  + lineno – line no shows correspond to pos
  + colon – column no correspond to pos

Example,

import json

#File I/O Open function for read data from JSON File

data = {} #Define Empty Dictionary Object

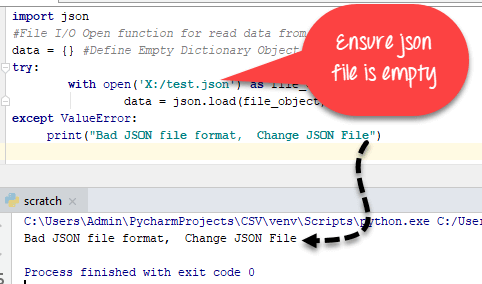
try:

with open('json\_file\_name.json') as file\_object:

data = json.load(file\_object)

except ValueError:

print("Bad JSON file format, Change JSON File")



## Infinite and NaN Numbers in Python

JSON Data Interchange Format (RFC – Request For Comments) doesn't allow Infinite or Nan Value but there is no restriction in Python- JSON Library to perform Infinite and Nan Value related operation. If JSON gets INFINITE and Nan datatype than it's converted it into literal.

Example,

import json

# pass float Infinite value

infinite\_json = json.dumps(float('inf'))

# check infinite json type

print(infinite\_json)

print(type(infinite\_json))

json\_nan = json.dumps(float('nan'))

print(json\_nan)

# pass json\_string as Infinity

infinite = json.loads('Infinity')

print(infinite)

# check type of Infinity

print(type(infinite))

Output:

Infinity

<class 'str'>

NaN

inf

<class 'float'>

## Repeated key in JSON String

RFC specifies the key name should be unique in a JSON object, but it's not mandatory. Python JSON library does not raise an exception of repeated objects in JSON. It ignores all repeated key-value pair and considers only last key-value pair among them.

* Example,

import json

repeat\_pair = '{"a": 1, "a": 2, "a": 3}'

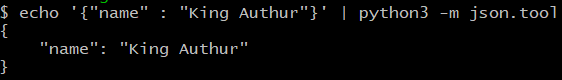
json.loads(repeat\_pair)

Output:

{'a': 3}

## CLI (Command Line Interface) with JSON in Python

**json.tool**provides the command line interface to validate JSON pretty-print syntax. Let's see an example of CLI

[](https://www.guru99.com/images/1/122818_1014_PythonJSONE8.png)

$ echo '{"name" : "Kings Authur" }' | python3 -m json.tool

Output:

{

"name": " Kings Authur "

}

## Advantages of JSON in Python

* Easy to move back between container and value (JSON to Python and Python to JSON)
* Human readable (Pretty-print) JSON Object
* Widely used in data handling.
* Doesn't have the same data structure in the single file.

## Implementation limitation of JSON in Python

* In deserializer of JSON range and prediction of a number
* The Maximum length of JSON string and arrays of JSON and nesting levels of object.

## Cheat Code

|  |  |
| --- | --- |
| json.dumps(person\_data) | Create JSON Object |
| json.dump(person\_data, file\_write) | Create JSON File using File I/O of Python |
| compact\_obj = json.dumps(data, separators=(',',':')) | Compact JSON Object by removing space character from JSON Object using separator |
| formatted\_obj = json.dumps(dic, indent=4, separators=(',', ': ')) | Formatting JSON code using Indent |
| sorted\_string = json.dumps(x, indent=4, sort\_keys=True) | Sorting JSON object key by alphabetic order |
| complex\_obj = json.dumps(4 + 5j, default=complex\_encode) | Python Complex Object encoding in JSON |
| JSONEncoder().encode(colour\_dict) | Use of JSONEncoder Class for Serialization |
| json.loads(data\_string) | Decoding JSON String in Python dictionary using json.loads() function |
| json.loads('{"\_\_complex\_\_": true, "real": 4, "img": 5}', object\_hook = is\_complex) | Decoding of complex JSON object to Python |
| JSONDecoder().decode(colour\_string) | Use of Decoding JSON to Python with Deserialization |

# Python with MySQL: Connect, Create Database, Table, Insert [Examples]

In order to work with MySQL using Python, you must have some knowledge of [SQL](https://www.guru99.com/sql.html)

Before diving deep, let's understand

## What is MySQL?

MySQL is an Open-Source database and one of the best type of RDBMS (Relational Database Management System). Co-founder of MySQLdb is Michael Widenius's, and also MySQL name derives from the daughter of Michael.

## How to Install MySQL

**Install MySQL in Linux/Unix:**

Download RPM package for Linux/Unix from Official site: <https://www.mysql.com/downloads/>

In terminal use following command

rpm -i <Package\_name>

Example rpm -i MySQL-5.0.9.0.i386.rpm

To check in Linux

mysql --version

**Install MySQL in Windows**

Download MySQL database exe from [official site](https://www.mysql.com/downloads/) and install as usual normal installation of software in Windows. Refer this [tutorial](https://www.guru99.com/introduction-to-mysql-workbench.html), for a step by step guide

## Install MySQL Connector Library for Python

For Python 2.7 or lower install using pip as:

pip install mysql-connector

For Python 3 or higher version install using pip3 as:

pip3 install mysql-connector

## Test the MySQL Database connection with Python

To test database connection here we use pre-installed MySQL connector and pass credentials into **connect()** function like host, username and password.

Syntax to access MySQL with Python:

import mysql.connector

db\_connection = mysql.connector.connect(

host="hostname",

user="username",

passwd="password"

)

Example,

import mysql.connector

db\_connection = mysql.connector.connect(

host="localhost",

user="root",

passwd="root"

)

print(db\_connection)

Output:

<mysql.connector.connection.MySQLConnection object at 0x000002338A4C6B00>

Here output shows the connection created successfully.

## Creating Database in MySQL using Python

Syntax to Create new database in SQL is

CREATE DATABASE "database\_name"

Now we create database using Python in MySQL

import mysql.connector

db\_connection = mysql.connector.connect(

host= "localhost",

user= "root",

passwd= "root"

)

# creating database\_cursor to perform SQL operation

db\_cursor = db\_connection.cursor()

# executing cursor with execute method and pass SQL query

db\_cursor.execute("CREATE DATABASE my\_first\_db")

# get list of all databases

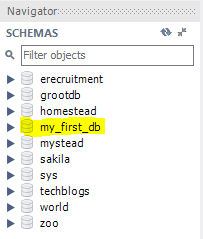
db\_cursor.execute("SHOW DATABASES")

#print all databases

for db in db\_cursor:

print(db)

Output:



Here above image shows the **my\_first\_db** database is created

## Create a Table in MySQL with Python

Let's create a simple table "student" which has two columns.

SQL Syntax:

CREATE TABLE student (id INT, name VARCHAR(255))

**Example:**

import mysql.connector

db\_connection = mysql.connector.connect(

host="localhost",

user="root",

passwd="root",

database="my\_first\_db"

)

db\_cursor = db\_connection.cursor()

#Here creating database table as student'

db\_cursor.execute("CREATE TABLE student (id INT, name VARCHAR(255))")

#Get database table'

db\_cursor.execute("SHOW TABLES")

for table in db\_cursor:

print(table)

Output:

('student',)

## Create a Table with Primary Key

Let's create an **Employee** table with three different columns. We will add a primary key in **id** column with AUTO\_INCREMENT constraint

SQL Syntax,

CREATE TABLE employee(id INT AUTO\_INCREMENT PRIMARY KEY, name VARCHAR(255), salary INT(6))

Example,

import mysql.connector

db\_connection = mysql.connector.connect(

host="localhost",

user="root",

passwd="root",

database="my\_first\_db"

)

db\_cursor = db\_connection.cursor()

#Here creating database table as employee with primary key

db\_cursor.execute("CREATE TABLE employee(id INT AUTO\_INCREMENT PRIMARY KEY, name VARCHAR(255), salary INT(6))")

#Get database table

db\_cursor.execute("SHOW TABLES")

for table in db\_cursor:

print(table)

Output:

('employee',) ('student',)

## ALTER table in MySQL with Python

Alter command is used for modification of Table structure in SQL. Here we will alter **Student**table and add a primary key to the **id** field.

SQL Syntax,

ALTER TABLE student MODIFY id INT PRIMARY KEY

Example,

import mysql.connector

db\_connection = mysql.connector.connect(

host="localhost",

user="root",

passwd="root",

database="my\_first\_db"

)

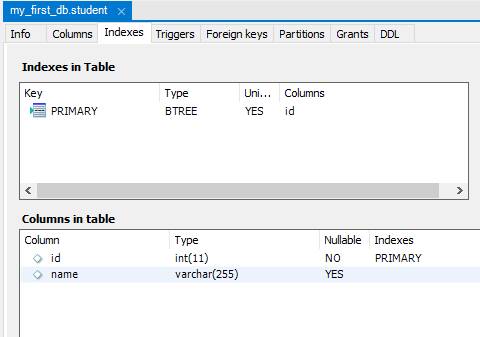
db\_cursor = db\_connection.cursor()

#Here we modify existing column id

db\_cursor.execute("ALTER TABLE student MODIFY id INT PRIMARY KEY")

Output:

Here below you can see the **id**column is modified.

[](https://www.guru99.com/images/1/030819_0707_PythonwithM2.png)

## Insert Operation with MySQL in Python:

Let's perform insertion operation in MySQL Database table which we already create. We will insert data oi STUDENT table and EMPLOYEE table.

SQL Syntax,

INSERT INTO student (id, name) VALUES (01, "John")

INSERT INTO employee (id, name, salary) VALUES(01, "John", 10000)

Example,

import mysql.connector

db\_connection = mysql.connector.connect(

host="localhost",

user="root",

passwd="root",

database="my\_first\_db"

)

db\_cursor = db\_connection.cursor()

student\_sql\_query = "INSERT INTO student(id,name) VALUES(01, 'John')"

employee\_sql\_query = " INSERT INTO employee (id, name, salary) VALUES (01, 'John', 10000)"

#Execute cursor and pass query as well as student data

db\_cursor.execute(student\_sql\_query)

#Execute cursor and pass query of employee and data of employee

db\_cursor.execute(employee\_sql\_query)

db\_connection.commit()

print(db\_cursor.rowcount, "Record Inserted")

Output:

2 Record Inserted

# PyUnit Tutorial: Python Unit Testing Framework (with Example)

### What is Unit Testing?

[Unit Testing](https://www.guru99.com/unit-testing-guide.html) in[Python](https://www.guru99.com/python-tutorials.html)is done to identify bugs early in the development stage of the application when bugs are less recurrent and less expensive to fix.

A unit test is a scripted code level test designed in Python to verify a small "unit" of functionality. Unit test is an object oriented framework based around test fixtures.

## Python Unit Testing Techniques

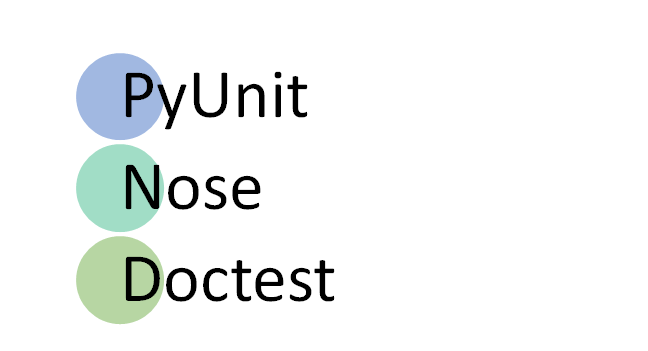
Python Unit Testing mainly involves testing a particular module without accessing any dependent code. Developers can use techniques like stubs and mocks to separate code into "units" and run unit level testing on the individual pieces.

* **Test-Driven Development TDD:**Unit Testing should be done along with the Python, and for that developers use Test-Driven Development method. In TDD method, you first design Python Unit tests and only then you carry on writing the code that will implement this feature.
* **Stubs and Mocks:**They are two main techniques that simulate fake methods that are being tested. A **Stub**is used to fill in some dependency required for unit test to run correctly. A **Mock** on the other hand is a fake object which runs the tests where we put assert.

The intentions of both methods are same to eliminate testing all the dependencies of a class or function.

## Python Unit Testing Framework

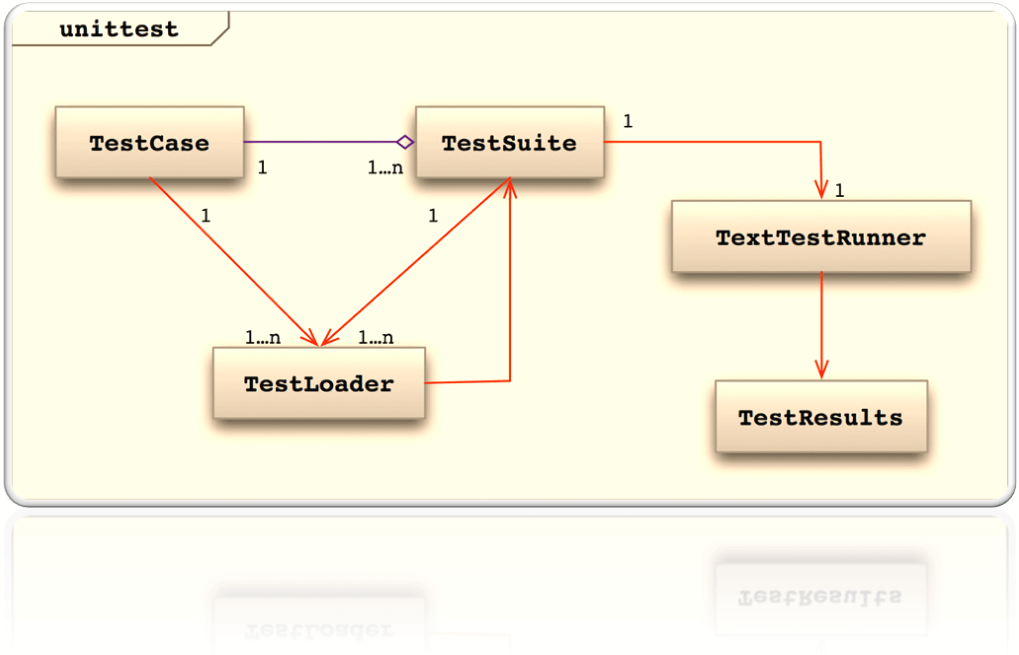
To make the Unit Testing process easier and improve the quality of your project, it is recommended the Python Unit Testing Framework. The Unit Testing framework includes



* **PyUnit:**PyUnit supports fixtures, test cases, test suites and a test runner for the automated testing of the code. In PyUnit, you can organize test cases into suites with the same fixtures
* **Nose:**Nose's built in plug-ins helps you with output capture, code coverage, doctests, etc. Nose syntax is pretty simpler and reduces the barriers to writing tests. It extends Python unittest to make testing easier.
* **Doctest :**Doctest testing script goes in docstring with small function at the bottom of file. Doctest allows you to test your code by running examples included in the documentation and verifying that they returned the expected results. The use-case of doctest is less detailed and don't catch special cases. They are useful as an expressive documentation of the main use case of a module and its components.

## Unit Testing with PyUnit

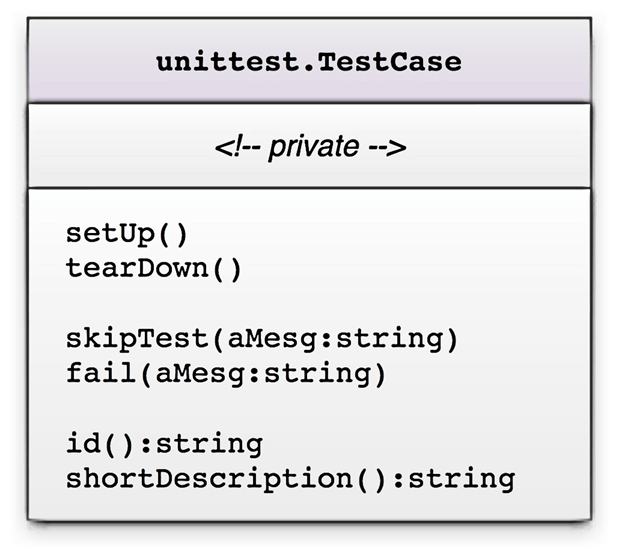
Pyunit is a Python port of JUnit. As a part of Pyunit, in the unittest module there are five key classes.



* **TestCase class**: The TestCase class bears the test routines and delivers hooks for making each routine and cleaning up thereafter
* **TestSuite class**: It caters as a collection container, and it can possess multiple testcase objects and multiple testsuites objects
* **TestLoader class**: This class loads test cases and suites defined locally or from an external file. It emits a testsuite objects that posseses those suites and cases
* **TextTestRunner class**: To run the tests it caters a standard platform to execute the tests
* **The TestResults class**: It offers a standard container for the test results

## Designing a test case for Python Testing using PyUnit

A unit test provides a base class, test case, which may be used to create new test cases. For designing the test case, there are three sets of methods used are



unittest.TestCase

setUp()

teardown()

skipTest(aMesg:string)

fail(aMesg:string)

id():string

shortDescription():string

In the first set are the pre and post test hooks. The setup() method begins before each test routine, the teardown() after the routine.

The second set of method controls test execution. Both methods take a message string as input, and both cancel an ongoing test. But the skiptest() method aborts the current test while the fail() method fails it completely.

The last or third method help determining the test. The method id() returns a string consisting of the name of the testcase object and of the test routine. And the method shortDescription() returns the docstr comment at the initiation of each test routine.

## Advantages of using Python Unit testing

* It helps you to detect bugs early in the development cycle
* It helps you to write better programs
* It syncs easily with other testing methods and tools
* It will have many fewer bugs
* It is easier to modify in future with very less consequence

# Python 2 vs Python 3: Key Differences

## What is Python 2?

Python 2 made code development process easier than earlier versions. It implemented technical details of Python Enhancement Proposal (PEP). Python 2.7 (last version in 2.x ) is no longer under development and in 2020 will be discontinued.

In this tutorial, you will learn,

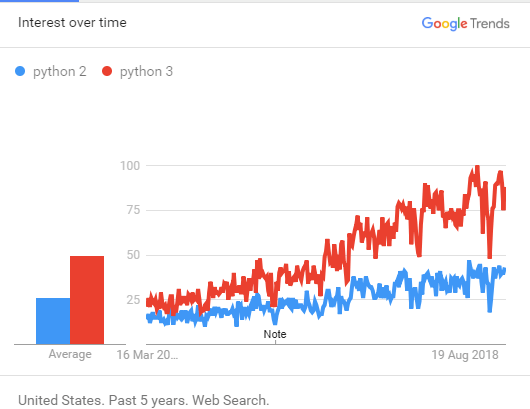
* [What is Python 2?](https://www.guru99.com/python-2-vs-python-3.html#1)
* [What is Python 3?](https://www.guru99.com/python-2-vs-python-3.html#2)
* [Why should you learn Python 2?](https://www.guru99.com/python-2-vs-python-3.html#3)
* [Why should you use Python 3?](https://www.guru99.com/python-2-vs-python-3.html#4)
* [History of Python 2](https://www.guru99.com/python-2-vs-python-3.html#5)
* [History of Python 3](https://www.guru99.com/python-2-vs-python-3.html#6)
* [Key Differences Between Python 2 and Python 3](https://www.guru99.com/python-2-vs-python-3.html#7)
* [Python 2 vs. Python 3 Example Code](https://www.guru99.com/python-2-vs-python-3.html#8)
* [Which Python Version to Use?](https://www.guru99.com/python-2-vs-python-3.html#9)

## What is Python 3?

On December 2008, Python released version 3.0. This version was mainly released to fix problems which exist in Python 2. The nature of these change is such that Python 3 was incompatible with Python 2. It is **backward incompatible** Some features of Python 3 have been backported to Python 2.x versions to make the migration process easy in Python 3.

As a result, for any organization who was using Python 2.x version, migrating their project to 3.x needed lots of changes. These changes not only relate to projects and applications but also all the libraries that form part of the Python ecosystem.

## Why should you learn Python 2?

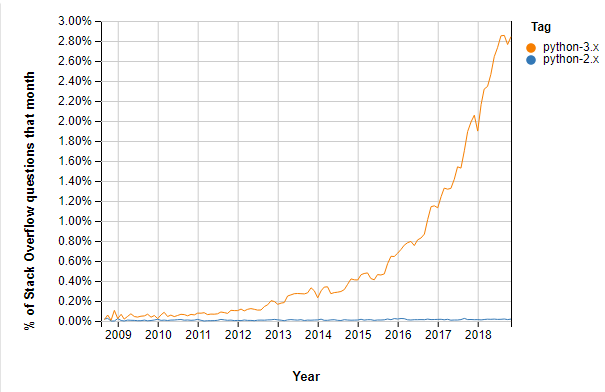


Google Trends Python 2 vs. Python 3

Although, Python 2 is an old open source version here are where you still need to learn Python 2:

* To become a DevOps engineer and you need to work with configurations management tools like puppet or ansible. Here, you need to work with both of these versions.
* If your company's code written in Python 2, you will require to learn to work with that
* If your development team is working on a project that depends on specific third-party libraries or software which you are not able to port to Python 3, then Python 2 is the only option available for you.

## Why should you use Python 3?



Stack Overflow Questions Python 2 vs. Python 3

Here, are prime reasons for using Python 3.x versions:

* Python 3 supports modern techniques like AI, machine learning, and data science
* Python 3 is supported by a large Python developer's community. Getting support is easy.
* Its easier to learn Python language compared to earlier versions.
* Offers Powerful toolkit and libraries
* Mixable with other languages

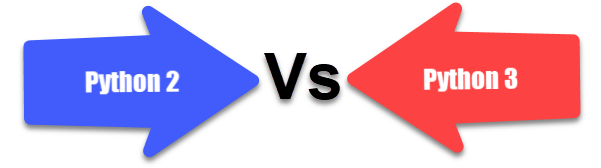
## History of Python 2

* Python 2.0 - October 16, 2000
* Python 2.1 - April 17, 2001
* Python 2.2 - December 21, 2001
* Python 2.3 - July 29, 2003
* Python 2.4 - November 30, 2004
* Python 2.5 - September 19, 2006
* Python 2.6 - October 1, 2008
* Python 2.7-July 3, 2010

## History of Python 3

* Python 3.0 - December 3, 2008
* Python 3.1 - June 27, 2009
* Python 3.2 - February 20, 2011
* Python 3.3 - September 29, 2012
* Python 3.4-March 16, 2014
* Python 3.5 - September 13, 2015
* Python 3.6- October 2016
* Python 3.7- June 2018.

## Key Differences Between Python 2 and Python 3



| **Basis of comparison** | **Python 3** | **Python 2** |
| --- | --- | --- |
| Release Date | 2008 | 2000 |
| Function print | print ("hello") | print "hello" |
| Division of Integers | Whenever two integers are divided, you get a float value | When two integers are divided, you always provide integer value. |
| Unicode | In Python 3, default storing of strings is Unicode. | To store Unicode string value, you require to define them with "u". |
| Syntax | The syntax is simpler and easily understandable. | The syntax of Python 2 was comparatively difficult to understand. |
| Rules of ordering Comparisons | In this version, Rules of ordering comparisons have been simplified. | Rules of ordering comparison are very complex. |
| Iteration | The new Range() function introduced to perform iterations. | In Python 2, the xrange() is used for iterations. |
| Exceptions | It should be enclosed in parenthesis. | It should be enclosed in notations. |
| Leak of variables | The value of variables never changes. | The value of the global variable will change while using it inside for-loop. |
| Backward compatibility | Not difficult to port python 2 to python 3 but it is never reliable. | Python version 3 is not backwardly compatible with Python 2. |
| Library | Many recent developers are creating libraries which you can only use with Python 3. | Many older libraries created for Python 2 is not forward-compatible. |

## Python 2 vs. Python 3 Example Code

**Python 3**

def main():

print("Hello World!")

if \_\_name\_\_== "\_\_main\_\_":

main()

**Python 2**

def main():

print "Hello World!"

if \_\_name\_\_== "\_\_main\_\_":

main()

## Which Python Version to Use?

When it comes to Python version 2 vs. 3 today, Python 3 is the outright winner. That's because Python 2 won't be available after 2020. Mass Python 3 adoption is the clear direction of the future.

After considering declining support for Python 2 programming language and added benefits from upgrades to Python 3, it is always advisable for a new developer to select Python version 3. However, if a job demands Python 2 capabilities, that would be an only compelling reason to use this version.

**Facebook Login using Python: FB Login Example**

In order to log into Facebook using Python, you need to use Selenium (a web automation tool). Selenium can automate and control a browser and click, fill text, submit buttons that appear on various websites.

To log in to Facebook, we will use a Python Script that drives Selenium. The Selenium Python Script will

* Step 1) Open Firefox
* Step 2) Navigate to Facebook
* Step 3) Search & Enter the Email or Phone field & Enter Password
* Step 4) Click Login

Here is a quick video on the system will work.

Note: You can configure Selenium to use any browser like Chrome, Safari, IE, etc. In this tutorial, we will use FireFox

What do you need to Install?

* Ensure you have Selenium installed on your PC. Refer this [link](https://www.guru99.com/installing-selenium-webdriver.html) to learn to install Selenium
* Use [this](https://www.guru99.com/selenium-python.html) link to install Python for Selenium

Code to Login into Facebook using Python

from selenium import webdriver

from selenium.webdriver.support.ui import WebDriverWait

# Step 1) Open Firefox

browser = webdriver.Firefox()

# Step 2) Navigate to Facebook

browser.get("http://www.facebook.com")

# Step 3) Search & Enter the Email or Phone field & Enter Password

username = browser.find\_element\_by\_id("email")

password = browser.find\_element\_by\_id("pass")

submit = browser.find\_element\_by\_id("loginbutton")

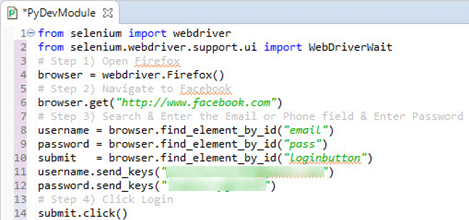
username.send\_keys("[you@email.com](mailto:you@email.com)

")

password.send\_keys("yourpassword")

# Step 4) Click Login

submit.click()

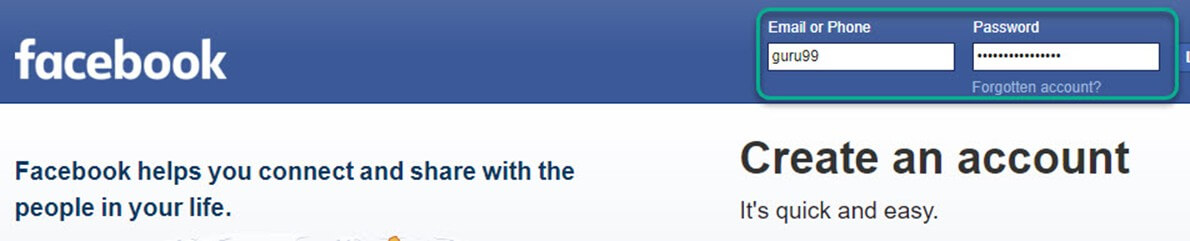
[](https://www.guru99.com/images/1/102219_1457_FacebookLog1.png)

**Explanation of the code**

* **Code line 1**: From selenium module import webdriver
* **Code line 2**: From selenium module import Keys
* **Code line 4**: In this line, we are initializing "FireFox" by making an object of it.
* **Code line 6**: The "browser.get method" will nagivagte to a page given by the URL. WebDriver wait until the page has been completely loaded (that is, the "onload" occasion has let go), before returning control to your test or script.
* **Code line 8**: In this line, we are finding the element of the textbox where the "email" has to be written.
* **Code line 9**: In this line, we are finding the element of the textbox where the "password" has to be written.
* **Code line 10**: In this line, we are finding the submit button element which we need to click
* **Code line 11**: Now we are sending the values to the email section
* **Code line 12**: Sending values to the password section
* Code line 14: Click on the "Submit" button

**Sample Output**

The values of the username "guru99" and password entered.



The Facebook page will login with email and password. Page opened (see image below)



### FAQ

**What else can I use except Selenium to login to Facebook using Python?**

You can use the Facebook [API](https://developers.facebook.com/docs/facebook-login/) to write Python Scripts to log into Facebook from your application

**Is there an alternative to using Selenium for Login to Facebook using Python?**

There are many alternatives to Selenium that you can check [here](https://www.guru99.com/selenium-alternatives.html) Though some of the tools may not support Python

# PyTest Tutorial: What is, Install, Fixture, Assertions

## What is PyTest?

Pytest is a testing framework which allows us to write test codes using python. You can write code to test anything like database , API, even UI if you want. But pytest is mainly being used in industry to write tests for APIs.

## Why use PyTest?

Some of the advantages of pytest are

* Very easy to start with because of its simple and easy syntax.
* Can run tests in parallel.
* Can run a specific test or a subset of tests
* Automatically detect tests
* Skip tests
* Open source

In this tutorial, you will learn:

* [What is PyTest?](https://www.guru99.com/pytest-tutorial.html#1)
* [Why use PyTest?](https://www.guru99.com/pytest-tutorial.html#2)
* [How to install PyTest](https://www.guru99.com/pytest-tutorial.html#3)
* [First Basic PyTest](https://www.guru99.com/pytest-tutorial.html#4)
* [Assertions in PyTest](https://www.guru99.com/pytest-tutorial.html#5)
* [How pytest identifies the test files and test methods](https://www.guru99.com/pytest-tutorial.html#6)
* [Run multiple tests from a specific file and multiple files.](https://www.guru99.com/pytest-tutorial.html#7)
* [Run a subset of entire test](https://www.guru99.com/pytest-tutorial.html#8)
* [Running tests in parallel](https://www.guru99.com/pytest-tutorial.html#9)
* [Pytest fixtures](https://www.guru99.com/pytest-tutorial.html#10)
* [Parameterized tests](https://www.guru99.com/pytest-tutorial.html#11)
* [Xfail / Skip tests](https://www.guru99.com/pytest-tutorial.html#12)
* [Results XML](https://www.guru99.com/pytest-tutorial.html#13)
* [A pytest framework testing an API](https://www.guru99.com/pytest-tutorial.html#14)

## How to install PyTest

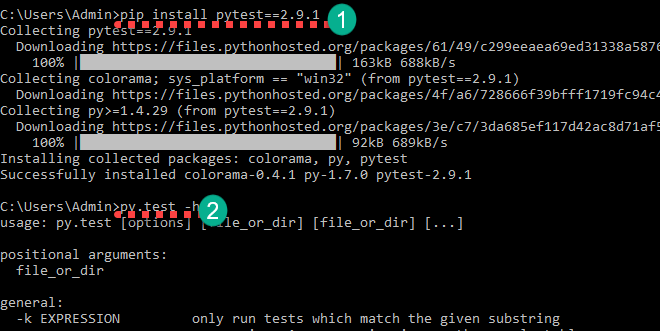
**Step 1)** You can install pytest by

pip install pytest==2.9.1

Once the installation is complete you can confirm it with by

py.test -h

This will display the help

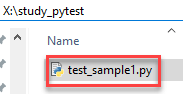


## First Basic PyTest

Create a folder study\_pytest. We are going to create our test files inside this folder.

Please navigate to that folder in your command line.

Create a file named test\_sample1.py inside the folder

[](https://www.guru99.com/images/1/011019_1320_PyTestTutor2.png)

Add the below code into it and save

import pytest

def test\_file1\_method1():

x=5

y=6

assert x+1 == y,"test failed"

assert x == y,"test failed"

def test\_file1\_method2():

x=5

y=6

assert x+1 == y,"test failed"

Run the test using the command

py.test

You'll get output as

test\_sample1.py F.

============================================== FAILURES ========================================

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ test\_sample1 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

def test\_file1\_method1():

x=5

y=6

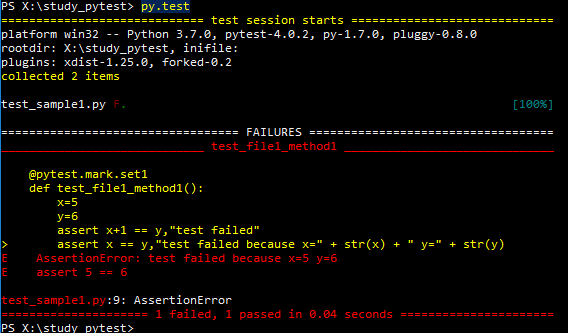
assert x+1 == y,"test failed"

> assert x == y,"test failed"

E AssertionError: test failed

E assert 5 == 6

test\_sample1.py:6: AssertionError



Here in test\_sample1.py F.

F says failure

Dot(.) says success.

In the failures section, you can see the failed method(s) and the line of failure. Here x==y means 5==6 which is false.

## Assertions in PyTest

Assertions are checks that return either True or False status. In pytest, if an assertion fails in a test method, then that method execution is stopped there. The remaining code in that test method is not executed, and pytest will continue with the next test method.

**Examples:**

assert "hello" == "Hai" is an assertion failure.

assert 4==4 is a successful assertion

assert True is a successful assertion

assert False is an assertion failure.

Consider

assert x == y,"test failed because x=" + str(x) + " y=" + str(y)

Place this code in test\_file1\_method1() instead of the assertion

assert x == y,"test failed"

Running the test will give the failure as **AssertionError: test failed x=5 y=6**

## How pytest identifies the test files and test methods

By default pytest only identifies the file names starting with **test\_** or ending with **\_test** as the test files. We can explicitly mention other filenames though (explained later). Pytest requires the test method names to start with **"test**." All other method names will be ignored even if we explicitly ask to run those methods.

See some examples of valid and invalid pytest file names

test\_login.py - valid

login\_test.py - valid

testlogin.py -invalid

logintest.py -invalid

Note: Yes we can explicitly ask pytest to pick testlogin.py and logintest.py

See some examples of valid and invalid pytest test methods

def test\_file1\_method1(): - valid

def testfile1\_method1(): - valid

def file1\_method1(): - invalid

Note: Even if we explicitly mention file1\_method1() pytest will not run this method.

## Run multiple tests from a specific file and multiple files.

Currently, inside the folder study\_pytest, we have a file test\_sample1.py. Suppose we have multiple files , say test\_sample2.py , test\_sample3.py. To run all the tests from all the files in the folder and subfolders we need to just run the pytest command.

py.test

This will run all the filenames starting with test\_ and the filenames ending with \_test in that folder and subfolders under that folder.

To run tests only from a specific file, we can use py.test <filename>

py.test test\_sample1.py

## Run a subset of entire test

Sometimes we don't want to run the entire test suite. Pytest allows us to run specific tests. We can do it in 2 ways

* Grouping of test names by substring matching
* Grouping of tests by markers

We already have test\_sample1.py. Create a file test\_sample2.py and add the below code into it

def test\_file2\_method1():

x=5

y=6

assert x+1 == y,"test failed"

assert x == y,"test failed because x=" + str(x) + " y=" + str(y)

def test\_file2\_method2():

x=5

y=6

assert x+1 == y,"test failed"

So we have currently

* test\_sample1.py
  + test\_file1\_method1()
  + test\_file1\_method2()
* test\_sample2.py
  + test\_file2\_method1()
  + test\_file2\_method2()

**Option 1) Run tests by substring matching**

Here to run all the tests having method1 in its name we have to run

py.test -k method1 -v

-k <expression> is used to represent the substring to match

-v increases the verbosity

So running py.test -k method1 -v will give you the following result

test\_sample2.py::test\_file2\_method1 FAILED

test\_sample1.py::test\_file1\_method1 FAILED

============================================== FAILURES ==============================================

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ test\_file2\_method1 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

def test\_file2\_method1():

x=5

y=6

assert x+1 == y,"test failed"

> assert x == y,"test failed because x=" + str(x) + " y=" + str(y)

E AssertionError: test failed because x=5 y=6

E assert 5 == 6

test\_sample2.py:5: AssertionError

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ test\_file1\_method1 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

@pytest.mark.only

def test\_file1\_method1():

x=5

y=6

assert x+1 == y,"test failed"

> assert x == y,"test failed because x=" + str(x) + " y=" + str(y)

E AssertionError: test failed because x=5 y=6

E assert 5 == 6

test\_sample1.py:8: AssertionError

================================= 2 tests deselected by '-kmethod1' ==================================

=============================== 2 failed, 2 deselected in 0.02 seconds ===============================

Here you can see towards the end **2 tests deselected by '-kmethod1'**which are test\_file1\_method2 and test\_file2\_method2

Try running with various combinations like:-

py.test -k method -v - will run all the four methods

py.test -k methods -v – will not run any test as there is no test name matches the substring 'methods'

**Option 2) Run tests by markers**

Pytest allows us to set various attributes for the test methods using pytest markers, @pytest.mark . To use markers in the test file, we need to import pytest on the test files.

Here we will apply different marker names to test methods and run specific tests based on marker names. We can define the markers on each test names by using

@pytest.mark.<name>.

We are defining markers set1 and set2 on the test methods, and we will run the test using the marker names. Update the test files with the following code

test\_sample1.py

import pytest

@pytest.mark.set1

def test\_file1\_method1():

x=5

y=6

assert x+1 == y,"test failed"

assert x == y,"test failed because x=" + str(x) + " y=" + str(y)

@pytest.mark.set2

def test\_file1\_method2():

x=5

y=6

assert x+1 == y,"test failed"

test\_sample2.py

import pytest

@pytest.mark.set1

def test\_file2\_method1():

x=5

y=6

assert x+1 == y,"test failed"

assert x == y,"test failed because x=" + str(x) + " y=" + str(y)

@pytest.mark.set1

def test\_file2\_method2():

x=5

y=6

assert x+1 == y,"test failed"

We can run the marked test by

py.test -m <name>

-m <name> mentions the marker name

Run py.test -m set1.This will run the methods test\_file1\_method1, test\_file2\_method1, test\_file2\_method2.

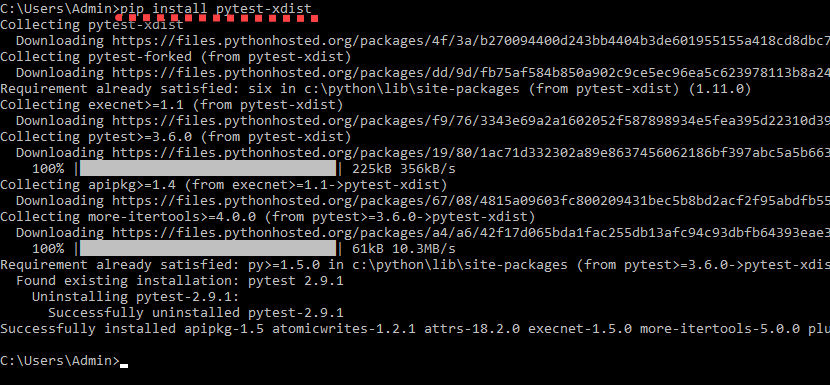
Running py.test -m set2 will run test\_file1\_method2.

## Running tests in parallel

Usually, a test suite will have multiple test files and hundreds of test methods which will take a considerable amount of time to execute. Pytest allows us to run tests in parallel.

For that we need to first install pytest-xdist by running

pip install pytest-xdist



You can run tests now by

py.test -n 4

-n <num> runs the tests by using multiple workers. In the above command, there will be 4 workers to run the test.

## Pytest fixtures

Fixtures are used when we want to run some code before every test method. So instead of repeating the same code in every test we define fixtures. Usually, fixtures are used to initialize database connections, pass the base , etc

A method is marked as a fixture by marking with

@pytest.fixture

A test method can use a fixture by mentioning the fixture as an input parameter.

Create a new file test\_basic\_fixture.py with following code

import pytest

@pytest.fixture

def supply\_AA\_BB\_CC():

aa=25

bb =35

cc=45

return [aa,bb,cc]

def test\_comparewithAA(supply\_AA\_BB\_CC):

zz=35

assert supply\_AA\_BB\_CC[0]==zz,"aa and zz comparison failed"

def test\_comparewithBB(supply\_AA\_BB\_CC):

zz=35

assert supply\_AA\_BB\_CC[1]==zz,"bb and zz comparison failed"

def test\_comparewithCC(supply\_AA\_BB\_CC):

zz=35

assert supply\_AA\_BB\_CC[2]==zz,"cc and zz comparison failed"

Here

* We have a fixture named supply\_AA\_BB\_CC. This method will return a list of 3 values.
* We have 3 test methods comparing against each of the values.

Each of the test function has an input argument whose name is matching with an available fixture. Pytest then invokes the corresponding fixture method and the returned values will be stored in the input argument , here the list [25,35,45]. Now the list items are being used in test methods for the comparison.

Now run the test and see the result

py.test test\_basic\_fixture

test\_basic\_fixture.py::test\_comparewithAA FAILED

test\_basic\_fixture.py::test\_comparewithBB PASSED

test\_basic\_fixture.py::test\_comparewithCC FAILED

============================================== FAILURES ==============================================

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ test\_comparewithAA \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

supply\_AA\_BB\_CC = [25, 35, 45]

def test\_comparewithAA(supply\_AA\_BB\_CC):

zz=35

> assert supply\_AA\_BB\_CC[0]==zz,"aa and zz comparison failed"

E AssertionError: aa and zz comparison failed

E assert 25 == 35

test\_basic\_fixture.py:10: AssertionError

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ test\_comparewithCC \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

supply\_AA\_BB\_CC = [25, 35, 45]

def test\_comparewithCC(supply\_AA\_BB\_CC):

zz=35

> assert supply\_AA\_BB\_CC[2]==zz,"cc and zz comparison failed"

E AssertionError: cc and zz comparison failed

E assert 45 == 35

test\_basic\_fixture.py:16: AssertionError

================================= 2 failed, 1 passed in 0.05 seconds =================================

The test test\_comparewithBB is passed since zz=BB=35, and the remaining 2 tests are failed.

The fixture method has a scope only within that test file it is defined. If we try to access the fixture in some other test file , we will get an error saying fixture **'supply\_AA\_BB\_CC' not found**for the test methods in other files.

To use the same fixture against multiple test files, we will create fixture methods in a file called conftest.py.

Let's see this by the below example. Create 3 files conftest.py, test\_basic\_fixture.py, test\_basic\_fixture2.py with the following code

conftest.py

import pytest

@pytest.fixture

def supply\_AA\_BB\_CC():

aa=25

bb =35

cc=45

return [aa,bb,cc]

test\_basic\_fixture.py

import pytest

def test\_comparewithAA(supply\_AA\_BB\_CC):

zz=35

assert supply\_AA\_BB\_CC[0]==zz,"aa and zz comparison failed"

def test\_comparewithBB(supply\_AA\_BB\_CC):

zz=35

assert supply\_AA\_BB\_CC[1]==zz,"bb and zz comparison failed"

def test\_comparewithCC(supply\_AA\_BB\_CC):

zz=35

assert supply\_AA\_BB\_CC[2]==zz,"cc and zz comparison failed"

test\_basic\_fixture2.py

import pytest

def test\_comparewithAA\_file2(supply\_AA\_BB\_CC):

zz=25

assert supply\_AA\_BB\_CC[0]==zz,"aa and zz comparison failed"

def test\_comparewithBB\_file2(supply\_AA\_BB\_CC):

zz=25

assert supply\_AA\_BB\_CC[1]==zz,"bb and zz comparison failed"

def test\_comparewithCC\_file2(supply\_AA\_BB\_CC):

zz=25

assert supply\_AA\_BB\_CC[2]==zz,"cc and zz comparison failed"

pytest will look for the fixture in the test file first and if not found it will look in the conftest.py

Run the test by py.test -k test\_comparewith -v to get the result as below

test\_basic\_fixture.py::test\_comparewithAA FAILED

test\_basic\_fixture.py::test\_comparewithBB PASSED

test\_basic\_fixture.py::test\_comparewithCC FAILED

test\_basic\_fixture2.py::test\_comparewithAA\_file2 PASSED

test\_basic\_fixture2.py::test\_comparewithBB\_file2 FAILED

test\_basic\_fixture2.py::test\_comparewithCC\_file2 FAILED

## Parameterized tests

The purpose of parameterizing a test is to run a test against multiple sets of arguments. We can do this by @pytest.mark.parametrize.

We will see this with the below example. Here we will pass 3 arguments to a test method. This test method will add the first 2 arguments and compare it with the 3rd argument.

Create the test file test\_addition.py with the below code

import pytest

@pytest.mark.parametrize("input1, input2, output",[(5,5,10),(3,5,12)])

def test\_add(input1, input2, output):

assert input1+input2 == output,"failed"

Here the test method accepts 3 arguments- input1, input2, output. It adds input1 and input2 and compares against the output.

Let's run the test by py.test -k test\_add -v and see the result

test\_addition.py::test\_add[5-5-10] PASSED

test\_addition.py::test\_add[3-5-12] FAILED

============================================== FAILURES ==============================================

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ test\_add[3-5-12] \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

input1 = 3, input2 = 5, output = 12

@pytest.mark.parametrize("input1, input2, output",[(5,5,10),(3,5,12)])

def test\_add(input1, input2, output):

> assert input1+input2 == output,"failed"

E AssertionError: failed

E assert (3 + 5) == 12

test\_addition.py:5: AssertionError

You can see the tests ran 2 times – one checking 5+5 ==10 and other checking 3+5 ==12

test\_addition.py::test\_add[5-5-10] PASSED

test\_addition.py::test\_add[3-5-12] FAILED

## Xfail / Skip tests

There will be some situations where we don't want to execute a test, or a test case is not relevant for a particular time. In those situations, we have the option to xfail the test or skip the tests

The xfailed test will be executed, but it will not be counted as part failed or passed tests. There will be no traceback displayed if that test fails. We can xfail tests using

@pytest.mark.xfail.

Skipping a test means that the test will not be executed. We can skip tests using

@pytest.mark.skip.

Edit the test\_addition.py with the below code

import pytest

@pytest.mark.skip

def test\_add\_1():

assert 100+200 == 400,"failed"

@pytest.mark.skip

def test\_add\_2():

assert 100+200 == 300,"failed"

@pytest.mark.xfail

def test\_add\_3():

assert 15+13 == 28,"failed"

@pytest.mark.xfail

def test\_add\_4():

assert 15+13 == 100,"failed"

def test\_add\_5():

assert 3+2 == 5,"failed"

def test\_add\_6():

assert 3+2 == 6,"failed"

Here

* test\_add\_1 and test\_add\_2 are skipped and will not be executed.
* test\_add\_3 and test\_add\_4 are xfailed. These tests will be executed and will be part of xfailed(on test failure) or xpassed(on test pass) tests. There won't be any traceback for failures.
* test\_add\_5 and test\_add\_6 will be executed and test\_add\_6 will report failure with traceback while the test\_add\_5 passes

Execute the test by py.test test\_addition.py -v and see the result

test\_addition.py::test\_add\_1 SKIPPED

test\_addition.py::test\_add\_2 SKIPPED

test\_addition.py::test\_add\_3 XPASS

test\_addition.py::test\_add\_4 xfail

test\_addition.py::test\_add\_5 PASSED

test\_addition.py::test\_add\_6 FAILED

============================================== FAILURES ==============================================

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ test\_add\_6 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

def test\_add\_6():

> assert 3+2 == 6,"failed"

E AssertionError: failed

E assert (3 + 2) == 6

test\_addition.py:24: AssertionError

================ 1 failed, 1 passed, 2 skipped, 1 xfailed, 1 xpassed in 0.07 seconds =================

## Results XML

We can create test results in XML format which we can feed to Continuous Integration servers for further processing and so. This can be done by

py.test test\_sample1.py -v --junitxml="result.xml"

The result.xml will record the test execution result. Find a sample result.xml below

<?xml version="1.0" encoding="UTF-8"?>

<testsuite errors="0" failures="1" name="pytest" skips="0" tests="2" time="0.046">

<testcase classname="test\_sample1" file="test\_sample1.py" line="3" name="test\_file1\_method1" time="0.001384973526">

<failure message="AssertionError:test failed because x=5 y=6 assert 5 ==6">

@pytest.mark.set1

def test\_file1\_method1():

x=5

y=6

assert x+1 == y,"test failed"

> assert x == y,"test failed because x=" + str(x) + " y=" + str(y)

E AssertionError: test failed because x=5 y=6

E assert 5 == 6

test\_sample1.py:9: AssertionError

</failure>

</testcase>

<testcase classname="test\_sample1" file="test\_sample1.py" line="10" name="test\_file1\_method2" time="0.000830173492432" />

</testsuite>

From <testsuite errors="0" failures="1" name="pytest" skips="0" tests="2" time="0.046"> we can see a total of two tests of which one is failed. Below that you can see the details regarding each executed test under <testcase> tag.

## A pytest framework testing an API

Now we will create a small pytest framework to test an API. The API here used is a free one from <https://reqres.in/>. This website is just to provide testable API. This website doesn't store our data.

Here we will write some tests for

* listing some users
* login with users

Create the below files with the code given

conftest.py - have a fixture which will supply base url for all the test methods

import pytest

@pytest.fixture

def supply\_url():

return "https://reqres.in/api"

test\_list\_user.py – contains the test methods for listing valid and invalid users

* test\_list\_valid\_user tests for valid user fetch and verifies the response
* test\_list\_invaliduser tests for invalid user fetch and verifies the response

import pytest

import requests

import json

@pytest.mark.parametrize("userid, firstname",[(1,"George"),(2,"Janet")])

def test\_list\_valid\_user(supply\_url,userid,firstname):

url = supply\_url + "/users/" + str(userid)

resp = requests.get(url)

j = json.loads(resp.text)

assert resp.status\_code == 200, resp.text

assert j['data']['id'] == userid, resp.text

assert j['data']['first\_name'] == firstname, resp.text

def test\_list\_invaliduser(supply\_url):

url = supply\_url + "/users/50"

resp = requests.get(url)

assert resp.status\_code == 404, resp.text

test\_login\_user.py – contains test methods for testing login functionality.

* test\_login\_valid tests the valid login attempt with email and password
* test\_login\_no\_password tests the invalid login attempt without passing password
* test\_login\_no\_email tests the invalid login attempt without passing email.

import pytest

import requests

import json

def test\_login\_valid(supply\_url):

url = supply\_url + "/login/"

data = {'email':['test@test.com](mailto:'test@test.com)

','password':'something'}

resp = requests.post(url, data=data)

j = json.loads(resp.text)

assert resp.status\_code == 200, resp.text

assert j['token'] == "QpwL5tke4Pnpja7X", resp.text

def test\_login\_no\_password(supply\_url):

url = supply\_url + "/login/"

data = {'email':['test@test.com](mailto:'test@test.com)'}

resp = requests.post(url, data=data)

j = json.loads(resp.text)

assert resp.status\_code == 400, resp.text

assert j['error'] == "Missing password", resp.text

def test\_login\_no\_email(supply\_url):

url = supply\_url + "/login/"

data = {}

resp = requests.post(url, data=data)

j = json.loads(resp.text)

assert resp.status\_code == 400, resp.text

assert j['error'] == "Missing email or username", resp.text

Run the test using py.test -v

See the result as

test\_list\_user.py::test\_list\_valid\_user[1-George] PASSED

test\_list\_user.py::test\_list\_valid\_user[2-Janet] PASSED

test\_list\_user.py::test\_list\_invaliduser PASSED

test\_login\_user.py::test\_login\_valid PASSED

test\_login\_user.py::test\_login\_no\_password PASSED

test\_login\_user.py::test\_login\_no\_email PASSED

Update the tests and try various outputs

## Summary

In this pytest tutorial, we covered

* Install pytest using pip install pytest=2.9.1
* Simple pytest program and run it with py.test command.
* Assertion statements, assert x==y, will return either True or False.
* How pytest identifies test files and methods.
  + Test files starting with **test\_** or ending with **\_test**
  + Test methods starting with **test**
* py.test command will run all the test files in that folder and subfolders. To run a specific file, we can use the command py.test <filename>
* Run a subset of test methods
  + Grouping of test names by substring matching.  
    py.test -k <name> -v will run all the tests having <name> in its name.
  + Run test by markers.Mark the tests using @pytest.mark.<name> and run the tests using pytest -m <name> to run tests marked as <name>.
* Run tests in parallel
  + Install pytest-xdist using pip install pytest-xdist
  + Run tests using py.test -n NUM where NUM is the number of workers
* Creating fixture methods to run code before every test by marking the method with @pytest.fixture
  + The scope of a fixture method is within the file it is defined.
  + A fixture method can be accessed across multiple test files by defining it in conftest.py file.
  + A test method can access a fixture by using it as an input argument.
* Parametrizing tests to run it against multiple set of inputs.  
  @pytest.mark.parametrize("input1, input2, output",[(5,5,10),(3,5,12)]) def test\_add(input1, input2, output):  
  assert input1+input2 == output,"failed"  
  will run the test with inputs (5,5,10) and (3,5,12)
* Skip/xfail tests using @pytets.mark.skip and @pytest.mark.xfail
* Create test results in XML format which covers executed test details using py.test test\_sample1.py -v --junitxml="result.xml"
* A sample pytest framework to test an API

# Django Tutorials for Beginners

Before we learn Django, let's understand:

## What is a Web Framework?

A web framework is a code library which helps you to build a flexible, scalable, and maintainable; dynamic website, web app, and web services. Different web frameworks are Zend for PHP, Ruby on Rails for Ruby, etc.

## What is Django?

Django is a web development framework for Python which offers a standard method for fast and effective website development. It helps you to assists in building and maintaining quality web applications. It enables you to make the development process smooth and time-saving.

It is a high-level web framework which allows performing rapid development. The primary goal of this web framework is to create complex database-driven websites.

In this tutorial, you will learn

* [What is a Web Framework?](https://www.guru99.com/django-tutorial.html#1)
* [What is Django?](https://www.guru99.com/django-tutorial.html#2)
* [Why Django? Key Advantages](https://www.guru99.com/django-tutorial.html#3)
* [History of Django](https://www.guru99.com/django-tutorial.html#4)
* [Features of Django](https://www.guru99.com/django-tutorial.html#5)
* [Characteristics of Django](https://www.guru99.com/django-tutorial.html#6)
* [Django Architecture](https://www.guru99.com/django-tutorial.html#7)
* [Components of Django](https://www.guru99.com/django-tutorial.html#8)
* [Famous Companies Using Django](https://www.guru99.com/django-tutorial.html#9)
* [Disadvantages of using Django](https://www.guru99.com/django-tutorial.html#10)

## Why Django? Key Advantages

Here, are prime reasons for using Django:

* Django is easy to set up and run. It offers a variety of options to get started
* It provides a ready-to-use user interface for administrative activities
* It enables multilingual websites by using its built-in internationalization system
* Helps you to meet the massive traffic demands quickly
* Django is used to build all types of content management systems, social networks as well as scientific computing platforms.
* Django helps you to provide end-to-end application testing
* Helps you to document your API with an HTML output
* REST Framework has rich support for several authentication protocols
* Permissions and throttling policies
* It is widely used for rate limiting API requests from a single user.

## History of Django

* The project was started by Adian Holovaty and Simon Willison as an internal project at Lawrence Journal-World newspaper in 2003
* It was released In July 2005 as "Django," named after the jazz guitarist Django Reinhard
* In June 2008, it was declared that a Django Software Foundation (DSF) would maintain Django.
* Django 1.1 version was released on July 29, 2009, with features like Aggregates, transaction-based tests.
* Django 1.2 version was released on May 17th, 2010. The update offered features like multiple databases, Implementation of email backend and improvements for sessions and cookie are made in this version.
* On 23 Mar 2011, 1.3 version with features like Class-based views, static files were introduced
* On 23 Mar 2012, 1.4 version launched with new features like timezones and in-browser testing
* On 26 Feb 2013,1.5 version comes with Python 3 Support, configurable user model features.
* On 6 Nov 2013, 1.6 version comes with DB transaction management, connection pooling features.
* On 2 Sep 2014. 1.7 version had new features like application load, migrations, and configuration
* On, 1 Apr 2015 native support provided for multiple template engines.
* On, 1 Dec 2015, in 1.9 version, automatic password validation feature was introduced.
* On, 1 Aug 2016, in 1.10 version full-text search for PostgreSQL launched.
* On, 4 Apr 2017, 1.11 version becomes the last version to support Python 2.7
* On, 2 Dec 2017, version 2.0 becomes first python 3- only release version.
* On, 1 Aug 2018, version 2.1 comes with Model "view" permission support feature.

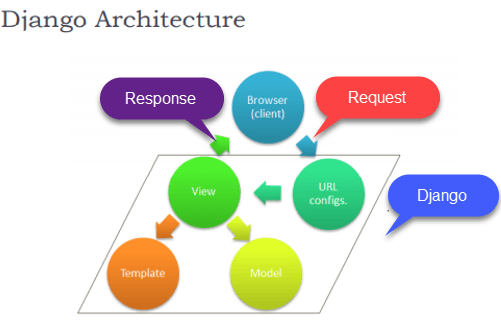
## Features of Django

* Helps you to define patterns for the URLs in your application
* Simple but powerful URL system
* Built-in authentication system
* Object-oriented programming language database which offers best in class data storage and retrieval
* Automatic admin interface feature allows the functionality of adding, editing and deleting items. You can customize the admin panel as per your need.
* Cache framework comes with multiple cache mechanisms.

## Characteristics of Django

* Loosely Coupled − Django helps you to make each element of its stack independent of the others.
* Less code - Ensures effective development
* Not repeated- Everything should be developed in precisely one place instead of repeating it again
* Fast development- Django's offers fast and reliable application development.
* Consistent design - Django maintains a clean design and makes it easy to follow the best web development practices.

## Django Architecture: MVC and MVT



### MVC Pattern:

When talking about applications which provide UI (web or desktop), we usually talk about MVC architecture. MVC pattern is based on Model, View, and Controller.

The **Model** defines the data structure and takes to care for querying the database.

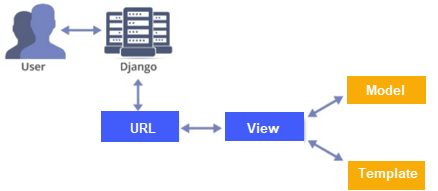
The **View** defines what data should be presented and returns an HTTP response.

The **Controller** is that part of the application that handles the user interaction.

### Django MVC-MVT pattern

The Model-View-Template (MVT) is a different concept compared to MVC. The main difference between these two architectural patterns is that Django itself manages the Controller part (software code that controls the interactions between the Model and View). The template is an HTML file which mixed with Django Template Language file which also called DTL.

The below-given diagram shows how all the components of the MVT pattern interact with each other to serve specific to a user request.

[](https://www.guru99.com/images/1/030819_1130_DjangoTutor2.png)MVT Architecture

As seen in above diagram, a user requests for a resource to Django. Django acts as a controller and checks to the available resource in URL.

If URL maps, a view is called which interacts with model and template. Django then responds to the user and sends a template as a response.

## Components of Django

### Form:

Django has a powerful form library which handles rendering forms as HTML. The library helps in validating submitted data and converting it to Python types.

### Authentication:

It handles user accounts, groups, cookie-based user sessions, etc.

### Admin:

It reads metadata in your models to provide a robust interface which can be used to manage content on your site.

### Internationalization:

Django provides support for translating text into various languages, locale-specific formatting of dates, times, numbers, and timezones.

### Security:

Django provides safeguard against the following attacks:

* Cross-Site Request Forgery (CSRF)
* Cross-site scripting
* SQL injection
* Clickjacking
* Remote code execution

## Famous Companies Using Django

Here, are companies which are using Django:

* BitBucket
* DISQUS
* NASA
* PBS (Public Broadcasting Service)
* Pinterest
* Instagram
* Mozilla
* The Washington Post
* NY Times
* LA Times
* The Guardian National Geographic
* Discovery Channel

## Disadvantages of using Django

* A process cannot handle multiple requests simultaneously.
* Django is a very much monolithic tool.
* Components get deployed together which can create confusion.
* Knowledge of full system needed to work on Django
* Makes web application components tightly-coupled
* You need to do lengthy programming for minor tasks.

### Summary

* Django is a web development framework for Python which offers a standard method for fast and effective website development.
* Django started was by Adian Holovaty and Simon Willison as an internal project at Lawrence Journal-World newspaper in 2003
* Django helps you to document your API with an HTML output
* Django allows you to customize the admin panel as per your need.
* Django helps you to make each element of its stack independent of the others.
* The main difference between MVC and MVT architectural patterns is that Django itself manages the Controller part.
* Essential components of Django's architecture are:

Form, Authentication, Admin, Internationalization, Security

* Bitbucket, DISQUS, Nasa, Pinterest, Instagram, Mozilla, etc. use Django
* The biggest drawback of Django is that it can't handle multiple requests simultaneously.

# 16 Best Web Scraping Tools for Data Extraction in 2020

Web scraping tools are specially developed software for extracting useful information from the websites. These tools are helpful for anyone who is looking to collect some form of data from the Internet.

Here, is a curated list of top 16 Web Scraping Tools. This list includes commercial as well as open-source tools with popular features and latest download link.

### 1) [Scraping-Bot](https://bit.ly/2OJ66Wl)

[](https://bit.ly/2OJ66Wl)

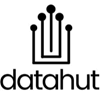
[Scraping-Bot.io](https://bit.ly/2OJ66Wl) is an efficient tool to scrape data from a URL. It works particularly well on product pages where it collects all you need to know: image, product title, product price, product description, stock, delivery costs, EAN, product category etc. You can also use it to check your ranking on google and improve your SEO. Use the Live test on the Dashboard to test without coding.

**Features:**

* JS rendering (Headless Chrome)
* High quality proxies
* Full Page HTML
* Up to 20 concurrent requests
* Geotargeting
* Allows for large bulk scraping needs
* Free basic usage monthly plan.

[](https://bit.ly/2OJ66Wl)

### 2) [Datahut](https://bit.ly/2svIprE)

[](https://bit.ly/2svIprE)

Get structured data from websites. It works pretty well on eCommerce websites where a large amount of data is scattered all over. No need for coding, servers or expensive software is the factors that make this web scraping platform unique.

**Features**

* Machine learning-based is used by the Q&A to verify the extracted data.
* Data will be shred through Amazon S3, Dropbox, Box, FTP upload or even via a custom API.
* You can build your app on top of the data extraction infrastructure, which is maintained by us.
* Best services are offered in a competitive price range.

[](https://bit.ly/2svIprE)

### 3) [Scraper API](https://bit.ly/2Yq33F8)

[](https://bit.ly/2Yq33F8)

[Scraper API](https://bit.ly/2Yq33F8) tool helps you to manage proxies, browsers, and CAPTCHAs. This allows you to get the HTML from any web page with a simple API call. It is easy to integrate as you just need to send a GET request to API endpoint with your API key and URL.

**Features:**

* Helps you to render JavaScript
* It allows you to customize the headers of each request as well as the request type
* The tool offers unparalleled speed and reliability which allows building scalable web scrapers
* Geolocated Rotating Proxies

[](https://bit.ly/2Yq33F8)

### 4) [Octoparse](https://bit.ly/2TDFpl4)

[](https://bit.ly/2TDFpl4)

[Octoparse](https://bit.ly/2TDFpl4) is another useful web scraping tool that is easy to configure. The point and click user interface allow you to teach the scraper how to navigate and extract fields from a website.

**Features:**

* Ad Blocking technique feature helps you to extract data from Ad-heavy pages
* The tool provides support to mimics a human user while visiting and scraping data from the specific websites
* Octoparse allows you to run your extraction on the cloud and your local machine
* Allows you to export all types of scraped data in TXT, HTML CSV, or Excel formats

[](https://bit.ly/2TDFpl4)

### 5) Import.io

[https://www.guru99.com/images/1/020819_1039_16BestWebSc3.png](https://www.guru99.com/images/1/020819_1039_16BestWebSc3.png)

This web scraping tool helps you to form your datasets by importing the data from a specific web page and exporting the data to CSV. It allows you to Integrate data into applications using APIs and webhooks.

**Features:**

* Easy interaction with web forms/logins
* Schedule data extraction
* You can store and access data by using Import.io cloud
* Gain insights with reports, charts, and visualizations
* Automate web interaction and workflows

URL: <http://www.import.io/>

### 6) Webhose.io

[https://www.guru99.com/images/1/020819_1039_16BestWebSc4.png](https://www.guru99.com/images/1/020819_1039_16BestWebSc4.png)

Webhose.io provides direct access to structured and real-time data to crawling thousands of websites. It allows you to access historical feeds covering over ten years' worth of data.

**Features:**

* Get structured, machine-readable datasets in JSON and XML formats
* Helps you to access a massive repository of data feeds without paying any extra fees
* An advanced filter allows you to conduct granular analyze and datasets you want to feed

Url: <https://webhose.io/products/archived-web-data/>

### 7) Dexi Intelligent

[](https://www.guru99.com/images/1/020819_1039_16BestWebSc5.png)

Dexi intelligent is a web scraping tool allows you to transform unlimited web data into immediate business value. This web scraping tool enables you to cut cost and saves precious time of your organization.

**Features:**

* Increased efficiency, accuracy and quality
* Ultimate scale and speed for data intelligence
* Fast, efficient data extraction
* High scale knowledge capture

Url: <http://dexi.io/>

### 8) Scrapinghub

[](https://www.guru99.com/images/1/020819_1039_16BestWebSc6.png)

Scrapinghub is a hassle-free cloud base data extraction tool which helps companies to fetch valuable data. The tool allows you to store data in the high-ability database.

**Features:**

* Allows you to converts the entire web page into organized content
* Helps you to deploy crawlers and scale them on demand without the need to care about servers, monitoring or backups
* Supports bypassing bot counter-measures to crawl large or bot-protected sites

Url: <http://scrapinghub.com/>

### 9) Outwit

[](https://www.guru99.com/images/1/020819_1039_16BestWebSc7.jpg)

It is a Firefox extension that can be easily downloaded from the Firefox add-ons store. You will get three distinct option according to your requirement to buy this product. 1.Pro edition, 2.Expert edition, and 3.Enterpsie edition.

**Features:**

* Allows you to grab contacts from the web and email source simply
* No programming skill is needed to exact data from sites using Outwit hub
* With just single click on the exploration button, you can launch the scraping on hundreds of web pages

Url: <http://www.outwit.com/>

### 10) PareseHub

[](https://www.guru99.com/images/1/020819_1039_16BestWebSc8.jpg)

ParseHub is a free web scraping tool. This advanced web scraper allows extracting data is as easy as clicking the data you need. It allows you to download your scraped data in any format for analysis.

**Features:**

* Clean text & HTML before downloading data
* The easy to use graphical interface
* Helps you to collect and store data on servers automatically

Url: <http://www.parsehub.com/>

### 11) Diffbot

[](https://www.guru99.com/images/1/020819_1039_16BestWebSc9.png)

Diffbot allows you to get various type of useful data from the web without the hassle. You don't need to pay the expense of costly web scraping or doing manual research. The tool will enable you to exact structured data from any URL with AI extractors.

**Features:**

* Offers multiple sources of data form a complete, accurate picture of every entity
* Provide support to extract structured data from any URL with AI Extractors
* Helps you to scale up your extraction to 10,000s of domains with Crawlbot
* Knowledge Graph feature offers accurate, complete and deep data from the web that BI needs to produce meaningful insights

Url: [http://www.diffbot.com](http://www.diffbot.com/)

### 12) Data streamer

[](https://www.guru99.com/images/1/020819_1039_16BestWebSc11.png)

Data Stermer tool helps you to fetch social media content from across the web. It allows you to extract critical metadata using Natural language processing.

**Features:**

* Integrated full-text search powered by Kibana and Elasticsearch
* Integrated boilerplate removal and content extraction based on information retrieval techniques
* Built on a fault-tolerant infrastructure and ensure high availability of information
* Easy to use and comprehensive admin console

Url: [http://www.datastreamer.io//](http://www.datastreamer.io/)

### 13) FMiner:

[](https://www.guru99.com/images/1/020819_1039_16BestWebSc12.jpg)

FMiner is another popular tool for web scraping, data extraction, crawling screen scraping, macro, and web support for Window and Mac OS.

**Features:**

* Allows you to design a data extraction project by using easy to use the visual editor
* Helps you to drill l through site pages using a combination of link structures, drop-down selections or url pattern matching
* You can extract data from hard to crawl Web 2.0 dynamic websites
* Allows you to target website CAPTCHA protection with the help of third-party automated decaptcha services or manual entry

Url: <http://www.fminer.com/>

### 14) Apify SDK:

[](https://www.guru99.com/images/1/020819_1039_16BestWebSc13.png)

Apify SDK is a scalable web crawling and scraping library for Javascript. It allows development and data exaction and web automation with headless crome and puppeteer.

**Features:**

* Automates any web workflow
* Allows easy and fast crawling across the web
* Works locally and in the cloud
* Runs on JavaScript

Url: <http://sdk.apify.com/>

### 15) Content Grabber:

[](https://www.guru99.com/images/1/020819_1039_16BestWebSc14.jpg)

The content grabber is a powerful big data solution for reliable web data extraction. It allows you to scale your organization. It offers easy to use features like visual point and clicks editor.

**Features:**

* Extract web data faster and faster way compares to other solution
* Help you to build web apps with the dedicated web API that allow you to execute web data directly from your website
* Helps you move between various platforms

Url: <http://www.contentgrabber.com/>

### 16) Mozenda:

Mozenda allows you to extract text, images and PDF content from web pages. It helps you to organize and prepare data files for publishing.

[](https://www.guru99.com/images/1/020819_1039_16BestWebSc15.jpg)

**Features:**

* You can collect and publish your web data to your preferred Bl tool or database
* Offers point-and-click interface to create web scraping agents in minutes
* Job Sequencer and Request Blocking features to harvest web data in a real time
* Best in class account management and customer support

Url: <http://www.mozenda.com/>

### 17) Web Scraper Chrome Extension

[](https://www.guru99.com/images/1/020819_1039_16BestWebSc16.jpg)

Web scraper is a chrome extension which helps you for the web scraping and data acquisition. It allows you to scape multiple pages and offers dynamic data extraction capabilities.

**Features:**

* Scraped data is stored in local storage
* Multiple data selection types
* Extract data from dynamic pages
* Browse scraped data
* Export scraped data as CSV
* Import, Export sitemaps

Url: <https://chrome.google.com/webstore/detail/data-scraper-easy-web-scr/nndknepjnldbdbepjfgmncbggmopgden?hl=en>

# 11 BEST Python IDEs in 2020

Python code editors are designed for the developers to code and debug program easily. Using these Python IDE(Integrated Development Environment), you can manage a large codebase and achieve quick deployment.

Developers can use these editors to create desktop or web application. The IDEs can also be used by DevOps engineers for continuous Integration.

Following is a handpicked list of Top Python Code

Editors, with popular features and latest download links. The list contains both open-source(free) and premium tools.

### 1) PyCharm

[](https://www.guru99.com/images/1/090719_0649_11BESTPytho1.png)

PayCharm is a cross-platform IDE used for Python programming. This editor can be used on Windows, macOS, and Linux. This software contains API that can be used by the developers to write their own Python plugins so that they can extend the basic functionalities.

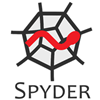
**Price:** Free

**Features:**

* It is an intelligent Python code editor supports for CoffeeScript, JavaScript, CSS, and TypeScript.
* Provides smart search to jump to any file, symbol, or class.
* Smart Code Navigation
* It offers quick and safe refactoring of code.
* It allows you to access PostgreSQL, Oracle, MySQL, SQL Server, and many other databases from the IDE.

**Download Link:**<https://www.jetbrains.com/pycharm/>

### 2) Spyder

[](https://www.guru99.com/images/1/090719_0649_11BESTPytho2.png)

Spyder is a scientific integrated development environment written in Python. This software is designed for and by scientists who can integrate with Matplotlib, SciPy, NumPy, Pandas, Cython, IPython, SymPy, and other open-source software. Spyder is available through Anaconda (open-source distribution system) distribution on Windows, macOS, and Linux.

**Price**: Free

**Features:**

* Allows you to run Python code by cell, line, or file.
* Plot a histogram or time-series, make changes in dateframe or numpy array.
* It offers automatic code completion and horizontal/vertical splitting.
* Find and eliminate bottlenecks
* An interactive way to trace each step of Python code execution.

**Download Link:** <https://www.spyder-ide.org/>

### 3) IDLE

[](https://www.guru99.com/images/1/090719_0649_11BESTPytho3.png)

IDLE (Integrated Development and Learning Environment) is a default editor that comes with Python. This software helps a beginner to learn Python easily. IDLE software package is optional for many Linux distributions. The tool can be used on Windows, macOS, and Unix.

**Price:**free

**Features:**

* Search multiple files
* It has an interactive interpreter with colorizing of input, output, and error messages.
* Supports smart indent, undo, call tips, and auto-completion.
* Enable you to search and replace within any window.

**Download Link:** <https://docs.python.org/3/library/idle.html>

### 4) Sublime Text 3

[](https://www.guru99.com/images/1/090719_0649_11BESTPytho4.jpg)

Sublime Text 3 is a code editor which supports many languages including Python. It has basic built-in support for Python. Customization of Sublime Text 3 is available for creating create a full-fledged Python programming environment. The editor supports OS X, Windows, and Linux operating systems.

**Price:** Free Trial

**Features:**

* Allows you to highlight syntax.
* It has command Palette implementation that accepts text input from users.
* Handle UTF8 BOMs in .gitignore files
* Display badges for folders and file to indicate Git status
* Changes to a file are represented by markers available in the gutter.

Download Link: <https://www.sublimetext.com/3>

### 5) Visual Studio Code

[](https://www.guru99.com/images/1/090719_0649_11BESTPytho5.png)

Visual Studio Code (VS Code) is an open-source environment developed by Microsoft. This IDE can be used for Python development. Visual Studio Code is based on Electron which is a framework to deploy Node JS applications for the computer running on the Blink browser engine.

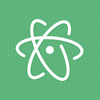
**Price:** Free

**Features:**

* The editor provides smart code completion based on function definition, imported modules, as well as variable types.
* You can work with Git as well as other SCM providers
* Enable you to debug code from the editor.
* Provides extensions to add new languages, debuggers, themes to gain the advantage of additional services.

**Download Link:**<https://code.visualstudio.com/>

### 6) Atom

[](https://www.guru99.com/images/1/090719_0649_11BESTPytho6.png)

Atom is a useful code editor tool preferred by programmers due to its simple interface compared to the other editors. Atom users can submit packages and them for the software.

**Price:** Free

**Features:**

* Package Manager Integrated for Plugins support
* The feature of smart autocompletion
* Supports Command Palette
* Multiple panes
* Allow cross-platform editing

**Download Link:** <https://atom.io/>

### 7) Jupyter

[](https://www.guru99.com/images/1/090719_0649_11BESTPytho7.png)

Jupyter is a tool for people who have just started with data science. It is easy to use, interactive data science IDE across many programming languages that just not work as an editor, but also as an educational tool or presentation.

**Price:** Free

**Features:**

* It supports for Numerical simulation, data cleaning machine learning data visualization, and statistical modeling.
* Combine code, text, and images.
* Support for many programming languages.
* Integrated data science libraries (matplotlib, NumPy, Pandas).

**Download Link:** <https://jupyter.org/install.html>

### 8) Pydev

[](https://www.guru99.com/images/1/090719_0649_11BESTPytho8.png)

PyDev is a third-party Python editor for Eclipse. This editor can be used in not only Python but IronPython and Jython development.

**Price:** Free

**Features:**

* It has interactive console shortcuts
* Allows you to create a Google App Engine (GAE) Python project
* Find and Go to definition
* Automatically import code to complete it.
* You can Configure Django integration.

**Download Link:** <https://www.pydev.org/>

### 9) Thonny

[](https://www.guru99.com/images/1/090719_0649_11BESTPytho9.png)

Thonny is an IDE for learning and teaching programming, specially designed with the beginner Pythonista scripting environment. It is developed at The University of Tartu, which you can download for free on the Bitbucket repository for Windows, Linux, and Mac.

**Price:**Free

**Features:**

* Allows developers to view how their code and shell commands affect Python variables.
* It has a simple debugger.
* Provides support for evaluating an expression.
* Python function call opens a new window with separate local variables table as well as code pointer.
* Automatically spot syntax error.

**Download Link:** <https://thonny.org/>

### 10) Wing

[](https://www.guru99.com/images/1/090719_0649_11BESTPytho10.png)

Wing is a lightweight Python environment which is designed to give you productive development experience.

**Price:** Wing Pro trial is free. Wind Personal and Wing 101 are paid versions.

**Features:**

* Immediate feedback by writing your Python code.
* Helps you to remove common errors and write better Python code.
* You can check for debug data and try out bug fixes interactively without restarting your app.
* Wing supports test-driven development with various frameworks like the unittest, pytest, nose, doctest, and Django testing.

**Download Link:** <https://wingware.com/>

### 11) ActivePython

[](https://www.guru99.com/images/1/090719_0649_11BESTPytho11.png)

Increase software development data science with a secure and supported Python distribution. ActivePython is software consisting of the Python implementation CPython and a set of various extensions to facilitate installation.

**Price:** Free for community, however, coder, team, business. Enterprise versions are paid.

**Features:**

* It allows you to connect to your big data and databases, including Redis, MySQL, Hadoop, and MongoDB.
* Helps you to manage your data using, SciPy, Pandas, NumPy, and MatPlotLib.
* Supports machine learning models like TensorFlow, Keras, and Theano.
* Compatible with open-source Python so that you can avoid vendor lock-in.
* Uses OpenSSL patch for security.

**Download Link:** <https://www.activestate.com/products/activepython/>

# Python Tutorial for Beginners PDF

Python is an object-oriented programming language created by Guido Rossum in 1989. It is ideally designed for rapid prototyping of complex applications.

## Key Highlights of Python Tutorial PDF are

* 211+ pages
* eBook Designed for beginners
* Beautifully annotated screenshots
* You will get lifetime access



### Inside this PDF

1. How to Install Python
2. Creating Your First Python Program
3. Python Main Function
4. Variables in Python
5. Learning Python Strings: Replace, Join, Split, Reverse
6. Python Tuples Tutorial: Comparing, Deleting, Slicing, Keys, Unpacking
7. Python Dictionary: Beginners Tutorial
8. Python Operators: Complete Tutorial
9. Functions in Python
10. If Loop: Python Conditional Structures
11. Python Loops: While, For, Break, Continue, Enumerate
12. Python Class & Objects: Object Oriented Programming (OOP’s)
13. Python Regular Expressions: Complete Tutorial
14. Date, time and datetime classes in Python
15. Calendar in Python
16. Reading and Writing Files in Python
17. Python Check If File or Directory Exists
18. Python COPY File using shutil.copy(), shutil.copystat()
19. Python Rename File and Directory using os.rename()
20. Python ZIP file with Example
21. Accessing Internet Data with Python
22. Manipulating XML with Python