What is Python?

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

It is used for:

* web development (server-side),
* software development,
* mathematics,
* system scripting.

What can Python do?

* Python can be used on a server to create web applications.
* Python can be used alongside software to create workflows.
* Python can connect to database systems. It can also read and modify files.
* Python can be used to handle big data and perform complex mathematics.
* Python can be used for rapid prototyping, or for production-ready software development.

Why Python?

* Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
* Python has a simple syntax similar to the English language.
* Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
* Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
* Python can be treated in a procedural way, an object-orientated way or a functional way.

# **Python Features**

Python provides lots of features that are listed below.

#### **1) Easy to Learn and Use**

Python is easy to learn and use. It is developer-friendly and high level programming language.

#### **2) Expressive Language**

Python language is more expressive means that it is more understandable and readable.

#### **3) Interpreted Language**

Python is an interpreted language i.e. interpreter executes the code line by line at a time. This makes debugging easy and thus suitable for beginners.

#### **4) Cross-platform Language**

Python can run equally on different platforms such as Windows, Linux, Unix and Macintosh etc. So, we can say that Python is a portable language.

#### **5) Free and Open Source**

Python language is freely available at [offical web address](https://www.python.org/).The source-code is also available. Therefore it is open source.

#### **6) Object-Oriented Language**

Python supports object oriented language and concepts of classes and objects come into existence.

#### **7) Extensible**

It implies that other languages such as C/C++ can be used to compile the code and thus it can be used further in our python code.

#### **8) Large Standard Library**

Python has a large and broad library and prvides rich set of module and functions for rapid application development.

#### **9) GUI Programming Support**

Graphical user interfaces can be developed using Python.

#### **10) Integrated**

It can be easily integrated with languages like C, C++, JAVA etc.

# **Python Applications**

Python is known for its general purpose nature that makes it applicable in almost each domain of software development. Python as a whole can be used in any sphere of development.

Here, we are specifing applications areas where python can be applied.

#### **1) Web Applications**

We can use Python to develop web applications. It provides libraries to handle internet protocols such as HTML and XML, JSON, Email processing, request, beautifulSoup, Feedparser etc. It also provides Frameworks such as Django, Pyramid, Flask etc to design and delelop web based applications. Some important developments are: PythonWikiEngines, Pocoo, PythonBlogSoftware etc.

#### **2) Desktop GUI Applications**

Python provides Tk GUI library to develop user interface in python based application. Some other useful toolkits wxWidgets, Kivy, pyqt that are useable on several platforms. The Kivy is popular for writing multitouch applications.

#### **3) Software Development**

Python is helpful for software development process. It works as a support language and can be used for build control and management, testing etc.

#### **4) Scientific and Numeric**

Python is popular and widely used in scientific and numeric computing. Some useful library and package are SciPy, Pandas, IPython etc. SciPy is group of packages of engineering, science and mathematics.

#### **5) Business Applications**

Python is used to build Bussiness applications like ERP and e-commerce systems. Tryton is a high level application platform.

#### **6) Console Based Application**

We can use Python to develop console based applications. For example: **IPython**.

#### **7) Audio or Video based Applications**

Python is awesome to perform multiple tasks and can be used to develop multimedia applications. Some of real applications are: TimPlayer, cplay etc.

#### **8) 3D CAD Applications**

To create CAD application Fandango is a real application which provides full features of CAD.

#### **9) Enterprise Applications**

Python can be used to create applications which can be used within an Enterprise or an Organization. Some real time applications are: OpenErp, Tryton, Picalo etc.

#### **10) Applications for Images**

Using Python several application can be developed for image. Applications developed are: VPython, Gogh, imgSeek etc.

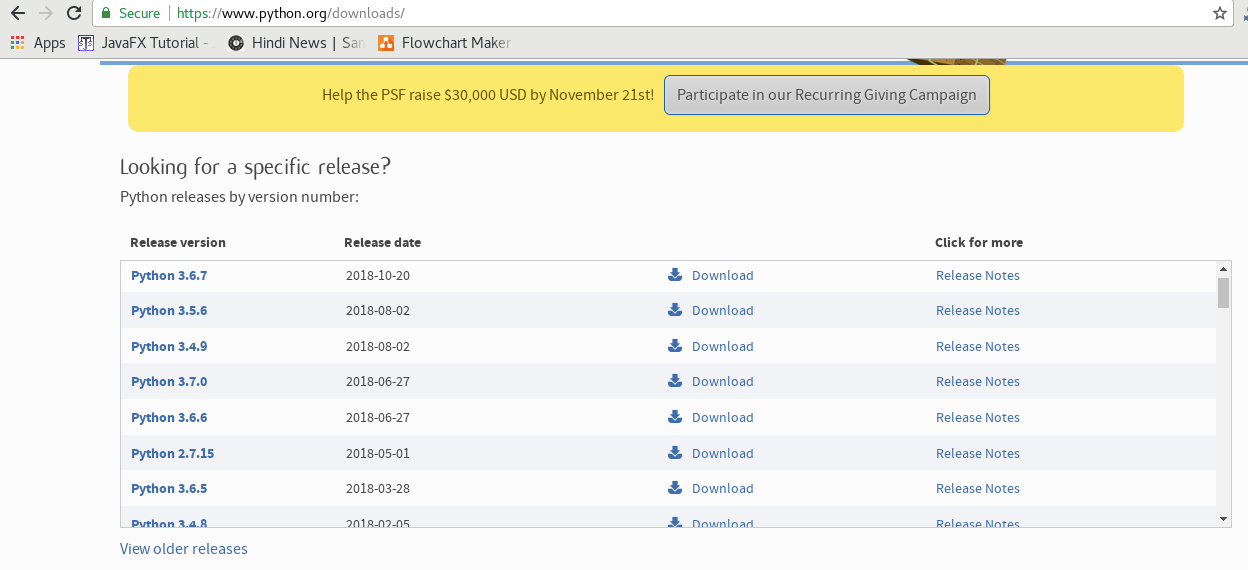
There are several such applications which can be developed using Python

# **How to Install Python (Environment Set-up)**

In this section of the tutorial, we will discuss the installation of python on various operating systems.

## Installation on Windows

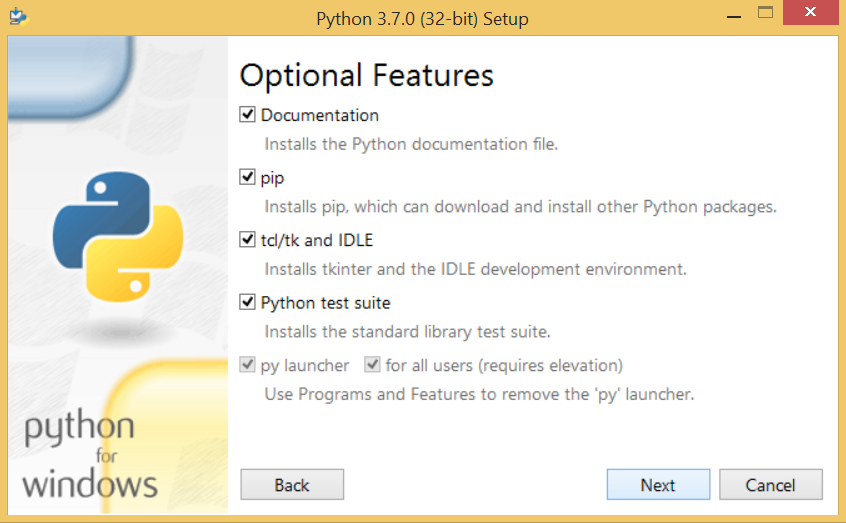
Visit the link <https://www.python.org/downloads/> to download the latest release of Python. In this process, we will install Python 3.6.7 on our Windows operating system.



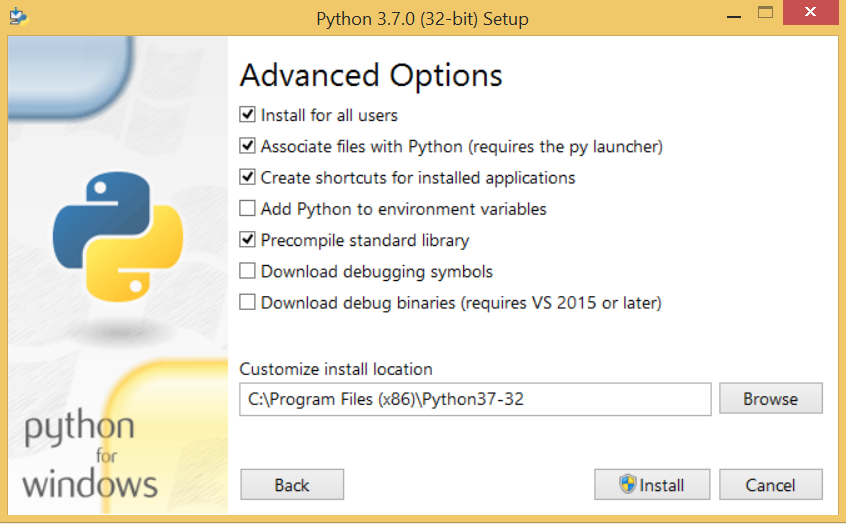
Double-click the executable file which is downloaded; the following window will open. Select Customize installation and proceed.

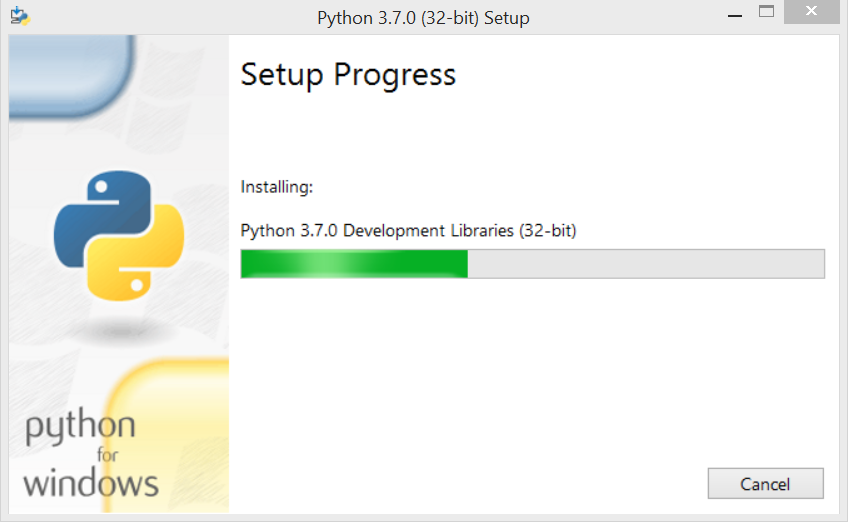
The following window shows all the optional features. All the features need to be installed and are checked by default; we need to click next to continue.

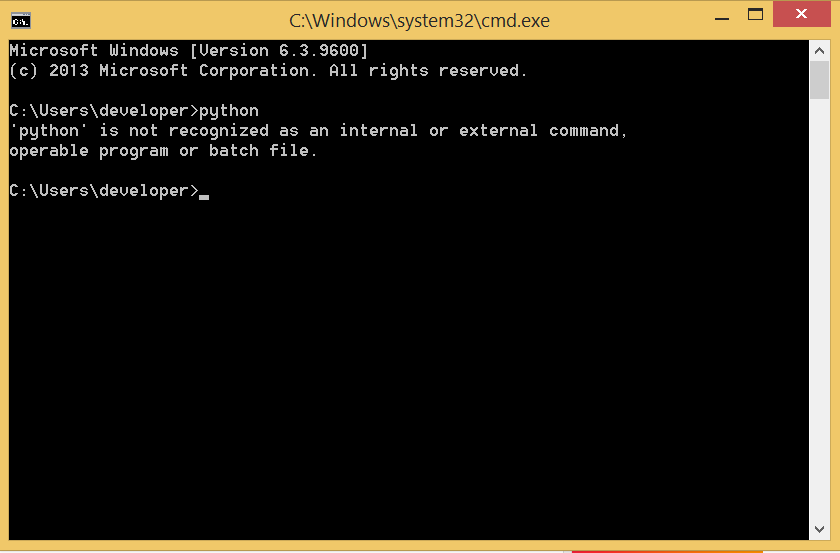
The following window shows a list of advanced options. Check all the options which you want to install and click next. Here, we must notice that the first check-box (install for all users) must be checked.

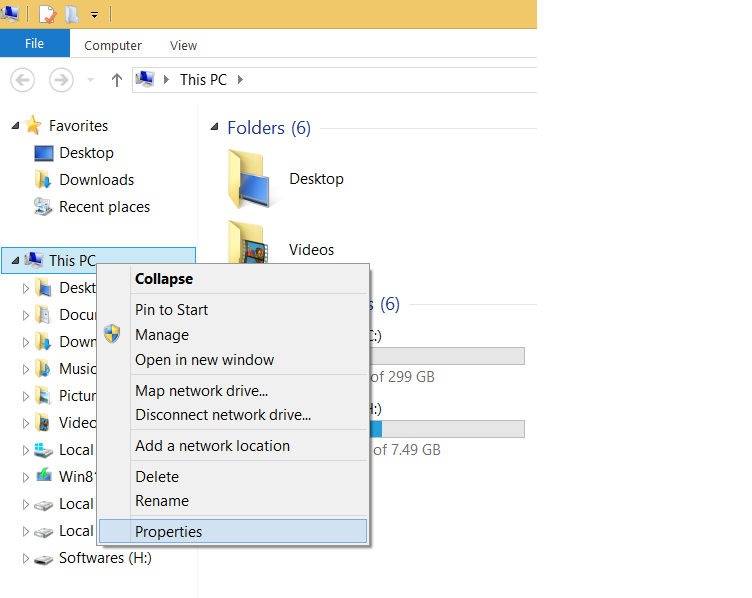
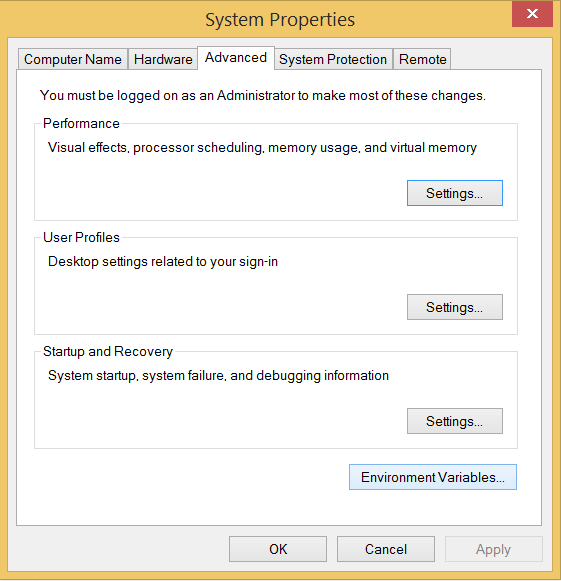
Now, we are ready to install python-3.6.7. Let's install it.

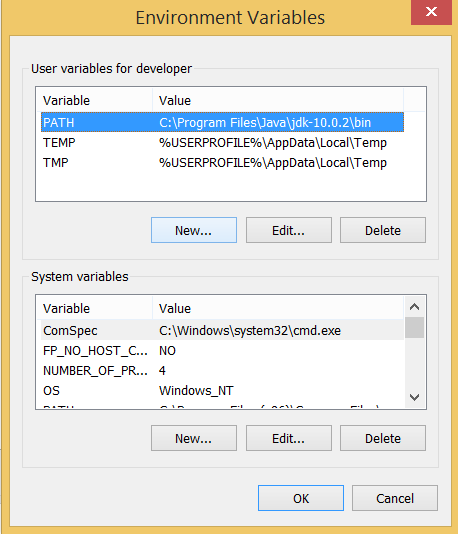
Now, try to run python on the command prompt. Type the command **python** in case of python2 or python3 in case of **python3**. It will show an error as given in the below image. It is because we haven't set the path.

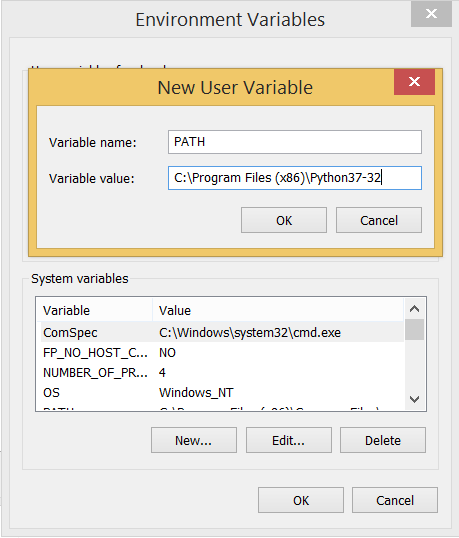
To set the path of python, we need to the right click on "my computer" and go to Properties → Advanced → Environment Variables.

Add the new path variable in the user variable section.

Type **PATH** as the variable name and set the path to the installation directory of the python shown in the below image.

Now, the path is set, we are ready to run python on our local system. Restart CMD, and type **python** again. It will open the python interpreter shell where we can execute the python statements.

# 

# **First Python Program**

In this Section, we will discuss the basic syntax of python by using which, we will run a simple program to print hello world on the console.

Python provides us the two ways to run a program:

* Using Interactive interpreter prompt
* Using a script file

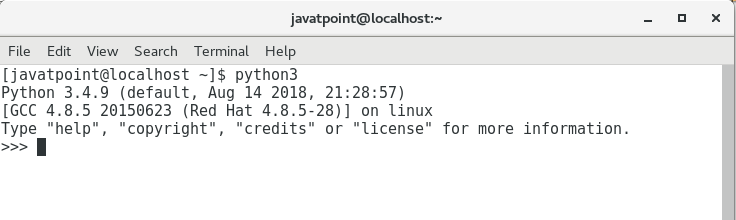
Let's discuss each one of them in detail.

## Interactive interpreter prompt

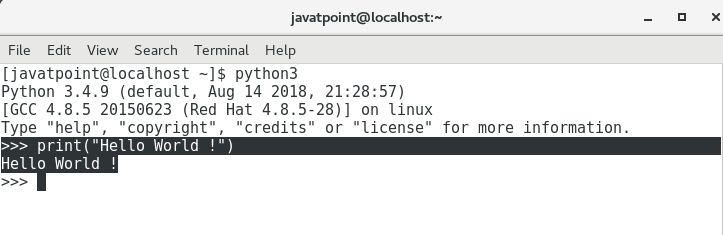
Python provides us the feature to execute the python statement one by one at the interactive prompt. It is preferable in the case where we are concerned about the output of each line of our python program.

To open the interactive mode, open the terminal (or command prompt) and type python (python3 in case if you have python2 and python3 both installed on your system).

It will open the following prompt where we can execute the python statement and check their impact on the console.

Let's run a python statement to print the traditional hello world on the console. Python3 provides print() function to print some message on the console. We can pass the message as a string into this function. Consider the following image.

Here, we get the message **"Hello World !"** printed on the console.

## Using a script file

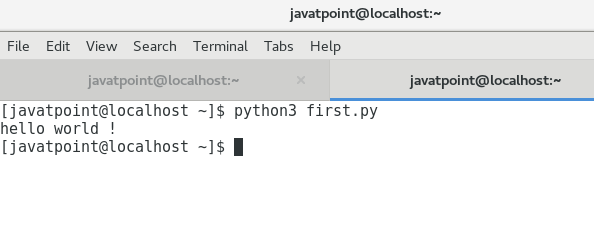
Interpreter prompt is good to run the individual statements of the code. However, we can not write the code every-time on the terminal.

We need to write our code into a file which can be executed later. For this purpose, open an editor like notepad, create a file named first.py (python used .py extension) and write the following code in it.

1. Print ("hello world"); #here, we have used print() function to print the message on the console.

To run this file named as first.py, we need to run the following command on the terminal.

**$ python3 first.py**

Hence, we get our output as the message **Hello World !** is printed on the console.

## Get Started with PyCharm

In our first program, we have used gedit on our CentOS as an editor. On Windows, we have an alternative like notepad or notepad++ to edit the code. However, these editors are not used as IDE for python since they are unable to show the syntax related suggestions.

JetBrains provides the most popular and a widely used cross-platform IDE **PyCharm** to run the python programs.

## PyCharm installation

As we have already stated, PyCharm is a cross-platform IDE, and hence it can be installed on a variety of the operating systems. In this section of the tutorial, we will cover the installation process of PyCharm on Windows, MacOS, CentOS, and Ubuntu.

### Windows

Installing PyCharm on Windows is very simple. To install PyCharm on Windows operating system, visit the link <https://www.jetbrains.com/pycharm/download/download-thanks.html?platform=windows> to download the executable installer. **Double click** the installer (.exe) file and install PyCharm by clicking next at each step.

# **Python Keywords**

Python Keywords are special reserved words which convey a special meaning to the compiler/interpreter. Each keyword have a special meaning and a specific operation. These keywords can't be used as variable. Following is the List of Python Keywords.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| True | False | None | and | as |
| asset | def | class | continue | break |
| else | finally | elif | del | except |
| global | for | if | from | import |
| raise | try | or | return | pass |
| nonlocal | in | not | is | lambda |

# **Python Variables**

Variable is a name which is used to refer memory location. Variable also known as identifier and used to hold value.

In Python, we don't need to specify the type of variable because Python is a type infer language and smart enough to get variable type.

Variable names can be a group of both letters and digits, but they have to begin with a letter or an underscore.

It is recomended to use lowercase letters for variable name. Rahul and rahul both are two different variables.

## Identifier Naming

Variables are the example of identifiers. An Identifier is used to identify the literals used in the program. The rules to name an identifier are given below.

* The first character of the variable must be an alphabet or underscore ( \_ ).
* All the characters except the first character may be an alphabet of lower-case(a-z), upper-case (A-Z), underscore or digit (0-9).
* Identifier name must not contain any white-space, or special character (!, @, #, %, ^, &, \*).
* Identifier name must not be similar to any keyword defined in the language.
* Identifier names are case sensitive for example my name, and MyName is not the same.
* Examples of valid identifiers : a123, \_n, n\_9, etc.
* Examples of invalid identifiers: 1a, n%4, n 9, etc.

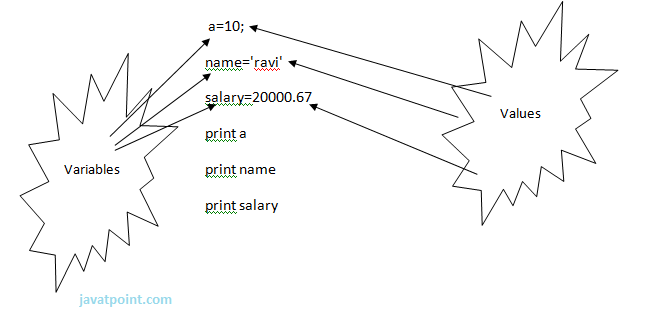
## Declaring Variable and Assigning Values

Python does not bound us to declare variable before using in the application. It allows us to create variable at required time.

We don't need to declare explicitly variable in Python. When we assign any value to the variable that variable is declared automatically.

The equal (=) operator is used to assign value to a variable.

**Eg:**



**Output:**

>>>

10

ravi

20000.67

>>>

## Multiple Assignment

Python allows us to assign a value to multiple variables in a single statement which is also known as multiple assignment.

We can apply multiple assignments in two ways either by assigning a single value to multiple variables or assigning multiple values to multiple variables. Lets see given examples.

**1. Assigning single value to multiple variables**

**Eg:**

x=y=z=50

**print** iple

**print** y

**print** z

**Output:**

>>>

50

50

50

>>>

**2.Assigning multiple values to multiple variables:**

**Eg:**

a,b,c=5,10,15

**print** a

**print** b

**print** c

**Output:**

>>>

5

10

15

>>>

The values will be assigned in the order in which variables appears.

### Basic Fundamentals:

This section contains the basic fundamentals of Python like :

**i)Tokens and their types.**

**ii) Comments**

**a)Tokens:**

* Tokens can be defined as a punctuator mark, reserved words and each individual word in a statement.
* Token is the smallest unit inside the given program.

There are following tokens in Python:

* Keywords.
* Identifiers.
* Literals.
* Operators.

### Tuples:

* Tuple is another form of collection where different type of data can be stored.
* It is similar to list where data is separated by commas. Only the difference is that list uses square bracket and tuple uses parenthesis.
* Tuples are enclosed in parenthesis and cannot be changed.

**Eg:**

>>> tuple=('rahul',100,60.4,'deepak')

>>> tuple1=('sanjay',10)

>>> tuple

('rahul', 100, 60.4, 'deepak')

>>> tuple[2:]

(60.4, 'deepak')

>>> tuple1[0]

'sanjay'

>>> tuple+tuple1

('rahul', 100, 60.4, 'deepak', 'sanjay', 10)

>>>

### Dictionary:

* Dictionary is a collection which works on a key-value pair.
* It works like an associated array where no two keys can be same.
* Dictionaries are enclosed by curly braces ({}) and values can be retrieved by square bracket([]).

**Eg:**

>>> dictionary={'name':'charlie','id':100,'dept':'it'}

>>> dictionary

{'dept': 'it', 'name': 'charlie', 'id': 100}

>>> dictionary.keys()

['dept', 'name', 'id']

>>> dictionary.values()

['it', 'charlie', 100]

>>>

# **Python Comments**

[❮ Previous](https://www.w3schools.com/python/python_syntax.asp)[Next ❯](https://www.w3schools.com/python/python_variables.asp)

Comments can be used to explain Python code.

Comments can be used to make the code more readable.

Comments can be used to prevent execution when testing code.

## Creating a Comment

Comments starts with a #, and Python will ignore them:

### Example

#This is a comment  
print("Hello, World!")

Comments can be placed at the end of a line, and Python will ignore the rest of the line:

### Example

print("Hello, World!") #This is a comment

Comments does not have to be text to explain the code, it can also be used to prevent Python from executing code:

### Example

#print("Hello, World!")  
print("Cheers, Mate!")

## Multi Line Comments

Python does not really have a syntax for multi line comments.

To add a multiline comment you could insert a # for each line:

### Example

#This is a comment  
#written in  
#more than just one line  
print("Hello, World!")

Or, not quite as intended, you can use a multiline string.

Since Python will ignore string literals that are not assigned to a variable, you can add a multiline string (triple quotes) in your code, and place your comment inside it:

### Example

"""  
This is a comment  
written in   
more than just one line  
"""  
print("Hello, World!")

As long as the string is not assigned to a variable, Python will read the code, but then ignore it, and you have made a multiline comment.

# **Python Variables**

## Creating Variables

Variables are containers for storing data values.

Unlike other programming languages, Python has no command for declaring a variable.

A variable is created the moment you first assign a value to it.

### Example

x = 5  
y = "John"  
print(x)  
print(y)

Variables do not need to be declared with any particular type and can even change type after they have been set.

### Example

x = 4 # x is of type int  
x = "Sally" # x is now of type str  
print(x)

String variables can be declared either by using single or double quotes:

### Example

x = "John"  
# is the same as  
x = 'John'

## Variable Names

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total\_volume). Rules for Python variables:

* A variable name must start with a letter or the underscore character
* A variable name cannot start with a number
* A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and \_ )
* Variable names are case-sensitive (age, Age and AGE are three different variables)

Remember that variable names are case-sensitive

## Assign Value to Multiple Variables

Python allows you to assign values to multiple variables in one line:

### Example

x, y, z = "Orange", "Banana", "Cherry"  
print(x)  
print(y)  
print(z)

And you can assign the same value to multiple variables in one line:

### Example

x = y = z = "Orange"  
print(x)  
print(y)  
print(z)

## Output Variables

The Python print statement is often used to output variables.

To combine both text and a variable, Python uses the + character:

### Example

x = "awesome"  
print("Python is " + x)

You can also use the + character to add a variable to another variable:

### Example

x = "Python is "  
y = "awesome"  
z =  x + y  
print(z)

For numbers, the + character works as a mathematical operator:

### Example

x = 5  
y = 10  
print(x + y)

If you try to combine a string and a number, Python will give you an error:

### Example

x = 5  
y = "John"  
print(x + y)

## Global Variables

Variables that are created outside of a function (as in all of the examples above) are known as global variables.

Global variables can be used by everyone, both inside of functions and outside.

### Example

Create a variable outside of a function, and use it inside the function

x = "awesome"  
  
def myfunc():  
  print("Python is " + x)  
  
myfunc()

If you create a variable with the same name inside a function, this variable will be local, and can only be used inside the function. The global variable with the same name will remain as it was, global and with the original value.

### Example

Create a variable inside a function, with the same name as the global variable

x = "awesome"  
  
def myfunc():  
  x = "fantastic"  
  print("Python is " + x)  
  
myfunc()  
  
print("Python is " + x)

## The global Keyword

Normally, when you create a variable inside a function, that variable is local, and can only be used inside that function.

To create a global variable inside a function, you can use the global keyword.

### Example

If you use the global keyword, the variable belongs to the global scope:

def myfunc():  
  global x  
  x = "fantastic"  
  
myfunc()  
  
print("Python is " + x)

Also, use the global keyword if you want to change a global variable inside a function.

### Example

To change the value of a global variable inside a function, refer to the variable by using the global keyword:

x = "awesome"  
  
def myfunc():  
  global x  
  x = "fantastic"  
  
myfunc()  
  
print("Python is " + x)

# **Python Numbers**

## Python Numbers

There are three numeric types in Python:

* int
* float
* complex

Variables of numeric types are created when you assign a value to them:

### Example

x = 1    # int  
y = 2.8  # float  
z = 1j   # complex

To verify the type of any object in Python, use the type() function:

### Example

print(type(x))  
print(type(y))  
print(type(z))

## Int

Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.

### Example

Integers:

x = 1  
y = 35656222554887711  
z = -3255522  
  
print(type(x))  
print(type(y))  
print(type(z))

## Float

Float, or "floating point number" is a number, positive or negative, containing one or more decimals.

### Example

Floats:

x = 1.10  
y = 1.0  
z = -35.59  
  
print(type(x))  
print(type(y))  
print(type(z))

Float can also be scientific numbers with an "e" to indicate the power of 10.

### Example

Floats:

x = 35e3  
y = 12E4  
z = -87.7e100  
  
print(type(x))  
print(type(y))  
print(type(z))

## Complex

Complex numbers are written with a "j" as the imaginary part:

### Example

Complex:

x = 3+5j  
y = 5j  
z = -5j  
  
print(type(x))  
print(type(y))  
print(type(z))

## Type Conversion (Type Casting)

You can convert from one type to another with the int(), float(), and complex() methods:

### Example

Convert from one type to another:

x = 1 # int  
y = 2.8 # float  
z = 1j # complex  
  
#convert from int to float:  
a = float(x)  
  
#convert from float to int:  
b = int(y)  
  
#convert from int to complex:  
c = complex(x)  
  
print(a)  
print(b)  
print(c)  
  
print(type(a))  
print(type(b))  
print(type(c))

**Note:** You cannot convert complex numbers into another number type.

# input a number

num = float(input("Input a value: "))

# printing input value

print "num = ", num

**Random Number**

Python does not have a random() function to make a random number, but Python has a built-in module calledrandom that can be used to make random numbers:

### Example

Import the random module, and display a random number between 1 and 9:

import random  
  
print(random.randrange(1,10))

# **Python Casting**

## Specify a Variable Type

There may be times when you want to specify a type on to a variable. This can be done with casting. Python is an object-orientated language, and as such it uses classes to define data types, including its primitive types.

Casting in python is therefore done using constructor functions:

* int() - constructs an integer number from an integer literal, a float literal (by rounding down to the previous whole number), or a string literal (providing the string represents a whole number)
* float() - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)
* str() - constructs a string from a wide variety of data types, including strings, integer literals and float literals

### Example

Integers:

x = int(1)   # x will be 1  
y = int(2.8) # y will be 2  
z = int("3") # z will be 3

### Example

Floats:

x = float(1)     # x will be 1.0  
y = float(2.8)   # y will be 2.8  
z = float("3")   # z will be 3.0  
w = float("4.2") # w will be 4.2

### Example

Strings:

x = str("s1") # x will be 's1'  
y = str(2)    # y will be '2'  
z = str(3.0)  # z will be '3.0'

# **Python Strings**

## String Literals

String literals in python are surrounded by either single quotation marks, or double quotation marks.

'hello' is the same as "hello".

You can display a string literal with the print() function:

### Example

print("Hello")  
print('Hello')

## Assign String to a Variable

Assigning a string to a variable is done with the variable name followed by an equal sign and the string:

### Example

a = "Hello"  
print(a)

## Multiline Strings

You can assign a multiline string to a variable by using three quotes:

### Example

You can use three double quotes:

a = """Lorem ipsum dolor sit amet,  
consectetur adipiscing elit,  
sed do eiusmod tempor incididunt  
ut labore et dolore magna aliqua."""  
print(a)

Or three single quotes:

### Example

a = '''Lorem ipsum dolor sit amet,  
consectetur adipiscing elit,  
sed do eiusmod tempor incididunt  
ut labore et dolore magna aliqua.'''  
print(a)

**Note:** in the result, the line breaks are inserted at the same position as in the code.

## Strings are Arrays

Like many other popular programming languages, strings in Python are arrays of bytes representing unicode characters.

However, Python does not have a character data type, a single character is simply a string with a length of 1.

Square brackets can be used to access elements of the string.

### Example

Get the character at position 1 (remember that the first character has the position 0):

a = "Hello, World!"  
print(a[1])

## Slicing

You can return a range of characters by using the slice syntax.

Specify the start index and the end index, separated by a colon, to return a part of the string.

### Example

Get the characters from position 2 to position 5 (not included):

b = "Hello, World!"  
print(b[2:5])

## Negative Indexing

Use negative indexes to start the slice from the end of the string:

### Example

Get the characters from position 5 to position 1, starting the count from the end of the string:

b = "Hello, World!"  
print(b[-5:-2])

## String Length

To get the length of a string, use the len() function.

### Example

The len() function returns the length of a string:

a = "Hello, World!"  
print(len(a))

## String Methods

Python has a set of built-in methods that you can use on strings.

### Example

The strip() method removes any whitespace from the beginning or the end:

a = " Hello, World! "  
print(a.strip()) # returns "Hello, World!"

### Example

The lower() method returns the string in lower case:

a = "Hello, World!"  
print(a.lower())

### Example

The upper() method returns the string in upper case:

a = "Hello, World!"  
print(a.upper())

### Example

The replace() method replaces a string with another string:

a = "Hello, World!"  
print(a.replace("H", "J"))

### Example

The split() method splits the string into substrings if it finds instances of the separator:

a = "Hello, World!"  
print(a.split(",")) # returns ['Hello', ' World!']

## Check String

To check if a certain phrase or character is present in a string, we can use the keywords in or not in.

### Example

Check if the phrase "ain" is present in the following text:

txt = "The rain in Spain stays mainly in the plain"  
x = "ain" in txt  
print(x)

### Example

Check if the phrase "ain" is NOT present in the following text:

txt = "The rain in Spain stays mainly in the plain"  
x = "ain" not in txt  
print(x)

## String Concatenation

To concatenate, or combine, two strings you can use the + operator.

### Example

Merge variable a with variable b into variable c:

a = "Hello"  
b = "World"  
c = a + b  
print(c)

### Example

To add a space between them, add a " ":

a = "Hello"  
b = "World"  
c = a + " " + b  
print(c)

## String Format

As we learned in the Python Variables chapter, we cannot combine strings and numbers like this:

### Example

age = 36  
txt = "My name is John, I am " + age  
print(txt)

But we can combine strings and numbers by using the format() method!

The format() method takes the passed arguments, formats them, and places them in the string where the placeholders {} are:

### Example

Use the format() method to insert numbers into strings:

age = 36  
txt = "My name is John, and I am {}"  
print(txt.format(age))

The format() method takes unlimited number of arguments, and are placed into the respective placeholders:

### Example

quantity = 3  
itemno = 567  
price = 49.95  
myorder = "I want {} pieces of item {} for {} dollars."  
print(myorder.format(quantity, itemno, price))

You can use index numbers {0} to be sure the arguments are placed in the correct placeholders:

### Example

quantity = 3  
itemno = 567  
price = 49.95  
myorder = "I want to pay {2} dollars for {0} pieces of item {1}."  
print(myorder.format(quantity, itemno, price))

## Escape Character

To insert characters that are illegal in a string, use an escape character.

An escape character is a backslash \ followed by the character you want to insert.

An example of an illegal character is a double quote inside a string that is surrounded by double quotes:

### Example

You will get an error if you use double quotes inside a string that is surrounded by double quotes:

txt = "We are the so-called "Vikings" from the north."

To fix this problem, use the escape character \":

### Example

The escape character allows you to use double quotes when you normally would not be allowed:

txt = "We are the so-called \"Vikings\" from the north."

Other escape characters used in Python:

|  |  |  |
| --- | --- | --- |
| **Code** | **Result** | **Try it** |
| \' | Single Quote | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_string_escape2) |
| \\ | Backslash | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_string_backslash) |
| \n | New Line | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_string_newline) |
| \r | Carriage Return | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_string_r) |
| \t | Tab | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_string_t) |
| \b | Backspace | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_string_b) |
| \f | Form Feed |  |
| \ooo | Octal value | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_string_octal) |
| \xhh | Hex value | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_string_hex) |

## String Methods

Python has a set of built-in methods that you can use on strings.

**Note:** All string methods returns new values. They do not change the original string.

|  |  |
| --- | --- |
| **Method** | **Description** |
| [capitalize()](https://www.w3schools.com/python/ref_string_capitalize.asp) | Converts the first character to upper case |
| [casefold()](https://www.w3schools.com/python/ref_string_casefold.asp) | Converts string into lower case |
| [center()](https://www.w3schools.com/python/ref_string_center.asp) | Returns a centered string |
| [count()](https://www.w3schools.com/python/ref_string_count.asp) | Returns the number of times a specified value occurs in a string |
| [encode()](https://www.w3schools.com/python/ref_string_encode.asp) | Returns an encoded version of the string |
| [endswith()](https://www.w3schools.com/python/ref_string_endswith.asp) | Returns true if the string ends with the specified value |
| [expandtabs()](https://www.w3schools.com/python/ref_string_expandtabs.asp) | Sets the tab size of the string |
| [find()](https://www.w3schools.com/python/ref_string_find.asp) | Searches the string for a specified value and returns the position of where it was found |
| [format()](https://www.w3schools.com/python/ref_string_format.asp) | Formats specified values in a string |
| format\_map() | Formats specified values in a string |
| [index()](https://www.w3schools.com/python/ref_string_index.asp) | Searches the string for a specified value and returns the position of where it was found |
| [isalnum()](https://www.w3schools.com/python/ref_string_isalnum.asp) | Returns True if all characters in the string are alphanumeric |
| [isalpha()](https://www.w3schools.com/python/ref_string_isalpha.asp) | Returns True if all characters in the string are in the alphabet |
| [isdecimal()](https://www.w3schools.com/python/ref_string_isdecimal.asp) | Returns True if all characters in the string are decimals |
| [isdigit()](https://www.w3schools.com/python/ref_string_isdigit.asp) | Returns True if all characters in the string are digits |
| [isidentifier()](https://www.w3schools.com/python/ref_string_isidentifier.asp) | Returns True if the string is an identifier |
| [islower()](https://www.w3schools.com/python/ref_string_islower.asp) | Returns True if all characters in the string are lower case |
| [isnumeric()](https://www.w3schools.com/python/ref_string_isnumeric.asp) | Returns True if all characters in the string are numeric |
| [isprintable()](https://www.w3schools.com/python/ref_string_isprintable.asp) | Returns True if all characters in the string are printable |
| [isspace()](https://www.w3schools.com/python/ref_string_isspace.asp) | Returns True if all characters in the string are whitespaces |
| [istitle()](https://www.w3schools.com/python/ref_string_istitle.asp) | Returns True if the string follows the rules of a title |
| [isupper()](https://www.w3schools.com/python/ref_string_isupper.asp) | Returns True if all characters in the string are upper case |
| [join()](https://www.w3schools.com/python/ref_string_join.asp) | Joins the elements of an iterable to the end of the string |
| [ljust()](https://www.w3schools.com/python/ref_string_ljust.asp) | Returns a left justified version of the string |
| [lower()](https://www.w3schools.com/python/ref_string_lower.asp) | Converts a string into lower case |
| [lstrip()](https://www.w3schools.com/python/ref_string_lstrip.asp) | Returns a left trim version of the string |
| maketrans() | Returns a translation table to be used in translations |
| [partition()](https://www.w3schools.com/python/ref_string_partition.asp) | Returns a tuple where the string is parted into three parts |
| [replace()](https://www.w3schools.com/python/ref_string_replace.asp) | Returns a string where a specified value is replaced with a specified value |
| [rfind()](https://www.w3schools.com/python/ref_string_rfind.asp) | Searches the string for a specified value and returns the last position of where it was found |
| [rindex()](https://www.w3schools.com/python/ref_string_rindex.asp) | Searches the string for a specified value and returns the last position of where it was found |
| [rjust()](https://www.w3schools.com/python/ref_string_rjust.asp) | Returns a right justified version of the string |
| [rpartition()](https://www.w3schools.com/python/ref_string_rpartition.asp) | Returns a tuple where the string is parted into three parts |
| [rsplit()](https://www.w3schools.com/python/ref_string_rsplit.asp) | Splits the string at the specified separator, and returns a list |
| [rstrip()](https://www.w3schools.com/python/ref_string_rstrip.asp) | Returns a right trim version of the string |
| [split()](https://www.w3schools.com/python/ref_string_split.asp) | Splits the string at the specified separator, and returns a list |
| [splitlines()](https://www.w3schools.com/python/ref_string_splitlines.asp) | Splits the string at line breaks and returns a list |
| [startswith()](https://www.w3schools.com/python/ref_string_startswith.asp) | Returns true if the string starts with the specified value |
| [strip()](https://www.w3schools.com/python/ref_string_strip.asp) | Returns a trimmed version of the string |
| [swapcase()](https://www.w3schools.com/python/ref_string_swapcase.asp) | Swaps cases, lower case becomes upper case and vice versa |
| [title()](https://www.w3schools.com/python/ref_string_title.asp) | Converts the first character of each word to upper case |
| translate() | Returns a translated string |
| [upper()](https://www.w3schools.com/python/ref_string_upper.asp) | Converts a string into upper case |
| [zfill()](https://www.w3schools.com/python/ref_string_zfill.asp) | Fills the string with a specified number of 0 values at the beginning |

# **Python Booleans**

Booleans represent one of two values: True or False.

## Boolean Values

In programming you often need to know if an expression is True or False.

You can evaluate any expression in Python, and get one of two answers, True or False.

When you compare two values, the expression is evaluated and Python returns the Boolean answer:

### Example

print(10 > 9)  
print(10 == 9)  
print(10 < 9)

When you run a condition in an if statement, Python returns True or False:

### Example

Print a message based on whether the condition is True or False:

a = 200  
b = 33  
  
if b > a:  
  print("b is greater than a")  
else:  
  print("b is not greater than a")

## Evaluate Values and Variables

The bool() function allows you to evaluate any value, and give you True or False in return,

### Example

Evaluate a string and a number:

print(bool("Hello"))  
print(bool(15))

### Example

Evaluate two variables:

x = "Hello"  
y = 15  
  
print(bool(x))  
print(bool(y))

## Most Values are True

Almost any value is evaluated to True if it has some sort of content.

Any string is True, except empty strings.

Any number is True, except 0.

Any list, tuple, set, and dictionary are True, except empty ones.

### Example

The following will return True:

bool("abc")  
bool(123)  
bool(["apple", "cherry", "banana"])

## Some Values are False

In fact, there are not many values that evaluates to False, except empty values, such as (), [], {}, "", the number 0, and the value None. And of course the value False evaluates to False.

### Example

The following will return False:

bool(False)  
bool(None)  
bool(0)  
bool("")  
bool(())  
bool([])  
bool({})

One more value, or object in this case, evaluates to False, and that is if you have an objects that are made from a class with a \_\_len\_\_ function that returns 0 or False:

### Example

class myclass():  
  def \_\_len\_\_(self):  
    return 0  
  
myobj = myclass()  
print(bool(myobj))

## Functions can Return a Boolean

Python also has many built-in functions that returns a boolean value, like the isinstance() function, which can be used to determine if an object is of a certain data type:

### Example

Check if an object is an integer or not:

x = 200  
print(isinstance(x, int))

## Python Operators

Operators are used to perform operations on variables and values.

Python divides the operators in the following groups:

* Arithmetic operators
* Assignment operators
* Comparison operators
* Logical operators
* Identity operators
* Membership operators
* Bitwise operators

## Python Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Try it** |
| + | Addition | x + y | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_add) |
| - | Subtraction | x - y | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_sub) |
| \* | Multiplication | x \* y | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_mult) |
| / | Division | x / y | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_div) |
| % | Modulus | x % y | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_mod) |
| \*\* | Exponentiation | x \*\* y | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_exp) |
| // | Floor division | x // y | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_floordiv) |

## Python Assignment Operators

Assignment operators are used to assign values to variables:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Example** | **Same As** | **Try it** |
| = | x = 5 | x = 5 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_ass1) |
| += | x += 3 | x = x + 3 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_ass2) |
| -= | x -= 3 | x = x - 3 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_ass3) |
| \*= | x \*= 3 | x = x \* 3 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_ass4) |
| /= | x /= 3 | x = x / 3 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_ass5) |
| %= | x %= 3 | x = x % 3 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_ass6) |
| //= | x //= 3 | x = x // 3 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_ass7) |
| \*\*= | x \*\*= 3 | x = x \*\* 3 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_ass8) |
| &= | x &= 3 | x = x & 3 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_ass9) |
| |= | x |= 3 | x = x | 3 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_ass10) |
| ^= | x ^= 3 | x = x ^ 3 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_ass11) |
| >>= | x >>= 3 | x = x >> 3 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_ass12) |
| <<= | x <<= 3 | x = x << 3 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_ass13) |

## Python Comparison Operators

Comparison operators are used to compare two values:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Try it** |
| == | Equal | x == y | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_compare1) |
| != | Not equal | x != y | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_compare2) |
| > | Greater than | x > y | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_compare4) |
| < | Less than | x < y | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_compare5) |
| >= | Greater than or equal to | x >= y | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_compare6) |
| <= | Less than or equal to | x <= y | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_compare7) |

## Python Logical Operators

Logical operators are used to combine conditional statements:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Description** | **Example** | **Try it** |
| and | Returns True if both statements are true | x < 5 and  x < 10 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_logical1) |
| or | Returns True if one of the statements is true | x < 5 or x < 4 | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_logical2) |
| not | Reverse the result, returns False if the result is true | not(x < 5 and x < 10) | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_logical3) |

## Python Identity Operators

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Description** | **Example** | **Try it** |
| is | Returns true if both variables are the same object | x is y | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_identity1) |
| is not | Returns true if both variables are not the same object | x is not y | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_identity2) |

## Python Membership Operators

Membership operators are used to test if a sequence is presented in an object:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Description** | **Example** | **Try it** |
| in | Returns True if a sequence with the specified value is present in the object | x in y | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_membership1) |
| not in | Returns True if a sequence with the specified value is not present in the object | x not in y | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_oper_membership2) |

## Python Bitwise Operators

Bitwise operators are used to compare (binary) numbers:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Name** | **Description** |
| & | AND | Sets each bit to 1 if both bits are 1 |
| | | OR | Sets each bit to 1 if one of two bits is 1 |
| ^ | XOR | Sets each bit to 1 if only one of two bits is 1 |
| ~ | NOT | Inverts all the bits |
| << | Zero fill left shift | Shift left by pushing zeros in from the right and let the leftmost bits fall off |
| >> | Signed right shift | Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off |

# **Python Lists**

## Python Collections (Arrays)

There are four collection data types in the Python programming language:

* **List** is a collection which is ordered and changeable. Allows duplicate members.
* **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
* **Set** is a collection which is unordered and unindexed. No duplicate members.
* **Dictionary** is a collection which is unordered, changeable and indexed. No duplicate members.

When choosing a collection type, it is useful to understand the properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and, it could mean an increase in efficiency or security.

## List

A list is a collection which is ordered and changeable. In Python lists are written with square brackets.

### Example

Create a List:

thislist = ["apple", "banana", "cherry"]  
print(thislist)

## Access Items

You access the list items by referring to the index number:

### Example

Print the second item of the list:

thislist = ["apple", "banana", "cherry"]  
print(thislist[1])

### Negative Indexing

Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second last item etc.

### Example

Print the last item of the list:

thislist = ["apple", "banana", "cherry"]  
print(thislist[-1])

### Range of Indexes

You can specify a range of indexes by specifying where to start and where to end the range.

When specifying a range, the return value will be a new list with the specified items.

### Example

Return the third, fourth, and fifth item:

thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]  
print(thislist[2:5])

**Note:** The search will start at index 2 (included) and end at index 5 (not included).

Remember that the first item has index 0.

### Range of Negative Indexes

Specify negative indexes if you want to start the search from the end of the list:

### Example

This example returns the items from index -4 (included) to index -1 (excluded)

thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]  
print(thislist[-4:-1])

## Change Item Value

To change the value of a specific item, refer to the index number:

### Example

Change the second item:

thislist = ["apple", "banana", "cherry"]  
thislist[1] = "blackcurrant"  
print(thislist)

## Loop Through a List

You can loop through the list items by using a for loop:

### Example

Print all items in the list, one by one:

thislist = ["apple", "banana", "cherry"]  
for x in thislist:  
  print(x)

## Check if Item Exists

To determine if a specified item is present in a list use the in keyword:

### Example

Check if "apple" is present in the list:

thislist = ["apple", "banana", "cherry"]  
if "apple" in thislist:  
  print("Yes, 'apple' is in the fruits list")

## List Length

To determine how many items a list has, use the len() function:

### Example

Print the number of items in the list:

thislist = ["apple", "banana", "cherry"]  
print(len(thislist))

## Add Items

To add an item to the end of the list, use the append() method:

### Example

Using the append() method to append an item:

thislist = ["apple", "banana", "cherry"]  
thislist.append("orange")  
print(thislist)

To add an item at the specified index, use the insert() method:

### Example

Insert an item as the second position:

thislist = ["apple", "banana", "cherry"]  
thislist.insert(1, "orange")  
print(thislist)

## Remove Item

There are several methods to remove items from a list:

### Example

The remove() method removes the specified item:

thislist = ["apple", "banana", "cherry"]  
thislist.remove("banana")  
print(thislist)

### Example

The pop() method removes the specified index, (or the last item if index is not specified):

thislist = ["apple", "banana", "cherry"]  
thislist.pop()  
print(thislist)

### Example

The del keyword removes the specified index:

thislist = ["apple", "banana", "cherry"]  
del thislist[0]  
print(thislist)

### Example

The del keyword can also delete the list completely:

thislist = ["apple", "banana", "cherry"]  
del thislist

### Example

The clear() method empties the list:

thislist = ["apple", "banana", "cherry"]  
thislist.clear()  
print(thislist)

## Copy a List

You cannot copy a list simply by typing list2 = list1, because: list2 will only be a reference to list1, and changes made in list1 will automatically also be made in list2.

There are ways to make a copy, one way is to use the built-in List method copy().

### Example

Make a copy of a list with the copy() method:

thislist = ["apple", "banana", "cherry"]  
mylist = thislist.copy()  
print(mylist)

Another way to make a copy is to use the built-in method list().

### Example

Make a copy of a list with the list() method:

thislist = ["apple", "banana", "cherry"]  
mylist = list(thislist)  
print(mylist)

## Join Two Lists

There are several ways to join, or concatenate, two or more lists in Python.

One of the easiest ways are by using the + operator.

### Example

Join two list:

list1 = ["a", "b" , "c"]  
list2 = [1, 2, 3]  
  
list3 = list1 + list2  
print(list3)

Another way to join two lists are by appending all the items from list2 into list1, one by one:

### Example

Append list2 into list1:

list1 = ["a", "b" , "c"]  
list2 = [1, 2, 3]  
  
for x in list2:  
  list1.append(x)  
  
print(list1)

Or you can use the extend() method, which purpose is to add elements from one list to another list:

### Example

Use the extend() method to add list2 at the end of list1:

list1 = ["a", "b" , "c"]  
list2 = [1, 2, 3]  
  
list1.extend(list2)  
print(list1)

## The list() Constructor

It is also possible to use the list() constructor to make a new list.

### Example

Using the list() constructor to make a List:

thislist = list(("apple", "banana", "cherry")) # note the double round-brackets  
print(thislist)

## List Methods

Python has a set of built-in methods that you can use on lists.

|  |  |
| --- | --- |
| **Method** | **Description** |
| [append()](https://www.w3schools.com/python/ref_list_append.asp) | Adds an element at the end of the list |
| [clear()](https://www.w3schools.com/python/ref_list_clear.asp) | Removes all the elements from the list |
| [copy()](https://www.w3schools.com/python/ref_list_copy.asp) | Returns a copy of the list |
| [count()](https://www.w3schools.com/python/ref_list_count.asp) | Returns the number of elements with the specified value |
| [extend()](https://www.w3schools.com/python/ref_list_extend.asp) | Add the elements of a list (or any iterable), to the end of the current list |
| [index()](https://www.w3schools.com/python/ref_list_index.asp) | Returns the index of the first element with the specified value |
| [insert()](https://www.w3schools.com/python/ref_list_insert.asp) | Adds an element at the specified position |
| [pop()](https://www.w3schools.com/python/ref_list_pop.asp) | Removes the element at the specified position |
| [remove()](https://www.w3schools.com/python/ref_list_remove.asp) | Removes the item with the specified value |
| [reverse()](https://www.w3schools.com/python/ref_list_reverse.asp) | Reverses the order of the list |
| [sort()](https://www.w3schools.com/python/ref_list_sort.asp) | Sorts the list |

# **Python Tuples**

## Tuple

A tuple is a collection which is ordered and **unchangeable**. In Python tuples are written with round brackets.

### Example

Create a Tuple:

thistuple = ("apple", "banana", "cherry")  
print(thistuple)

## Access Tuple Items

You can access tuple items by referring to the index number, inside square brackets:

### Example

Print the second item in the tuple:

thistuple = ("apple", "banana", "cherry")  
print(thistuple[1])

### Negative Indexing

Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second last item etc.

### Example

Print the last item of the tuple:

thistuple = ("apple", "banana", "cherry")  
print(thistuple[-1])

### Range of Indexes

You can specify a range of indexes by specifying where to start and where to end the range.

When specifying a range, the return value will be a new tuple with the specified items.

### Example

Return the third, fourth, and fifth item:

thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")  
print(thistuple[2:5])

**Note:** The search will start at index 2 (included) and end at index 5 (not included).

Remember that the first item has index 0.

### Range of Negative Indexes

Specify negative indexes if you want to start the search from the end of the tuple:

### Example

This example returns the items from index -4 (included) to index -1 (excluded)

thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")  
print(thistuple[-4:-1])

## Change Tuple Values

Once a tuple is created, you cannot change its values. Tuples are **unchangeable**, or **immutable** as it also is called.

But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple.

### Example

Convert the tuple into a list to be able to change it:

x = ("apple", "banana", "cherry")  
y = list(x)  
y[1] = "kiwi"  
x = tuple(y)  
  
print(x)

## Loop Through a Tuple

You can loop through the tuple items by using a for loop.

### Example

Iterate through the items and print the values:

thistuple = ("apple", "banana", "cherry")  
for x in thistuple:  
  print(x)

## Check if Item Exists

To determine if a specified item is present in a tuple use the in keyword:

### Example

Check if "apple" is present in the tuple:

thistuple = ("apple", "banana", "cherry")  
if "apple" in thistuple:  
  print("Yes, 'apple' is in the fruits tuple")

## Tuple Length

To determine how many items a tuple has, use the len() method:

### Example

Print the number of items in the tuple:

thistuple = ("apple", "banana", "cherry")  
print(len(thistuple))

## Add Items

Once a tuple is created, you cannot add items to it. Tuples are **unchangeable**.

### Example

You cannot add items to a tuple:

thistuple = ("apple", "banana", "cherry")  
thistuple[3] = "orange" # This will raise an error  
print(thistuple)

## Create Tuple With One Item

To create a tuple with only one item, you have add a comma after the item, unless Python will not recognize the variable as a tuple.

### Example

One item tuple, remember the commma:

thistuple = ("apple",)  
print(type(thistuple))  
  
#NOT a tuple  
thistuple = ("apple")  
print(type(thistuple))

## Remove Items

**Note:** You cannot remove items in a tuple.

 Tuples are **unchangeable**, so you cannot remove items from it, but you can delete the tuple completely:

### Example

The del keyword can delete the tuple completely:

thistuple = ("apple", "banana", "cherry")  
del thistuple  
print(thistuple) #this will raise an error because the tuple no longer exists

## Join Two Tuples

To join two or more tuples you can use the + operator:

### Example

Join two tuples:

tuple1 = ("a", "b" , "c")  
tuple2 = (1, 2, 3)  
  
tuple3 = tuple1 + tuple2  
print(tuple3)

## The tuple() Constructor

It is also possible to use the tuple() constructor to make a tuple.

### Example

Using the tuple() method to make a tuple:

thistuple = tuple(("apple", "banana", "cherry")) # note the double round-brackets  
print(thistuple)

## Tuple Methods

Python has two built-in methods that you can use on tuples.

|  |  |
| --- | --- |
| **Method** | **Description** |
| [count()](https://www.w3schools.com/python/ref_tuple_count.asp) | Returns the number of times a specified value occurs in a tuple |
| [index()](https://www.w3schools.com/python/ref_tuple_index.asp) | Searches the tuple for a specified value and returns the position of where it was found |

# **Python Sets**

## Set

A set is a collection which is unordered and unindexed. In Python sets are written with curly brackets.

### Example

Create a Set:

thisset = {"apple", "banana", "cherry"}  
print(thisset)

**Note:** Sets are unordered, so you cannot be sure in which order the items will appear.

## Access Items

You cannot access items in a set by referring to an index, since sets are unordered the items has no index.

But you can loop through the set items using a for loop, or ask if a specified value is present in a set, by using the in keyword.

### Example

Loop through the set, and print the values:

thisset = {"apple", "banana", "cherry"}  
  
for x in thisset:  
  print(x)

### Example

Check if "banana" is present in the set:

thisset = {"apple", "banana", "cherry"}  
  
print("banana" in thisset)

## Change Items

Once a set is created, you cannot change its items, but you can add new items.

## Add Items

To add one item to a set use the add() method.

To add more than one item to a set use the update() method.

### Example

Add an item to a set, using the add() method:

thisset = {"apple", "banana", "cherry"}  
  
thisset.add("orange")  
  
print(thisset)

### Example

Add multiple items to a set, using the update() method:

thisset = {"apple", "banana", "cherry"}  
  
thisset.update(["orange", "mango", "grapes"])  
  
print(thisset)

## Get the Length of a Set

To determine how many items a set has, use the len() method.

### Example

Get the number of items in a set:

thisset = {"apple", "banana", "cherry"}  
  
print(len(thisset))

## Remove Item

To remove an item in a set, use the remove(), or the discard() method.

### Example

Remove "banana" by using the remove() method:

thisset = {"apple", "banana", "cherry"}  
  
thisset.remove("banana")  
  
print(thisset)

**Note:** If the item to remove does not exist, remove() will raise an error.

### Example

Remove "banana" by using the discard() method:

thisset = {"apple", "banana", "cherry"}  
  
thisset.discard("banana")  
  
print(thisset)

**Note:** If the item to remove does not exist, discard() will **NOT** raise an error.

You can also use the pop(), method to remove an item, but this method will remove the last item. Remember that sets are unordered, so you will not know what item that gets removed.

The return value of the pop() method is the removed item.

### Example

Remove the last item by using the pop() method:

thisset = {"apple", "banana", "cherry"}  
  
x = thisset.pop()  
  
print(x)  
  
print(thisset)

**Note:** Sets are unordered, so when using the pop() method, you will not know which item that gets removed.

### Example

The clear() method empties the set:

thisset = {"apple", "banana", "cherry"}  
  
thisset.clear()  
  
print(thisset)

### Example

The del keyword will delete the set completely:

thisset = {"apple", "banana", "cherry"}  
  
del thisset  
  
print(thisset)

## Join Two Sets

There are several ways to join two or more sets in Python.

You can use the union() method that returns a new set containing all items from both sets, or the update()method that inserts all the items from one set into another:

### Example

The union() method returns a new set with all items from both sets:

set1 = {"a", "b" , "c"}  
set2 = {1, 2, 3}  
  
set3 = set1.union(set2)  
print(set3)

### Example

The update() method inserts the items in set2 into set1:

set1 = {"a", "b" , "c"}  
set2 = {1, 2, 3}  
  
set1.update(set2)  
print(set1)

**Note:** Both union() and update() will exclude any duplicate items.

There are other methods that joins two sets and keeps ONLY the duplicates, or NEVER the duplicates, check the full list of set methods in the bottom of this page.

## The set() Constructor

It is also possible to use the set() constructor to make a set.

### Example

Using the set() constructor to make a set:

thisset = set(("apple", "banana", "cherry")) # note the double round-brackets  
print(thisset)

## Set Methods

Python has a set of built-in methods that you can use on sets.

|  |  |
| --- | --- |
| **Method** | **Description** |
| [add()](https://www.w3schools.com/python/ref_set_add.asp) | Adds an element to the set |
| [clear()](https://www.w3schools.com/python/ref_set_clear.asp) | Removes all the elements from the set |
| [copy()](https://www.w3schools.com/python/ref_set_copy.asp) | Returns a copy of the set |
| [difference()](https://www.w3schools.com/python/ref_set_difference.asp) | Returns a set containing the difference between two or more sets |
| [difference\_update()](https://www.w3schools.com/python/ref_set_difference_update.asp) | Removes the items in this set that are also included in another, specified set |
| [discard()](https://www.w3schools.com/python/ref_set_discard.asp) | Remove the specified item |
| [intersection()](https://www.w3schools.com/python/ref_set_intersection.asp) | Returns a set, that is the intersection of two other sets |
| [intersection\_update()](https://www.w3schools.com/python/ref_set_intersection_update.asp) | Removes the items in this set that are not present in other, specified set(s) |
| [isdisjoint()](https://www.w3schools.com/python/ref_set_isdisjoint.asp) | Returns whether two sets have a intersection or not |
| [issubset()](https://www.w3schools.com/python/ref_set_issubset.asp) | Returns whether another set contains this set or not |
| [issuperset()](https://www.w3schools.com/python/ref_set_issuperset.asp) | Returns whether this set contains another set or not |
| [pop()](https://www.w3schools.com/python/ref_set_pop.asp) | Removes an element from the set |
| [remove()](https://www.w3schools.com/python/ref_set_remove.asp) | Removes the specified element |
| [symmetric\_difference()](https://www.w3schools.com/python/ref_set_symmetric_difference.asp) | Returns a set with the symmetric differences of two sets |
| [symmetric\_difference\_update()](https://www.w3schools.com/python/ref_set_symmetric_difference_update.asp) | inserts the symmetric differences from this set and another |
| [union()](https://www.w3schools.com/python/ref_set_union.asp) | Return a set containing the union of sets |
| [update()](https://www.w3schools.com/python/ref_set_update.asp) | Update the set with the union of this set and others |

# **Python Dictionaries**

## Dictionary

A dictionary is a collection which is unordered, changeable and indexed. In Python dictionaries are written with curly brackets, and they have keys and values.

### Example

Create and print a dictionary:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
print(thisdict)

## Accessing Items

You can access the items of a dictionary by referring to its key name, inside square brackets:

### Example

Get the value of the "model" key:

x = thisdict["model"]

There is also a method called get() that will give you the same result:

### Example

Get the value of the "model" key:

x = thisdict.get("model")

## Change Values

You can change the value of a specific item by referring to its key name:

### Example

Change the "year" to 2018:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict["year"] = 2018

## Loop Through a Dictionary

You can loop through a dictionary by using a for loop.

When looping through a dictionary, the return value are the keys of the dictionary, but there are methods to return the values as well.

### Example

Print all key names in the dictionary, one by one:

for x in thisdict:  
  print(x)

### Example

Print all values in the dictionary, one by one:

for x in thisdict:  
  print(thisdict[x])

### Example

You can also use the values() function to return values of a dictionary:

for x in thisdict.values():  
  print(x)

### Example

Loop through both keys and values, by using the items() function:

for x, y in thisdict.items():  
  print(x, y)

## Check if Key Exists

To determine if a specified key is present in a dictionary use the in keyword:

### Example

Check if "model" is present in the dictionary:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
if "model" in thisdict:  
  print("Yes, 'model' is one of the keys in the thisdict dictionary")

## Dictionary Length

To determine how many items (key-value pairs) a dictionary has, use the len() method.

### Example

Print the number of items in the dictionary:

print(len(thisdict))

## Adding Items

Adding an item to the dictionary is done by using a new index key and assigning a value to it:

### Example

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict["color"] = "red"  
print(thisdict)

## Removing Items

There are several methods to remove items from a dictionary:

### Example

The pop() method removes the item with the specified key name:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict.pop("model")  
print(thisdict)

### Example

The popitem() method removes the last inserted item (in versions before 3.7, a random item is removed instead):

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict.popitem()  
print(thisdict)

### Example

The del keyword removes the item with the specified key name:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
del thisdict["model"]  
print(thisdict)

### Example

The del keyword can also delete the dictionary completely:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
del thisdict  
print(thisdict) #this will cause an error because "thisdict" no longer exists.

### Example

The clear() keyword empties the dictionary:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict.clear()  
print(thisdict)

## Copy a Dictionary

You cannot copy a dictionary simply by typing dict2 = dict1, because: dict2 will only be a reference to dict1, and changes made in dict1 will automatically also be made in dict2.

There are ways to make a copy, one way is to use the built-in Dictionary method copy().

### Example

Make a copy of a dictionary with the copy() method:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
mydict = thisdict.copy()  
print(mydict)

Another way to make a copy is to use the built-in method dict().

### Example

Make a copy of a dictionary with the dict() method:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
mydict = dict(thisdict)  
print(mydict)

## Nested Dictionaries

A dictionary can also contain many dictionaries, this is called nested dictionaries.

### Example

Create a dictionary that contain three dictionaries:

myfamily = {  
  "child1" : {  
    "name" : "Emil",  
    "year" : 2004  
  },  
  "child2" : {  
    "name" : "Tobias",  
    "year" : 2007  
  },  
  "child3" : {  
    "name" : "Linus",  
    "year" : 2011  
  }  
}

Or, if you want to nest three dictionaries that already exists as dictionaries:

### Example

Create three dictionaries, than create one dictionary that will contain the other three dictionaries:

child1 = {  
  "name" : "Emil",  
  "year" : 2004  
}  
child2 = {  
  "name" : "Tobias",  
  "year" : 2007  
}  
child3 = {  
  "name" : "Linus",  
  "year" : 2011  
}  
  
myfamily = {  
  "child1" : child1,  
  "child2" : child2,  
  "child3" : child3  
}

## The dict() Constructor

It is also possible to use the dict() constructor to make a new dictionary:

### Example

thisdict = dict(brand="Ford", model="Mustang", year=1964)  
# note that keywords are not string literals  
# note the use of equals rather than colon for the assignment  
print(thisdict)

## Dictionary Methods

Python has a set of built-in methods that you can use on dictionaries.

|  |  |
| --- | --- |
| **Method** | **Description** |
| [clear()](https://www.w3schools.com/python/ref_dictionary_clear.asp) | Removes all the elements from the dictionary |
| [copy()](https://www.w3schools.com/python/ref_dictionary_copy.asp) | Returns a copy of the dictionary |
| [fromkeys()](https://www.w3schools.com/python/ref_dictionary_fromkeys.asp) | Returns a dictionary with the specified keys and values |
| [get()](https://www.w3schools.com/python/ref_dictionary_get.asp) | Returns the value of the specified key |
| [items()](https://www.w3schools.com/python/ref_dictionary_items.asp) | Returns a list containing a tuple for each key value pair |
| [keys()](https://www.w3schools.com/python/ref_dictionary_keys.asp) | Returns a list containing the dictionary's keys |
| [pop()](https://www.w3schools.com/python/ref_dictionary_pop.asp) | Removes the element with the specified key |
| [popitem()](https://www.w3schools.com/python/ref_dictionary_popitem.asp) | Removes the last inserted key-value pair |
| [setdefault()](https://www.w3schools.com/python/ref_dictionary_setdefault.asp) | Returns the value of the specified key. If the key does not exist: insert the key, with the specified value |
| [update()](https://www.w3schools.com/python/ref_dictionary_update.asp) | Updates the dictionary with the specified key-value pairs |
| [values()](https://www.w3schools.com/python/ref_dictionary_values.asp) | Returns a list of all the values in the dictionary |

# **Python If-else statements**

Decision making is the most important aspect of almost all the programming languages. As the name implies, decision making allows us to run a particular block of code for a particular decision. Here, the decisions are made on the validity of the particular conditions. Condition checking is the backbone of decision making.

In python, decision making is performed by the following statements.

|  |  |
| --- | --- |
| **Statement** | **Description** |
|  |  |
| If Statement | The if statement is used to test a specific condition. If the condition is true, a block of code (if-block) will be executed. |
| If - else Statement | The if-else statement is similar to if statement except the fact that, it also provides the block of the code for the false case of the condition to be checked. If the condition provided in the if statement is false, then the else statement will be executed. |
| Nested if Statement | Nested if statements enable us to use if ? else statement inside an outer if statement. |

## Indentation in Python

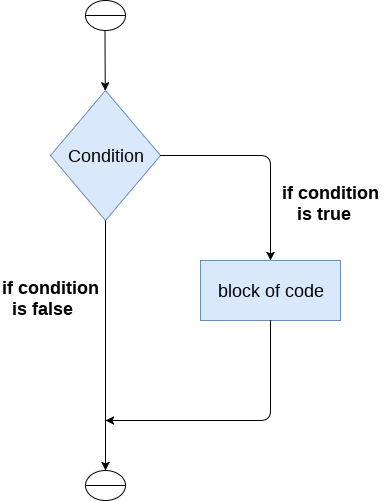
For the ease of programming and to achieve simplicity, python doesn't allow the use of parentheses for the block level code. In Python, indentation is used to declare a block. If two statements are at the same indentation level, then they are the part of the same block.

Generally, four spaces are given to indent the statements which are a typical amount of indentation in python.

Indentation is the most used part of the python language since it declares the block of code. All the statements of one block are intended at the same level indentation. We will see how the actual indentation takes place in decision making and other stuff in python.

## The if statement

The if statement is used to test a particular condition and if the condition is true, it executes a block of code known as if-block. The condition of if statement can be any valid logical expression which can be either evaluated to true or false.

The syntax of the if-statement is given below.

**if** expression:

    statement

### Example 1

num = int(input("enter the number?"))

**if** num%2 == 0:

**print**("Number is even")

**Output:**

enter the number?10

Number is even

### Example 2 : Program to print the largest of the three numbers.

a = int(input("Enter a? "));

b = int(input("Enter b? "));

c = int(input("Enter c? "));

**if** a>b **and** a>c:

**print**("a is largest");

**if** b>a **and** b>c:

**print**("b is largest");

**if** c>a **and** c>b:

**print**("c is largest");

**Output:**

Enter a? 100

Enter b? 120

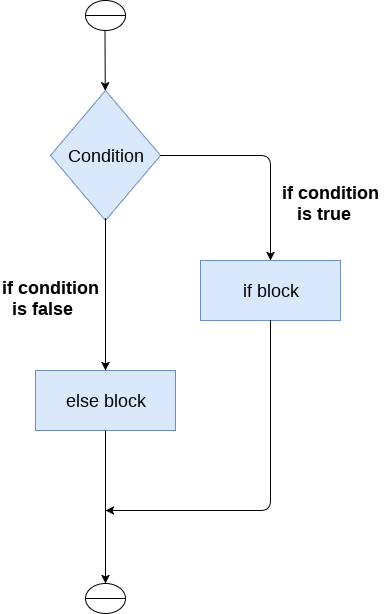
Enter c? 130

c is largest

## The if-else statement

The if-else statement provides an else block combined with the if statement which is executed in the false case of the condition.

If the condition is true, then the if-block is executed. Otherwise, the else-block is executed.

The syntax of the if-else statement is given below.

**if** condition:

    #block of statements

**else**:

    #another block of statements (else-block)

### Example 1 : Program to check whether a person is eligible to vote or not.

age = int (input("Enter your age? "))

**if** age>=18:

**print**("You are eligible to vote !!");

**else**:

**print**("Sorry! you have to wait !!");

**Output:**

Enter your age? 90

You are eligible to vote !!

### Example 2: Program to check whether a number is even or not.

num = int(input("enter the number?"))

**if** num%2 == 0:

**print**("Number is even...")

**else**:

**print**("Number is odd...")

**Output:**

enter the number?10

Number is even

## The elif statement

The elif statement enables us to check multiple conditions and execute the specific block of statements depending upon the true condition among them. We can have any number of elif statements in our program depending upon our need. However, using elif is optional.

The elif statement works like an if-else-if ladder statement in C. It must be succeeded by an if statement.

The syntax of the elif statement is given below.

**if** expression 1:

    # block of statements

**elif** expression 2:

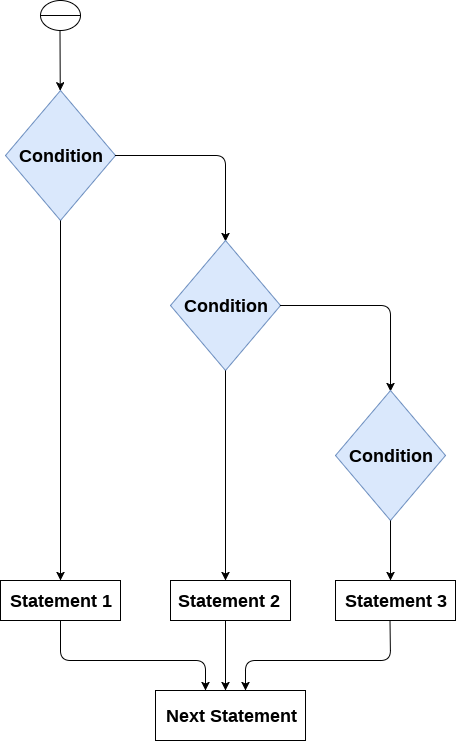
    # block of statements

**elif** expression 3:

    # block of statements

**else**:

    # block of statements



### Example 1

number = int(input("Enter the number?"))

**if** number==10:

**print**("number is equals to 10")

**elif** number==50:

**print**("number is equal to 50");

**elif** number==100:

**print**("number is equal to 100");

**else**:

**print**("number is not equal to 10, 50 or 100");

**Output:**

Enter the number?15

number is not equal to 10, 50 or 100

### Example 2

marks = int(input("Enter the marks? "))

if marks > 85 **and** marks <= 100:

**print**("Congrats ! you scored grade A ...")

elif marks > 60 **and** marks <= 85:

**print**("You scored grade B + ...")

elif marks > 40 **and** marks <= 60:

**print**("You scored grade B ...")

elif (marks > 30 **and** marks <= 40):

**print**("You scored grade C ...")

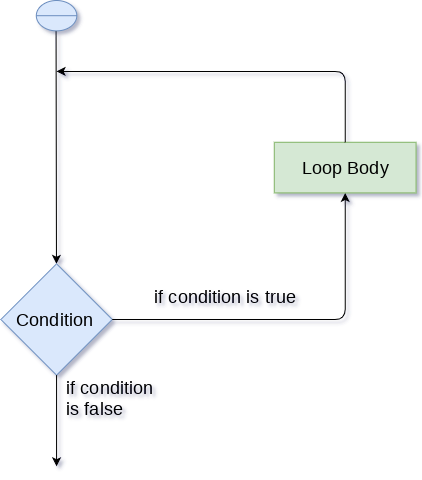
else:

**print**("Sorry you are fail ?")

# **Python Loops**

The flow of the programs written in any programming language is sequential by default. Sometimes we may need to alter the flow of the program. The execution of a specific code may need to be repeated several numbers of times.

For this purpose, The programming languages provide various types of loops which are capable of repeating some specific code several numbers of times. Consider the following diagram to understand the working of a loop statement.



## Why we use loops in python?

The looping simplifies the complex problems into the easy ones. It enables us to alter the flow of the program so that instead of writing the same code again and again, we can repeat the same code for a finite number of times. For example, if we need to print the first 10 natural numbers then, instead of using the print statement 10 times, we can print inside a loop which runs up to 10 iterations.

## Advantages of loops

There are the following advantages of loops in Python.

1. It provides code re-usability.
2. Using loops, we do not need to write the same code again and again.
3. Using loops, we can traverse over the elements of data structures (array or linked lists).

There are the following loop statements in Python.

|  |  |
| --- | --- |
| **Loop Statement** | **Description** |
| for loop | The for loop is used in the case where we need to execute some part of the code until the given condition is satisfied. The for loop is also called as a per-tested loop. It is better to use for loop if the number of iteration is known in advance. |
| while loop | The while loop is to be used in the scenario where we don't know the number of iterations in advance. The block of statements is executed in the while loop until the condition specified in the while loop is satisfied. It is also called a pre-tested loop. |

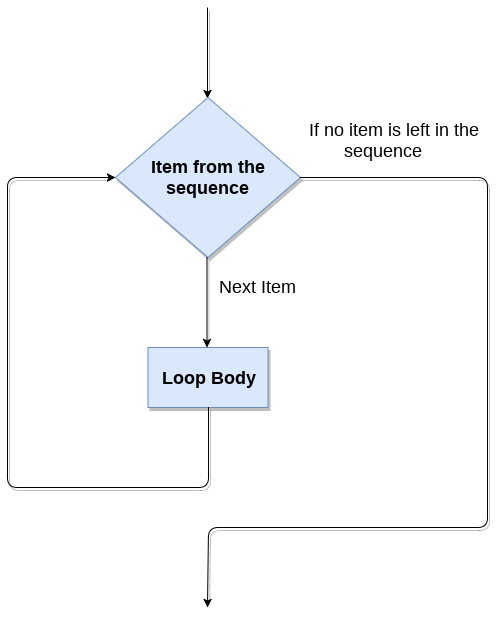
# **Python for loop**

The for **loop in Python** is used to iterate the statements or a part of the program several times. It is frequently used to traverse the data structures like list, tuple, or dictionary.

The syntax of for loop in python is given below.

**for** iterating\_var **in** sequence:

    statement(s)



## Example

i=1

n=int(input("Enter the number up to which you want to print the natural numbers?"))

**for** i **in** range(0,10):

**print**(i,end = ' ')

**Output:**

0 1 2 3 4 5 6 7 8 9

## Python for loop example : printing the table of the given number

i=1;

num = int(input("Enter a number:"));

**for** i **in** range(1,11):

**print**("%d X %d = %d"%(num,i,num\*i));

**Output:**

Enter a number:10

10 X 1 = 10

10 X 2 = 20

10 X 3 = 30

10 X 4 = 40

10 X 5 = 50

10 X 6 = 60

10 X 7 = 70

10 X 8 = 80

10 X 9 = 90

10 X 10 = 100

## Nested for loop in python

Python allows us to nest any number of for loops inside a for loop. The inner loop is executed n number of times for every iteration of the outer loop. The syntax of the nested for loop in python is given below.

**for** iterating\_var1 **in** sequence:

**for** iterating\_var2 **in** sequence:

        #block of statements

#Other statements

## Example 1

n = int(input("Enter the number of rows you want to print?"))

i,j=0,0

**for** i **in** range(0,n):

**print**()

**for** j **in** range(0,i+1):

**print**("\*",end="")

**Output:**

Enter the number of rows you want to print?5

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

## Using else statement with for loop

Unlike other languages like C, C++, or Java, python allows us to use the else statement with the for loop which can be executed only when all the iterations are exhausted. Here, we must notice that if the loop contains any of the break statement then the else statement will not be executed.

## Example 1

**for** i **in** range(0,5):

**print**(i)

**else**:**print**("for loop completely exhausted, since there is no break.");

In the above example, for loop is executed completely since there is no break statement in the loop. The control comes out of the loop and hence the else block is executed.

**Output:**

0

1

2

3

4

for loop completely exhausted, since there is no break.

## Example 2

**for** i **in** range(0,5):

**print**(i)

**break**;

**else**:**print**("for loop is exhausted");

**print**("The loop is broken due to break statement...came out of loop")

In the above example, the loop is broken due to break statement therefore the else statement will not be executed. The statement present immediate next to else block will be executed.

**Output:**

0

The loop is broken due to break statement...came out of loop

# **Python | Demonstrate an Example of break statement**

**Python break statement example:** Here, we are going to learn how to use break statement in the loops in Python?  
Submitted by [Pankaj Singh](https://www.includehelp.com/Members/Pankaj-Singh.aspx), on October 06, 2018

break is a keyword in python just like another programming language and it is used to break the execution of loop statement.

In the given example, loop is running from 1 to 10 and we are using the break statement if the value of i is 6. Thus when the value of i will be 6, program's execution will come out from the loop.

**Example 1:**

**for** i **in** range(1,11):

**if**(i==6):

**break**

**print**(i)

**Output**

1

2

3

4

5

**Example 2:** In this example, we are printing character by character of the value/string “Hello world” and terminating (using break), if the character is space.

**for** ch **in** "Hello world":

**if** ch == " ":

**break**

**print**(ch)

**Output**

H

e

l

l

o

# **Python | Demonstrate an Example of continue statement**

**Python continue statement example:** Here, we are going to learn how to use continue statement in the loops in Python?  
Submitted by [Pankaj Singh](https://www.includehelp.com/Members/Pankaj-Singh.aspx), on October 08, 2018

continue is a keyword in python just like another programming language and it is used to send the program’s section to loop by escaping the execution of next statement in the loop.

In the given example, loop is running from 1 to 10 and we are using the continue statement if value of ‘i’ is 6. This when the value of i will be 6, program’s execution will continue without prating the 6.

**Example 1:**

**for** i **in** range(1,11):

**if**(i==6):

**continue**

**print**(i)

**Output**

1

2

3

4

5

7

8

9

10

**Example 2:** In this example, we are printing character by character of the value/string “Hello world” and continuing the loop execution, if the character is space.

**for** ch **in** "Hello world":

**if** ch == " ":

**continue**

**print**(ch)

**Output**

H

e

l

l

o

w

o

r

l

d

# **Python | Demonstrate an Example of pass statement**

**Python pass statement with example:** Here, we are going to learn **what pass statement in python is and how and when it is used**?  
Submitted by [Pankaj Singh](https://www.includehelp.com/Members/Pankaj-Singh.aspx), on October 09, 2018

## pass statement in python

**"pass"** is a type of null operation or null statement, when it executes nothing happens. It is used when you want do not want to write any code/statement to execute but syntactically a statement is required.

**Let’s consider the given example...**

Here, we are using **"pass"** statement in the function **hello()** definition - there is no statement in the function and if we do not use **"pass"**, there will be an error.

**def** hello():

**pass**

hello()

**Example 2:**

**for** i **in** range(1,11):

**if**(i==6):

**continue**

**else**:

**pass**

**print**(i)

**Output**

1

2

3

4

5

7

8

9

10

# **Python while loop**

The while loop is also known as a pre-tested loop. In general, a while loop allows a part of the code to be executed as long as the given condition is true.

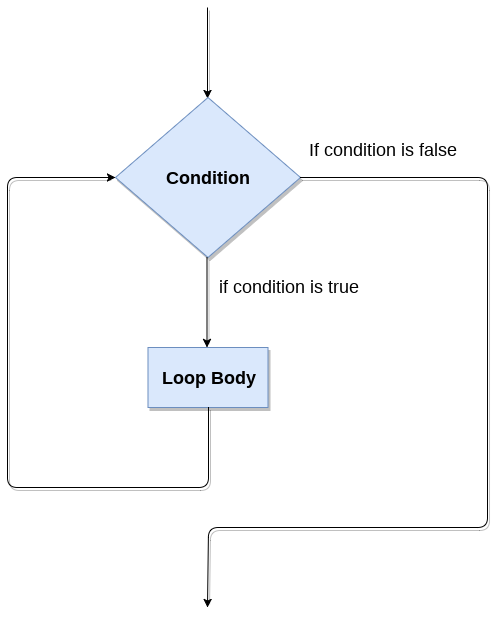
It can be viewed as a repeating if statement. The while loop is mostly used in the case where the number of iterations is not known in advance.

The syntax is given below.

**while** expression:

    statements

Here, the statements can be a single statement or the group of statements. The expression should be any valid python expression resulting into true or false. The true is any non-zero value.



## Example 1

i=1;

**while** i<=10:

**print**(i);

    i=i+1;

**Output:**

1

2

3

4

5

6

7

8

9

10

## Example 2

i=1

number=0

b=9

number = int(input("Enter the number?"))

**while** i<=10:

**print**("%d X %d = %d \n"%(number,i,number\*i));

    i = i+1;

**Output:**

Enter the number?10

10 X 1 = 10

10 X 2 = 20

10 X 3 = 30

10 X 4 = 40

10 X 5 = 50

10 X 6 = 60

10 X 7 = 70

10 X 8 = 80

10 X 9 = 90

10 X 10 = 100

## Infinite while loop

If the condition given in the while loop never becomes false then the while loop will never terminate and result into the infinite while loop.

Any non-zero value in the while loop indicates an always-true condition whereas 0 indicates the always-false condition. This type of approach is useful if we want our program to run continuously in the loop without any disturbance.

## Example 1

**while** (1):

**print**("Hi! we are inside the infinite while loop");

**Output:**

Hi! we are inside the infinite while loop

(infinite times)

## Example 2

var = 1

**while** var != 2:

    i = int(input("Enter the number?"))

**print** ("Entered value is %d"%(i))

**Output:**

Enter the number?102

Entered value is 102

Enter the number?102

Entered value is 102

Enter the number?103

Entered value is 103

Enter the number?103

(infinite loop)

## Using else with Python while loop

Python enables us to use the while loop with the while loop also. The else block is executed when the condition given in the while statement becomes false. Like for loop, if the while loop is broken using break statement, then the else block will not be executed and the statement present after else block will be executed.

Consider the following example.

i=1;

**while** i<=5:

**print**(i)

    i=i+1;

**else**:**print**("The while loop exhausted");

**Output:**

1

2

3

4

5

The while loop exhausted

## Example 2

i=1;

**while** i<=5:

**print**(i)

    i=i+1;

**if**(i==3):

**break**;

**else**:**print**("The while loop exhausted");

**Output:**

1

2

# **Python break statement**

The break is a keyword in python which is used to bring the program control out of the loop. The break statement breaks the loops one by one, i.e., in the case of nested loops, it breaks the inner loop first and then proceeds to outer loops. In other words, we can say that break is used to abort the current execution of the program and the control goes to the next line after the loop.

The break is commonly used in the cases where we need to break the loop for a given condition.

The syntax of the break is given below.

#loop statements

**break**;

**Example 1**

list =[1,2,3,4]

count = 1;

**for** i **in** list:

**if** i == 4:

**print**("item matched")

        count = count + 1;

**break**

**print**("found at",count,"location");

**Output:**

item matched

found at 2 location

## Example 2

str = "python"

**for** i **in** str:

**if** i == 'o':

**break**

**print**(i);

**Output:**

p

y

t

h

## Example 3: break statement with while loop

i = 0;

**while** 1:

**print**(i," ",end=""),

    i=i+1;

**if** i == 10:

**break**;

**print**("came out of while loop");

**Output:**

0 1 2 3 4 5 6 7 8 9 came out of while loop

## Example 3

n=2

**while** 1:

    i=1;

**while** i<=10:

**print**("%d X %d = %d\n"%(n,i,n\*i));

        i = i+1;

    choice = int(input("Do you want to continue printing the table, press 0 for no?"))

**if** choice == 0:

**break**;

    n=n+1

**Output:**

2 X 1 = 2

2 X 2 = 4

2 X 3 = 6

2 X 4 = 8

2 X 5 = 10

2 X 6 = 12

2 X 7 = 14

2 X 8 = 16

2 X 9 = 18

2 X 10 = 20

Do you want to continue printing the table, press 0 for no?1

3 X 1 = 3

3 X 2 = 6

3 X 3 = 9

3 X 4 = 12

3 X 5 = 15

3 X 6 = 18

3 X 7 = 21

3 X 8 = 24

3 X 9 = 27

3 X 10 = 30

Do you want to continue printing the table, press 0 for no?0

# **Python continue Statement**

The continue statement in python is used to bring the program control to the beginning of the loop. The continue statement skips the remaining lines of code inside the loop and start with the next iteration. It is mainly used for a particular condition inside the loop so that we can skip some specific code for a particular condition.

The syntax of Python continue statement is given below.

#loop statements

**continue**;

#the code to be skipped

## Example 1

i = 0;

**while** i!=10:

**print**("%d"%i);

**continue**;

    i=i+1;

**Output:**

infinite loop

## Example 2

i=1; #initializing a local variable

#starting a loop from 1 to 10

**for** i **in** range(1,11):

**if** i==5:

**continue**;

**print**("%d"%i);

**Output:**

1

2

3

4

6

7

8

9

10

## Pass Statement

The pass statement is a null operation since nothing happens when it is executed. It is used in the cases where a statement is syntactically needed but we don't want to use any executable statement at its place.

For example, it can be used while overriding a parent class method in the subclass but don't want to give its specific implementation in the subclass.

Pass is also used where the code will be written somewhere but not yet written in the program file.

The syntax of the pass statement is given below.

## Example

list = [1,2,3,4,5]

flag = 0

**for** i **in** list:

**print**("Current element:",i,end=" ");

**if** i==3:

**pass**;

**print**("\nWe are inside pass block\n");

        flag = 1;

**if** flag==1:

**print**("\nCame out of pass\n");

        flag=0;

**Output:**

Current element: 1 Current element: 2 Current element: 3

We are inside pass block

Came out of pass

Current element: 4 Current element: 5

# **Python Pass**

In Python, pass keyword is used to execute nothing; it means, when we don't want to execute code, the pass can be used to execute empty. It is same as the name refers to. It just makes the control to pass by without executing any code. If we want to bypass any code pass statement can be used.

**Python Pass Syntax**

**pass**

**Python Pass Example**

**for** i **in** [1,2,3,4,5]:

**if** i==3:

**pass**

**print** "Pass when value is",i

**print** i,

**Output:**

>>>

1 2 Pass when value **is** 3

3 4 5

>>>

# **Python Functions**

Functions are the most important aspect of an application. A function can be defined as the organized block of reusable code which can be called whenever required.

Python allows us to divide a large program into the basic building blocks known as function. The function contains the set of programming statements enclosed by {}. A function can be called multiple times to provide reusability and modularity to the python program.

In other words, we can say that the collection of functions creates a program. The function is also known as procedure or subroutine in other programming languages.

Python provide us various inbuilt functions like range() or print(). Although, the user can create its functions which can be called user-defined functions.

**Types of functions in Python**

There are two types of functions in Python.

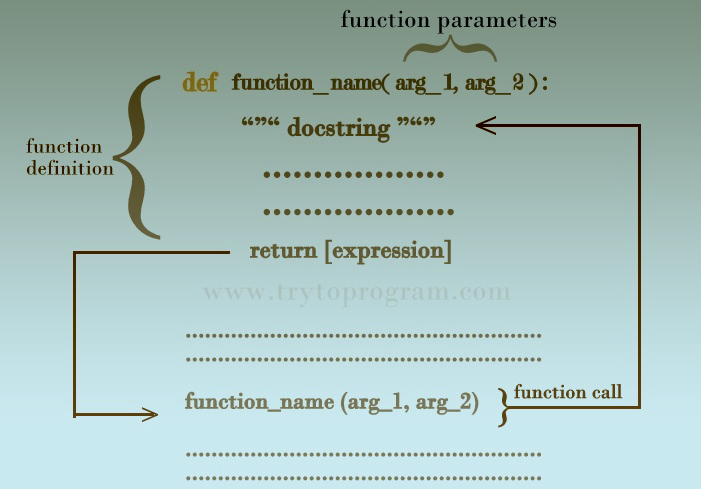
* **Built-in Functions**

These are the built-in functions of Python with pre-defined functionalities.

* **User-defined Functions**

As the name goes, these are the functions defined by the users as per the requirement at different stages.We define our own functionalities and give a name to these functions.User-defined function with return type has return statements to return values after calculation whereas, function with non-return type has no return statement.

## ****How does a function work in Python?****



As seen in above diagram, a Python function consists of **function definition** where the functionality of a function is defined. Function definition as seen above consists of a **function name,** **function arguments, docstring, code statements, and the return statement.**

Once a function is defined, we need to call the function in the main program to execute the function. For that, there is function call statement.

Let’s go through each [keyword](http://www.trytoprogram.com/python-programming/python-keywords-and-data-types/), expressions, and statements used in the function.

## ****Python Functions: Definition****

Let’s look at the syntax of function definition in Python and explain it bits by bits afterward.

def function\_name (arg 1, arg2,...):

"""docstring"""

statement(s)

return [expression]

* A function definition in Python starts with the keyword def followed by the function name.
* Depending on the operational requirement or user’s choice, a function may or may not have parameters inside the parentheses just after function name and it is followed by a colon (:).
* After that is an optional statement – the documentation string of function or docstring followed by the indented block of codes or statements designated for the operation of the function.
* The return statement is to exit the function and return the computed value, making the program flow to jump wherever the function is called.

## ****Function call****

Defining a function means giving the function a name, specifying the parameters and defining the operation of function with code statements.

Once a function is defined,  we need to call the function wherever and whenever we need to execute it. To call a function, we simply need to write the function and supply corresponding arguments for the parameters we included in the function definition.

Here is the basic syntax of a function call in Python.

function\_name(arg1, arg2,...)

## ****Docstring****

As we saw in the function definition, the first statement of the body function is a documentation string called doctsring, which explains in brief about the function and its operation. In Python, string literals are used for documenting modules.

We can access the docstring by \_\_doc\_\_ attribute. For example function\_name.\_\_doc\_\_.

**Note:**  The doc attribute has two underscores(\_\_) before and after. Using single underscore (\_) will raise an error.

Here is the example to show how we can access the docstring in Python functions.

def Hello():

""" Hello World """

print ('Hi')

print ("The doctsring of the function Hello is: "+ Hello.\_\_doc\_\_)

**Output:**

This script will generate following output.

The docstring of the function Hello is: Hello World

## ****The return statement****

The return statement ends the execution of the function and returns the value of expression following the keyword return to the caller.

### Syntax of return statement

return [expression]

If the return expression has no expression or if there is no return statement in the function, a special value 'None' is returned.

The return statement in a function is used to return a single value or a single object, be it an integer or a floating point number.

But what if we wanted to return multiple values from a function with the return statement.

So let’s see how we can achieve this.

**How to return multiple values using the return statement?**

Multiple return statements are not allowed in Python functions. So, the only way to return multiple values from a function is by returning [lists](http://www.trytoprogram.com/python-programming/python-list/) or [tuples](http://www.trytoprogram.com/python-programming/python-tuples/)which can have multiple integers or any other data type. This is the indirect way of returning multiple values.

Here is the example illustrating the concept of returning multiple values from a function. We will return a tuple containing multiple values.

def add(a,b,c,d):

x = a + b

y = c + d

return (x,y)

As shown in above example, we can return multiple values using tuples or lists or dictionaries. But it becomes problematic if the number values to be returned is huge.

Now that we know about function definition, function call, docstring, and the return statement.

## Advantage of functions in python

There are the following advantages of C functions.

* By using functions, we can avoid rewriting same logic/code again and again in a program.
* We can call python functions any number of times in a program and from any place in a program.
* We can track a large python program easily when it is divided into multiple functions.
* Reusability is the main achievement of python functions.
* However, Function calling is always overhead in a python program.

## Creating a function

In python, we can use **def** keyword to define the function. The syntax to define a function in python is given below.

**def** my\_function():

    function-suite

**return** <expression>

The function block is started with the colon (:) and all the same level block statements remain at the same indentation.

A function can accept any number of parameters that must be the same in the definition and function calling.

## Function calling

In python, a function must be defined before the function calling otherwise the python interpreter gives an error. Once the function is defined, we can call it from another function or the python prompt. To call the function, use the function name followed by the parentheses.

A simple function that prints the message "Hello Word" is given below.

**def** hello\_world():

**print**("hello world")

hello\_world()

**Output:**

hello world

## Parameters in function

The information into the functions can be passed as the parameters. The parameters are specified in the parentheses. We can give any number of parameters, but we have to separate them with a comma.

Consider the following example which contains a function that accepts a string as the parameter and prints it.

## Example 1

#defining the function

**def** func (name):

**print**("Hi ",name);

#calling the function

func("Ayush")

**Example 2**

#python function to calculate the sum of two variables

#defining the function

**def** sum (a,b):

**return** a+b;

#taking values from the user

a = int(input("Enter a: "))

b = int(input("Enter b: "))

#printing the sum of a and b

**print**("Sum = ",sum(a,b))

**Output:**

Enter a: 10

Enter b: 20

Sum = 30

## Call by reference in Python

In python, all the functions are called by reference, i.e., all the changes made to the reference inside the function revert back to the original value referred by the reference.

However, there is an exception in the case of mutable objects since the changes made to the mutable objects like string do not revert to the original string rather, a new string object is made, and therefore the two different objects are printed.

## Example 1 Passing Immutable Object (List)

#defining the function

**def** change\_list(list1):

    list1.append(20);

    list1.append(30);

**print**("list inside function = ",list1)

#defining the list

list1 = [10,30,40,50]

#calling the function

change\_list(list1);

**print**("list outside function = ",list1);

**Output:**

list inside function = [10, 30, 40, 50, 20, 30]

list outside function = [10, 30, 40, 50, 20, 30]

## Example 2 Passing Mutable Object (String)

#defining the function

**def** change\_string (str):

    str = str + " Hows you";

**print**("printing the string inside function :",str);

string1 = "Hi I am there"

#calling the function

change\_string(string1)

**print**("printing the string outside function :",string1)

**Output:**

printing the string inside function : Hi I am there Hows you

printing the string outside function : Hi I am there

## Types of arguments

There may be several types of arguments which can be passed at the time of function calling.

1. Required arguments
2. Keyword arguments
3. Default arguments
4. Variable-length arguments

## Required Arguments

Till now, we have learned about function calling in python. However, we can provide the arguments at the time of function calling. As far as the required arguments are concerned, these are the arguments which are required to be passed at the time of function calling with the exact match of their positions in the function call and function definition. If either of the arguments is not provided in the function call, or the position of the arguments is changed, then the python interpreter will show the error.

Consider the following example.

## Example 1

#the argument name is the required argument to the function func

**def** func(name):

    message = "Hi "+name;

**return** message;

name = input("Enter the name?")

**print**(func(name))

**Output:**

Enter the name?John

Hi John

## Example 2

#the function simple\_interest accepts three arguments and returns the simple interest accodingly

**def** simple\_interest(p,t,r):

**return** (p\*t\*r)/100

p = float(input("Enter the principle amount? "))

r = float(input("Enter the rate of interest? "))

t = float(input("Enter the time in years? "))

**print**("Simple Interest: ",simple\_interest(p,r,t))

**Output:**

Enter the principle amount? 10000

Enter the rate of interest? 5

Enter the time in years? 2

Simple Interest: 1000.0

## Example 3

#the function calculate returns the sum of two arguments a and b

**def** calculate(a,b):

**return** a+b

calculate(10) # this causes an error as we are missing a required arguments b.

**Output:**

TypeError: calculate() missing 1 required positional argument: 'b'

## Keyword arguments

Python allows us to call the function with the keyword arguments. This kind of function call will enable us to pass the arguments in the random order.

The name of the arguments is treated as the keywords and matched in the function calling and definition. If the same match is found, the values of the arguments are copied in the function definition.

Consider the following example.

## Example 1

#function func is called with the name and message as the keyword arguments

**def** func(name,message):

**print**("printing the message with",name,"and ",message)

func(name = "John",message="hello") #name and message is copied with the values Johnand hello respectively

**Output:**

printing the message with John and hello

## Example 2 providing the values in different order at the calling

#The function simple\_interest(p, t, r) is called with the keyword arguments the order of arguments doesn't matter in this case

**def** simple\_interest(p,t,r):

**return** (p\*t\*r)/100

**print**("Simple Interest: ",simple\_interest(t=10,r=10,p=1900))

**Output:**

Simple Interest: 1900.0

If we provide the different name of arguments at the time of function call, an error will be thrown.

Consider the following example.

## Example 3

#The function simple\_interest(p, t, r) is called with the keyword arguments.

**def** simple\_interest(p,t,r):

**return** (p\*t\*r)/100

**print**("Simple Interest: ",simple\_interest(time=10,rate=10,principle=1900)) # doesn't find the exact match of the name of the arguments (keywords)

**Output:**

TypeError: simple\_interest() got an unexpected keyword argument 'time'

The python allows us to provide the mix of the required arguments and keyword arguments at the time of function call. However, the required argument must not be given after the keyword argument, i.e., once the keyword argument is encountered in the function call, the following arguments must also be the keyword arguments.

Consider the following example.

## Example 4

**def** func(name1,message,name2):

**print**("printing the message with",name1,",",message,",and",name2)

func("John",message="hello",name2="David") #the first argument is not the keyword argment

**Output:**

printing the message with John , hello ,and David

The following example will cause an error due to an in-proper mix of keyword and required arguments being passed in the function call.

## Example 5

**def** func(name1,message,name2):

**print**("printing the message with",name1,",",message,",and",name2)

func("John",message="hello","David")

**Output:**

SyntaxError: positional argument follows keyword argument

## Default Arguments

Python allows us to initialize the arguments at the function definition. If the value of any of the argument is not provided at the time of function call, then that argument can be initialized with the value given in the definition even if the argument is not specified at the function call.

## Example 1

**def** printme(name,age=22):

**print**("My name is",name,"and age is",age)

printme(name = "john") #the variable age is not passed into the function however the default value of age is considered in the function

**Output:**

My name is john and age is 22

## Example 2

**def** printme(name,age=22):

**print**("My name is",name,"and age is",age)

printme(name = "john") #the variable age is not passed into the function however the default value of age is considered in the function

printme(age = 10,name="David") #the value of age is overwritten here, 10 will be printed as age

**Output:**

My name is john and age is 22

My name is David and age is 10

## Variable length Arguments

In the large projects, sometimes we may not know the number of arguments to be passed in advance. In such cases, Python provides us the flexibility to provide the comma separated values which are internally treated as tuples at the function call.

However, at the function definition, we have to define the variable with \* (star) as \*<variable - name >.

Consider the following example.

## Example

**def** printme(\*names):

**print**("type of passed argument is ",type(names))

**print**("printing the passed arguments...")

**for** name **in** names:

**print**(name)

printme("john","David","smith","nick")

**Output:**

type of passed argument is <class 'tuple'>

printing the passed arguments...

john

David

smith

nick

## Scope of variables

The scopes of the variables depend upon the location where the variable is being declared. The variable declared in one part of the program may not be accessible to the other parts.

In python, the variables are defined with the two types of scopes.

1. Global variables
2. Local variables

The variable defined outside any function is known to have a global scope whereas the variable defined inside a function is known to have a local scope.

Consider the following example.

## Example 1

**def** print\_message():

    message = "hello !! I am going to print a message." # the variable message is local to the function itself

**print**(message)

print\_message()

**print**(message) # this will cause an error since a local variable cannot be accessible here.

**Output:**

hello !! I am going to print a message.

File "/root/PycharmProjects/PythonTest/Test1.py", line 5, in

print(message)

NameError: name 'message' is not defined

## Example 2

**def** calculate(\*args):

    sum=0

**for** arg **in** args:

        sum = sum +arg

**print**("The sum is",sum)

sum=0

calculate(10,20,30) #60 will be printed as the sum

**print**("Value of sum outside the function:",sum) # 0 will be printed

**Output:**

The sum is 60

Value of sum outside the function: 0

**Scope and lifetime of the variable**

The scope of a variable determines its accessibility and availability in different portions of a program. Their availability depends on where they are defined. Similarly, life is a period in which the variable is stored in the memory.

Depending on the scope and the lifetime, there are two kinds of variables in Python.

* **Local Variables**
* **Global Variables**

Local Variables vs Global Variables

Here are some of the points to list out the difference between global and local variable for their proper understanding.

* Variables or parameters defined inside a function are called local variables as their scope is limited to the function only. On the contrary, Global variables are defined outside of the function.
* Local variables can’t be used outside the function whereas a global variable can be used throughout the program anywhere as per requirement.
* The lifetime of a local variable ends with the termination or the execution of a function, whereas the lifetime of a global variable ends with the termination of the entire program.
* The variable defined inside a function can also be made global by using the global statement.

def function\_name(args):

.............

global x #declaring global variable inside a function

## ****Docstring****

As we saw in the function definition, the first statement of the body function is a documentation string called doctsring, which explains in brief about the function and its operation. In Python, string literals are used for documenting modules.

We can access the docstring by \_\_doc\_\_ attribute. For example function\_name.\_\_doc\_\_.

**Note:**  The doc attribute has two underscores(\_\_) before and after. Using single underscore (\_) will raise an error.

Here is the example to show how we can access the docstring in Python functions.

def Hello():

""" Hello World """

print ('Hi')

print ("The doctsring of the function Hello is: "+ Hello.\_\_doc\_\_)

**Output:**

This script will generate following output.

The docstring of the function Hello is: Hello World

### What Are Namespaces In Python?

A namespace is a simple system to control the names in a program. It ensures that names are unique and won’t lead to any conflict.

Also, add to your knowledge that Python implements namespaces in the form of dictionaries. It maintains a name-to-object mapping where names act as keys and the objects as values. Multiple namespaces may have the same name but pointing to a different variable. Check out a few examples of namespaces for more clarity.

**Local Namespace**

This namespace covers the local names inside a function. Python creates this namespace for every function called in a program. It remains active until the function returns.

**Global Namespace**

This namespace covers the names from various imported modules used in a project. Python creates this namespace for every module included in your program. It’ll last until the program ends.

**Built-in Namespace**

This namespace covers the built-in functions and built-in exception names. Python creates it as the interpreter starts and keeps it until you exit.

### What Is Scope In Python?

Namespaces make our programs immune from name conflicts. However, it doesn’t give us a free ride to use a variable name anywhere we want. Python restricts names to be bound by specific rules known as a scope. The scope determines the parts of the program where you could use that name without any prefix.

* Python outlines different scopes for locals, function, modules, and built-ins. Check out from the below list.
* A local scope, also known as the innermost scope, holds the list of all local names available in the current function.
* A scope for all the enclosing functions, it finds a name from the nearest enclosing scope and goes outwards.
* A module level scope, it takes care of all the global names from the current module.
* The outermost scope which manages the list of all the built-in names. It is the last place to search for a name that you cited in the program.

### Scope Resolution In Python – Examples

Scope resolution for a given name begins from the inner-most function and then goes higher and higher until the program finds the related object. If the search ends without any outcome, then the program throws a NameError exception.

Let’s now see some examples which you can run inside any [**Python IDE**](https://www.techbeamers.com/best-python-ide-python-programming/) or with IDLE.

a\_var = 10

print("begin()-> ", dir())

def foo():

b\_var = 11

print("inside foo()-> ", dir())

foo()

print("end()-> ", dir())

The output is as follows.

begin()-> ['\_\_builtins\_\_', '\_\_doc\_\_', '\_\_file\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_spec\_\_', 'a\_var']

inside foo()-> ['b\_var']

end()-> ['\_\_builtins\_\_', '\_\_doc\_\_', '\_\_file\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_spec\_\_', 'a\_var', 'foo']

In this example, we used the dir() function. It lists all the names that are available in a Python program then.

In the first print() statement, the dir() only displays the list of names inside the current scope. While in the second print(), it finds only one name, “b\_var,” a local function variable.

Calling dir() after defining the foo() pushes it to the list of names available in the global namespace.

In the next example, we’ll see the list of names inside some nested functions. The code in this block continues from the previous block.

def outer\_foo():

outer\_var = 3

def inner\_foo():

inner\_var = 5

print(dir(), ' - names in inner\_foo')

outer\_var = 7

inner\_foo()

print(dir(), ' - names in outer\_foo')

outer\_foo()

The output is as follows.

['inner\_var'] - names in inner\_foo

['inner\_foo', 'outer\_var'] - names in outer\_foo

The above example defines two variables and a function inside the scope of outer\_foo(). Inside the inner\_foo(), the dir() function only displays one name i.e. “inner\_var”. It is alright as the “inner\_var” is the only variable defined in there.

If you reuse a global name inside a local namespace, then Python creates a new local variable with the same name.

a\_var = 5

b\_var = 7

def outer\_foo():

global a\_var

a\_var = 3

b\_var = 9

def inner\_foo():

global a\_var

a\_var = 4

b\_var = 8

print('a\_var inside inner\_foo :', a\_var)

print('b\_var inside inner\_foo :', b\_var)

inner\_foo()

print('a\_var inside outer\_foo :', a\_var)

print('b\_var inside outer\_foo :', b\_var)

outer\_foo()

print('a\_var outside all functions :', a\_var)

print('b\_var outside all functions :', b\_var)

Here goes the output of the above code after execution.

a\_var inside inner\_foo : 4

b\_var inside inner\_foo : 8

a\_var inside outer\_foo : 4

b\_var inside outer\_foo : 9

a\_var outside all functions : 4

b\_var outside all functions : 7

We’ve declared a global variable as “a\_var” inside both the outer\_foo() and inner\_foo() functions. However, we’ve assigned different values in the same global variable. And that’s the reason the value of “a\_var” is same (i.e., 4) on all occasions.

Whereas, each function is creating its own “b\_var” variable inside the local scope. And the print() function is showing the values of this variable as per its local context.

### How To Correctly Import Modules In Python?

It is very likely that you would import some of the external modules in your program. So, we’ll discuss here some of the import strategies, and you can choose the best one.

#### **Import All Names From A Module**

from <module name> import \*

It’ll import all the names from a module directly into your working namespace. Since it is an effortless way, so you might tempt to use this method. However, you may not be able to tell which module imported a particular function.

Here is an example of using this method.

print("namespace\_1: ", dir())

from math import \*

print("namespace\_2: ", dir())

print(sqrt(144.2))

from cmath import \*

print("namespace\_3: ", dir())

print(sqrt(144.2))

The output of the above code is as follows.

namespace\_1: ['\_\_builtins\_\_', '\_\_doc\_\_', '\_\_file\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_spec\_\_']

namespace\_2: ['\_\_builtins\_\_', '\_\_doc\_\_', '\_\_file\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_spec\_\_', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'trunc']

12.00833044182246

namespace\_3: ['\_\_builtins\_\_', '\_\_doc\_\_', '\_\_file\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_spec\_\_', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'phase', 'pi', 'polar', 'pow', 'radians', 'rect', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'trunc']

(12.00833044182246+0j)

In this example, we’ve imported two distinct math modules, one after the other. There are some common names which both of these modules have. So, the second module will override the definitions of functions in the first.

The first call to sqrt() returns a real number and the second one gives a complex number. And now, there is no way we can call the sqrt() function from the first math module.

Even if we call the function using the module name, then Python will raise the NameError exception. So, the lesson learned here is that there are no shortcuts for quality code.

#### **Import Specific Names From A Module**

from <module name> import <foo\_1>, <foo\_2>

If you are sure of the names to be used from a module, then import them directly into your program. This method is slightly better but will not prevent you from polluting the namespace completely. It is because you can’t use any other name from the module. Here also, any function having the same name in your program will also override the same definition present in the module. The affected method will become dormant in such a case.

Check out an example of using this method.

print("namespace\_1: ", dir())

from math import sqrt, pow

print("namespace\_2: ", dir())

print(sqrt(144.2))

The output of the above code is as follows.

namespace\_1: ['\_\_builtins\_\_', '\_\_doc\_\_', '\_\_file\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_spec\_\_']

namespace\_2: ['\_\_builtins\_\_', '\_\_doc\_\_', '\_\_file\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_spec\_\_', 'pow', 'sqrt']

12.00833044182246

#### **Import Just The Module Using Its Name**

import <module name>

It is the most reliable and suggested way of importing a module. However, it comes with a catch that you need to prefix the name of the module before using any name from it. But you can prevent the program from polluting the namespace and freely define functions with matching names in the module.

print("namespace\_1: ", dir())

import math

print("namespace\_2: ", dir())

print(math.sqrt(144.2))

The output of the above example goes like this.

namespace\_1: ['\_\_builtins\_\_', '\_\_doc\_\_', '\_\_file\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_spec\_\_']

namespace\_2: ['\_\_builtins\_\_', '\_\_doc\_\_', '\_\_file\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_spec\_\_', 'math']

12.00833044182246

## Quick Wrap Up – Python Namespace And Scope

If you want to do serious programming, then it is vital for you to know how scopes and namespaces work in Python. With this knowledge, you can even develop a scalable package ecosystem to be used by a large group working on a massive project.