

# Python

-Guido van Rossum released in 1991.

## Flavors of Python

- 1.CPython-used to work with C lanugage Applications
- 2.Jython or Jpython-for Java Applications
- 3.IronPython-for C#.Net platform
- 4.PyPy-alternative to CPython
- 5.RubyPython
- 6.AnacondaPython

## Features of Python:

1. Simple and easy to learn:
2. Freeware and Open Source
3. High Level Programming language
4. Platform Independent
5. Portability
6. Dynamically Typed
7. Both Procedure Oriented and Object Oriented
8. Interpreted
9. Extensible
10. Embedded
11. Extensive Library

## Is Python Programming Language/ Scripting Language?

Python is a programming language.  
Python can be used on a server to create web applications.  
Same as Java

Python is also Scripting Language whenever you  
Use it to automate a certain process in a specific  
environment.

Where we can use Python:

We can use everywhere. The most common important application areas are

1. For developing Desktop Applications
2. For developing web Applications
3. For developing database Applications
4. For Network Programming
5. For developing games
6. For Data Analysis Applications
7. For DataScience & Machine Learning
8. For developing Artificial Intelligence Applications
9. For IOT
10. For Scripting and Automation

# Why you use Python?

TestClass.java

## Java

```
class TestClass
{
    Public static void main(String[] args)
    {
        System.out.println("HelloWorld");
    }#No indentations Required here
}
```

Here Java is Object Oriented Programming Language

Test.c

## C

```
#include <stdio.h>
int main()
{
    //displays the string inside quotation
    printf("Hello, World!");
    return 0;
}
```

Here C is Procedure Oriented Programming Language

hello.py

## Python

```
print("HelloWorld")
```

Here Python is both Object Oriented and Procedure Oriented Programming Language and Requires Less Lines of Code

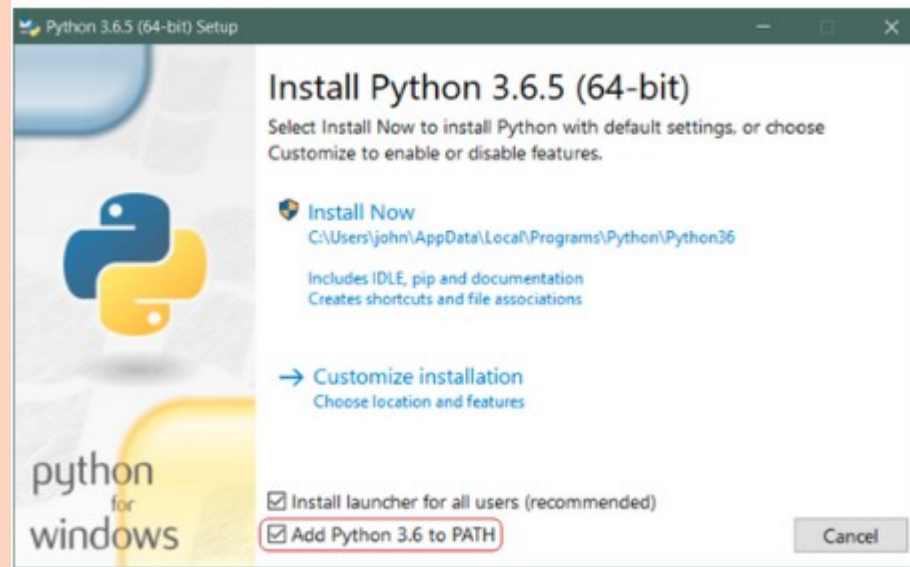
## Installation of Python in Windows

**Step 1:** Download the Python 3 Installer from below URL

<https://www.python.org/downloads/windows/>

Select Windows x86-64 executable installer for 64-bit or Windows x86 executable installer for 32-bit as per your Requirement.

**Step 2:** Run the Installer



For Installing in Different OS's visit

<https://realpython.com/installing-python/>

# Identifiers

- A name in Python program is called identifier.
- It can be class name or function name or module name or variable name.
- **Rules to define identifiers in Python:**

1).The only allowed characters in Python are  
a).alphabet symbols(either lower/upper case).  
b).digits(0 to 9).  
c).underscore symbol(\_).

2). Identifier should not starts with digit.  
Examples:

a).123total  
b).total123

4). We cannot use  
reserved words/keywords as identifiers  
Eg: def=10

3. Identifiers are case sensitive.

Ex:-       total=10  
              TOTAL=999  
              print(total) #10  
              print(TOTAL) #999

Note:

1. If identifier starts with \_ symbol then it indicates that it is private
2. If identifier starts with \_\_ (two under score symbols) indicating that strongly private identifier.
- 3.If the identifier starts and ends with two underscore symbols then the identifier is language defined special name,which is also known as magic methods.  
Eg: \_\_add\_\_

# Literals

## 1.Numeric Literals

Numeric Literals are immutable (unchangeable).

Numeric literals can belong to  
3 different numerical types Integer, Float and Complex.

Ex:-

```
#Int Literals
a = 0b1010 #Binary Literals
b = 100 #Decimal Literal
#Float Literal
float_1 = 10.5
float_2 = 1.5e2
#Complex Literal
x = 3.14j
print(a, b, float_1, float_2, x)
```

## 2.String literals

A string literal is a sequence of characters surrounded by quotes.  
We can use both single, double or triple quotes for a string.

Ex:-

```
strings = "This is Python"
char = "C"

multi_ln = """This is a multiline string with more than one line code."""
print(strings, char, multi_ln)
```

## 4.Special literals

Python contains one special literal i.e. None.  
We use it to specify to that field that is not created.

Ex:

```
val = None
```

## 3.Boolean literals

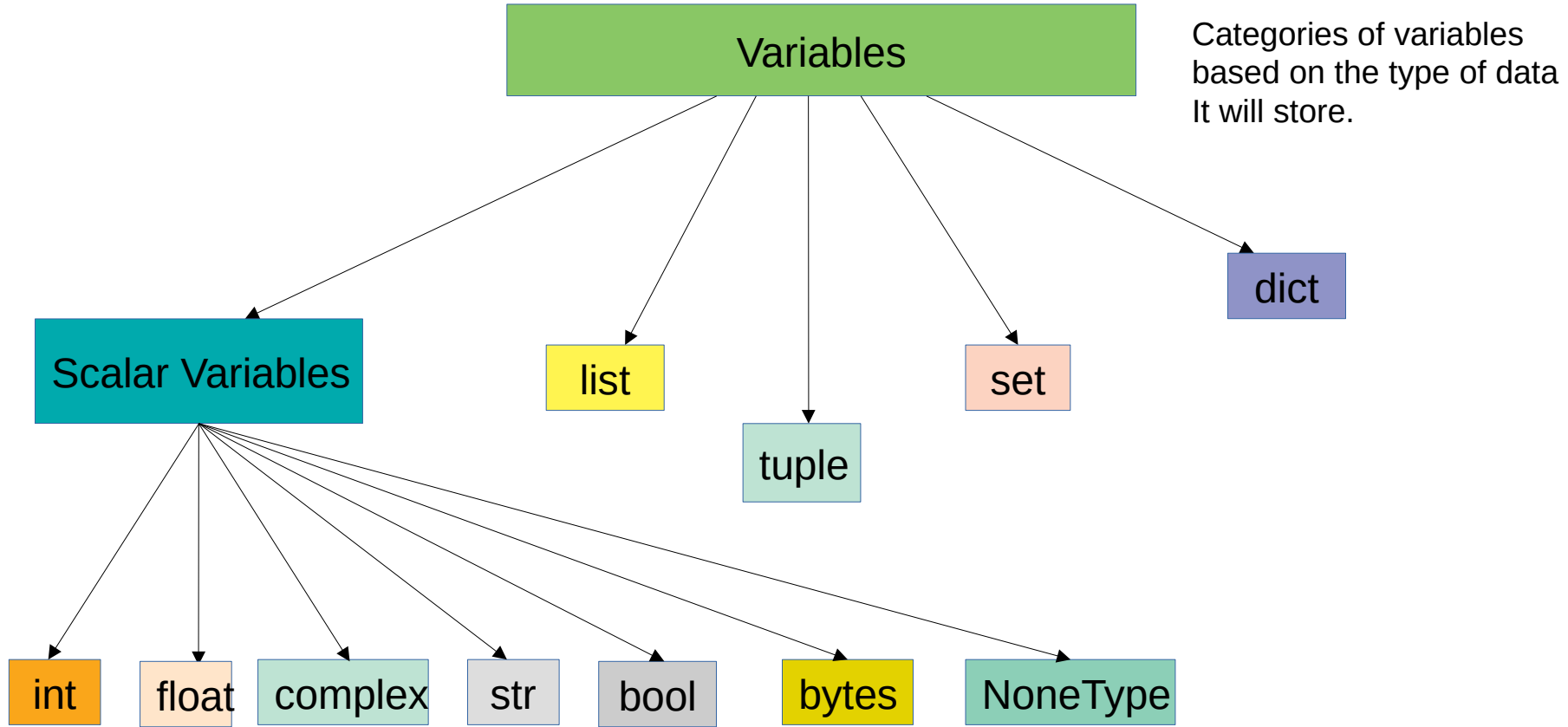
Ex:-

```
x = (1 == True)
y = (1 == False)
z=True
a = True + 4
b = False + 10
print(a, b, x, y, z)
```

## 5.Literal Collections

Syntax and Ex:

```
fruits = ["apple", "mango", "orange"] #list-[value,...]
numbers = (1, 2, 3) #tuple-(value,...)
alphabets = {'a':'apple', 'b':'ball', 'c':'cat'} #dictionary-{key:value,...}
vowels = {'a', 'e', 'i', 'o', 'u'} #set-{value,...}
print(fruits, type(fruits))
print(numbers, type(numbers))
print(alphabets, type(alphabets))
print(vowels, type(vowels))
```





# Variables

Categories of Variables based on  
Where it will declare

## Global

## Local

Variables declared outside function will be accessible(Scope)  
Anywhere in that program.  
If you declare inside a Class then that variable will be accessible  
Throughout that class not outside the class.

### Ex:

A = 10 # Global Variable

```
def my_function():  
    print(a)
```

```
my_function()
```

```
print(a)
```

Variables declared inside function/Method  
will be accessible(Scope) inside that function/Method only.

### Ex:

a = 0

```
def my_function():  
    a = 3 #local Variable  
    print(a)
```

```
my_function()
```

```
print(a)
```

## DataTypes

In Python everything is an object.

Python contains the following inbuilt data types

- 1.**int**-to represent whole numbers
- 2.**float**-to represent floating point values(decimal values)
- 3.**complex**-to represent the complex numbers
- 4.**bool**-to represent boolean values
- 5.**str**-sequence of Characters
- 6.**bytes**-To represent a sequence of byte values from 0-255
- 7.**bytearray**-To represent a sequence of byte values from 0-255
- 8.**list**-To represent an ordered collection of objects
- 9.**tuple**-To represent an ordered collections of objects
- 10.**set**-To represent an unordered collection of unique objects
- 11.**frozenset**-To represent an unordered collection of unique objects
- 12.**dict**-To represent a group of key-value pairs
- 13.**None**-Nothing or No value associated.

**int-Immutable**

```
a=10
print(type(a))#<class 'int'>
```

**float-Immutable**

```
b=10.5
print(type(b)) #<class 'float'>
```

**complex-Immutable**

```
c=10+5j
print(type(c)) #<class 'complex'>
print(c.real, c.imag) #10.0 5.0
```

**bool-Immutable**

Only allowed values are True and False  
val=True

**str-Immutable**

```
s1='EID'
s2="Ename"
s3=""Hai
Hello""
```

**bytes-Immutable**

```
list = [1, 2, 3, 4, 5]
arr = bytes(list)
print(arr, type(arr))
#b'\x01\x02\x03\x04\x05' <class 'bytes'>
```

**bytearray-Mutable**

```
list=[10,20,30]
ba=bytearray(list)
print(type(ba)) #<class 'bytearray'>
```

**list-Mutable**

```
l=[10,11,12,13,14,15]
print(type(l))
#<class 'list'>
```

**tuple-Immutable**

```
t=(10,11,12,13,14,15)
print(type(t))
#<class 'tuple'>
```

**Set-Mutable**

```
s={10,11,12,13,14,15}
print(type(s))
#<class 'set'>
```

**frozenset-Immutable**

```
s={11,2,3,'Durga',100,'Ramu'}
fs=frozenset(s)
type(fs)
#<class 'frozenset'>
```

**dict-Mutable**

```
d={101:'durga',
102:'ramu',
103:'hari'}
print(type(d))
```

```
def m1():
    a=10
    b=None
    print(m1()) #None
    print(b) #None
```

## Python Indentation

Most of the programming languages like C, C++, Java use braces { } to define a block of code. Python uses indentation.

### Note

Generally four whitespaces are used for indentation and is preferred over tabs.

```
script.py  IPython Shell
1  for i in range(1,11):
2      print(i)
3  if i == 5:
4      print(i*5)
```

Indentation can be ignored in line continuation. But it's a good idea to always indent. It makes the code more readable.  
For example:

```
if True:
    print('Hello')
    a = 5
```

Or

```
if True: print('Hello'); a = 5
```

both are valid and do the same thing. But the former style is clearer.

Incorrect indentation will result into IndentationError.

# Function

function is a group of related statements that perform a specific task.

Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable. Furthermore, it avoids repetition and makes code reusable. Functions can be defined inside a module, a class, or another function. Function defined inside a class is called a method.

## Syntax:

```
def function_name(parameters):  
    """docstring"""  
    statement(s)
```

## Ex:

```
def drive(key):  
    """This function makes the car drive with given key"""  
    print("Hello, " + key + ". Good morning!")  
    return key*5  
#An optional return statement to return a value from the function.
```

## Advantages of Functions:

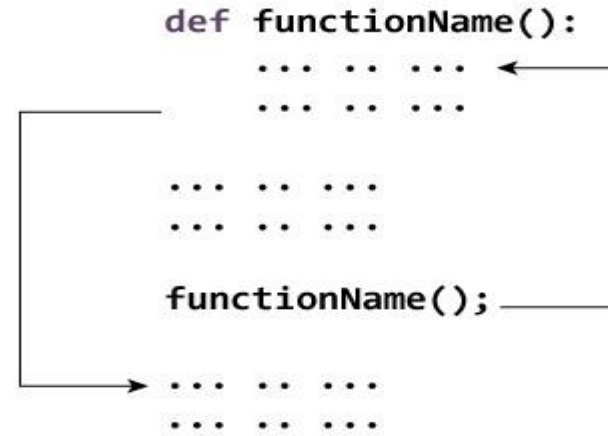
- 1).Reducing duplication of code
- 2).Decomposing complex problems into simpler pieces
- 3).Improving clarity of the code
- 4).Reuse of code
- 5).Information hiding

## Types of Functions

- 1.Built-in(Predefined)
- 2.UserDefined

## How Function works in Python?

```
def functionName():  
    ... ..  
    ... ..  
  
    ... ..  
    ... ..  
  
functionName();
```



Syntax of return  
return [expression\_list]

# Categories of Functions based on Parameters

## Categories of Functions based on Parameters

- 1.Functions without arguments
- 2.Functions with Positional arguments
- 3.Functions with Named arguments
- 4.Functions with Default arguments
- 5.Functions with Variable Length arguments

### Function Without Arguments

```
a=10
def my_function(): #Defining a Function
    print(a)
```

```
my_function() #Calling a Function
```

### Function With Positional Arguments

```
def my_function(name,age): #Defining a Function
    print(name,age)
```

```
my_function("ravi",20) #Calling a Function
my_function(20,"ravi") #Calling a Function
```

### Function With Named Arguments

```
def my_function(name,age): #Defining a Function
    print(name,age)
```

```
my_function(name="ravi",age=20) #Calling a Function
my_function(age=20,name="ravi") #Calling a Function
```

### Function With Default Arguments

```
def my_function(name="Cust",age=18): #Defining a Function
    print(name,age)
```

```
my_function("ravi",20) #Calling a Function
my_function(age=20,name="ravi") #Calling a Function
```

### Function With Variable Length Arguments

```
def my_function(*a): #Defining a Function
    for i in a:
        print(i)
```

```
my_function(10,20,30) #Calling a Function
```

## Traditional Predefined Functions

### Input():

Takes input as how you represent the data

### raw\_input():

Takes input as String

### Type():

Used to Check the type of variable  
(To check the data type)

### Print():

To print the Output in Console

### Id():

To See the HashCode/Id of an Variable/function

### del:

Used to delete a variable from Memory

Ex:

```
b=10
```

```
def myf():
```

```
    a=input("Enter Value")#takes and stores input in the format how you represent the value(literal)
```

```
    print(type(a),id(a))
```

```
    b=raw_input("Enter Data")#takes and stores input in string format
```

```
    print(type(b))
```

```
Myf()
```

```
print(b)
```

```
del b
```

## How to Run a Python Program/Script

#python filename.py

```
#!/usr/bin/env python
num1 = 15
num2 = 12

# Adding two nos
sum = num1 + num2

# printing values
print("Sum =" , sum)
```

#./filename.py

```
#!/usr/bin/env python
b=10
a=20
y='hello'
print a
print y
if a>b:
    print("a is less than b")
elif b>a:
    print("b is greater than a")
```

### Note:

If you go to Java/c, you need to use “; and {}” to tell to Compiler that this is the end of the statement.  
In Python, If you want to tell to interpreter that the line has been ended(End of the Statement)  
then you need to use “:” and/or Press Enter

## Multi-line statement

a)we can make a statement extend over multiple lines with the line continuation character (\).

Ex:

```
a = 1 + 2 + 3 + \  
    4 + 5 + 6 + \  
    7 + 8 + 9
```

b)This is explicit line continuation. In Python, line continuation is implied inside parentheses ( ),brackets [ ] and braces { }.

Ex:

```
1) a = (1 + 2 + 3 +  
        4 + 5 + 6 +  
        7 + 8 + 9)
```

c)Here, the surrounding parentheses ( ) do the line continuation implicitly. Same is the case with [ ] and { }

Ex:

```
1) colors = ['red',  
             'blue',  
             'green']
```

d)We could also put multiple statements in a single line using semicolons

Ex:

```
-->a = 1; b = 2; c = 3  
-->if True: print("Hello"); print("World")
```



## Python Comments

Comments are for programmers for better understanding of a program. Python Interpreter ignores comment.

### Single-line comments

For Single-line comments use “#” to comment a statement

Ex:

```
#this line is used to print Hello message to console.  
print('Hello')
```

### Multi-line comments

If we want to make multi-line comments,  
then we can do it in two ways

a) Using #

Ex:

```
#This is a long comment  
#and it extends  
#to multiple lines
```

b) Using triple quotes either “""" or “”

Ex:

```
"""This is also a  
perfect example of  
multi-line comments"""
```

### Docstring in Python

Docstring--->documentation string.

It is a string that occurs as the first statement in a  
module, function, class, or method definition.  
We must write what a function/class does in the docstring.

Triple quotes are used while writing docstrings.

Ex:

```
def double(num):  
    """Function to double the value"""  
    return 2*num  
double(10)  
print(double.__doc__)
```

Docstring is available to us as the attribute


“\_\_doc\_\_” of the function.

Issue the following code in shell once you run the above program.

```
>>> print(double.__doc__)  
Function to double the value
```

## DocString Example

```
public static void main(String[] args) throws Exception {  
    SpringApplication.run(CrunchifyHelloWorldSpringBoot.class, args);  
}
```

 **org.springframework.boot.SpringApplication**

*Note: This element neither has attached source nor attached Javadoc and hence no Javadoc could be found.*

mouse hover



Click here "Open Declaration" to add source

Note:

If you go to Java/c, you need to use “;” and “}” to tell to Compiler that this is the end of the statement  
If you want to tell to interpreter that the line has been ended(End of the Statement).  
Then you need to use “:” and/or Press Enter

### **Escape Characters:**

In String literals we can use escape characters to associate a special meaning  
The following are various important escape characters in Python

- 1).\n==>New Line
- 2).\t==>Horizontal tab
- 3).\r ==>Carriage Return
- 4).\b==>Back space
- 5).\f==>Form Feed
- 6).\v==>Vertical tab
- 7).\'==>Single quote
- 8).\\"==>Double quote
- 9).\\"==>back slash symbol