

Introduction To Python

➤ What is PYTHON ? :

Python is an interpreter, object-oriented, high-level programming and scripting language. It was created by “**Guido van Rossum**” and first released in 1991. Python can be used for developing desktop GUI applications, websites and web applications. Python can also be translated into binary code like java. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective.

Python programmer wastes no time declaring the types of arguments or variables, and Python's powerful polymorphic list and dictionary types, for which rich syntactic support is built straight into the language, find a use in almost every Python program. Because of the run-time typing, Python's run time must work harder than Java's. For example, when evaluating the expression **a + b**, it must first inspect the objects **a and b** to find out their type, which is not known at compile time. It then invokes the appropriate addition operation, which may be an overloaded user-defined method. Java can perform an efficient integer or floating point addition, but requires variable declarations for **a and b**, and does not allow overloading of the **+** operator for instances of user-defined classes.

One of the most promising benefits of Python are that both the standard library and the interpreter are available free of charge, in both binary and source form. There is no exclusivity either, as Python and all the necessary tools are available on all major platforms. Therefore, it is an enticing option for developers who don't want to worry about paying high development costs.

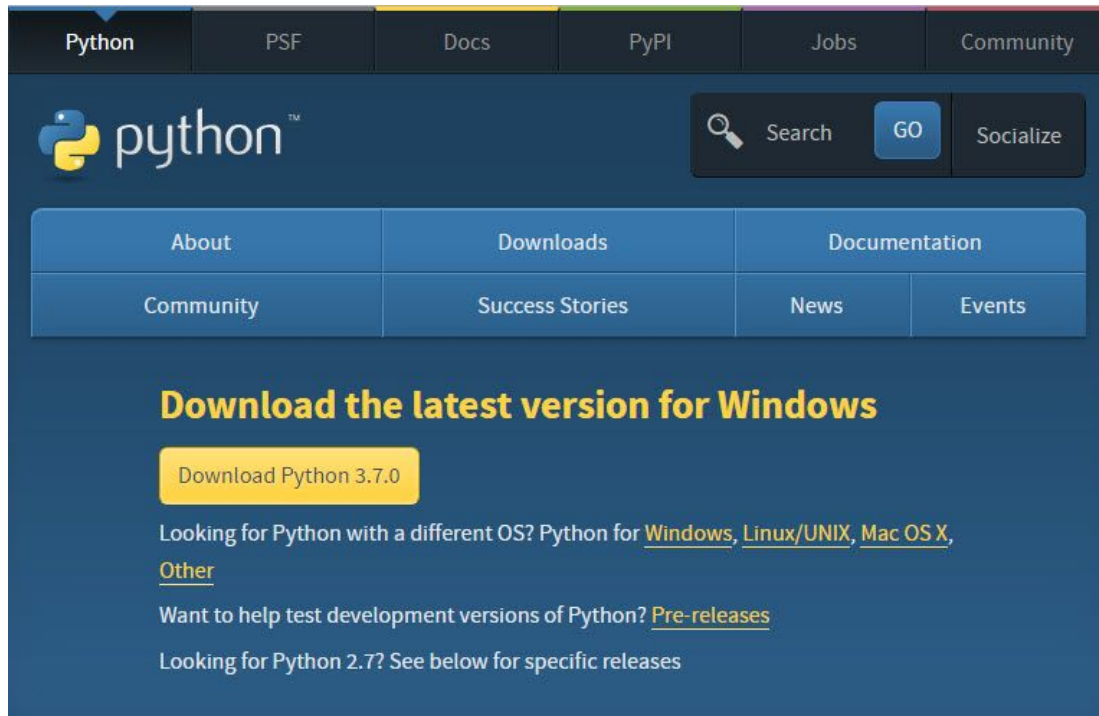
➤ PYTHON Installation on Window OS :

The Python download requires about 25 Mb of disk space; keep it on your machine, in case you need to re-install Python. When installed, Python requires about an additional 90 Mb of disk space.

Downloading Python: Version 3.7.0


1. Click **Python Download** (<https://www.python.org/downloads/>) :

The following page will appear in your browser.



2. Click the **Download Python 3.7.0** button.

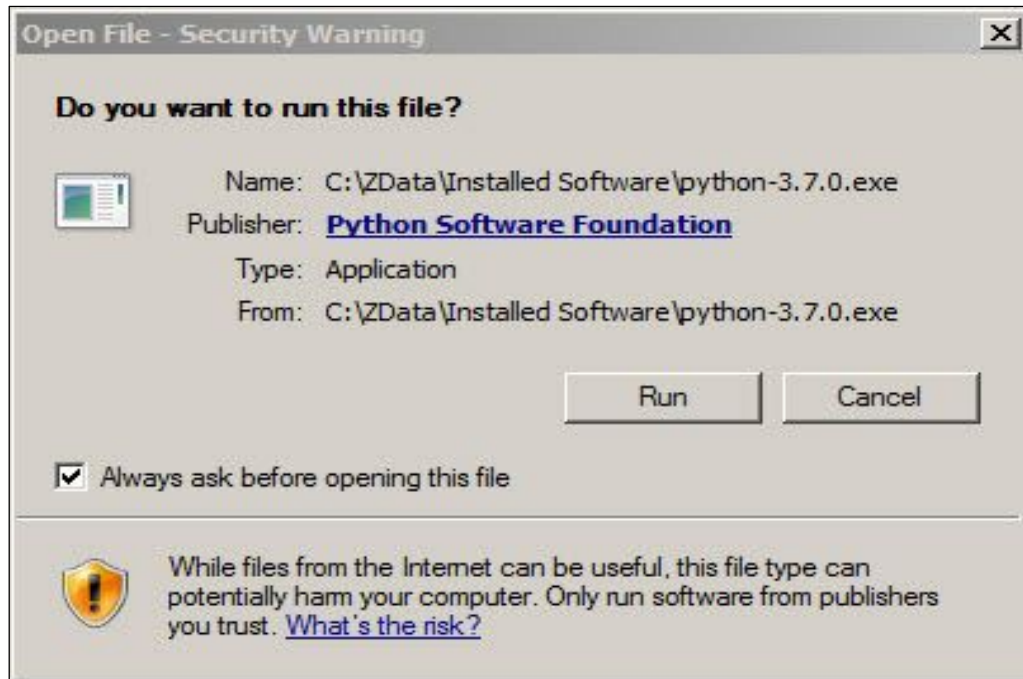
The file named **python-3.7.0.exe** should start downloading into your standard download folder. This file is about 30 Mb so it might take a while to download fully if you are on a slow internet connection (it took me about 10 seconds over a cable modem).

The file should appear as :  python-3.6.2.exe

3. Move this file to a more permanent location, so that you can install Python (and reinstall it easily later, if necessary).
4. Feel free to explore this webpage further; if you want to just continue the installation, you can terminate the tab browsing this webpage.
5. Start the **Installing** instructions directly below.

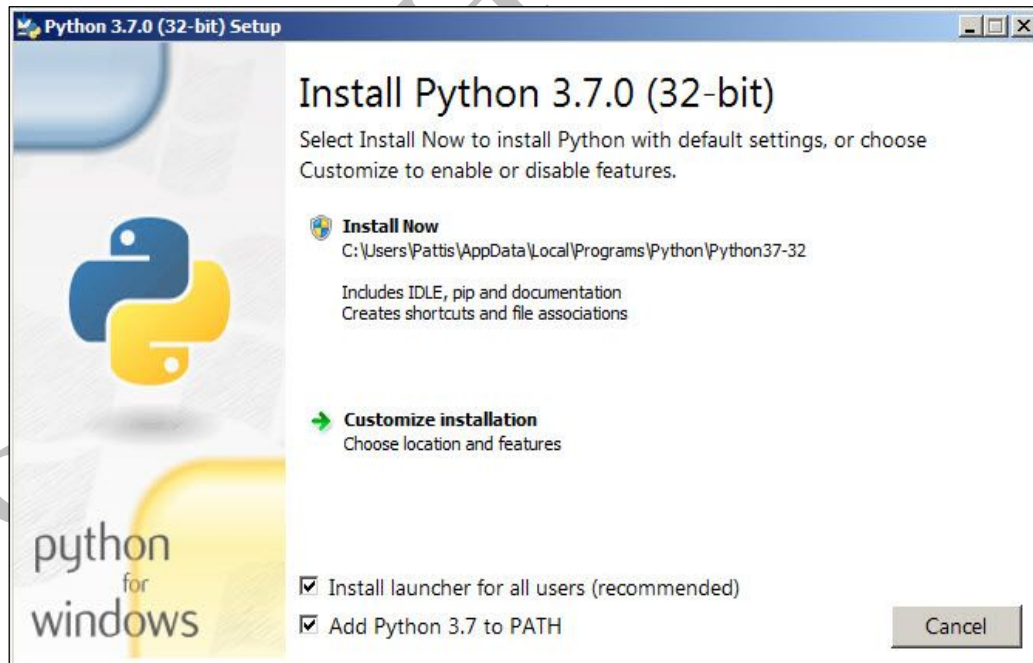
Installing :

1. Double-click the icon labeling the file **python-3.7.0.exe**.
An **Open File - Security Warning** pop-up window will appear.



2. Click **Run**.

A **Python 3.7.0 (32-bit) Setup** pop-up window will appear.



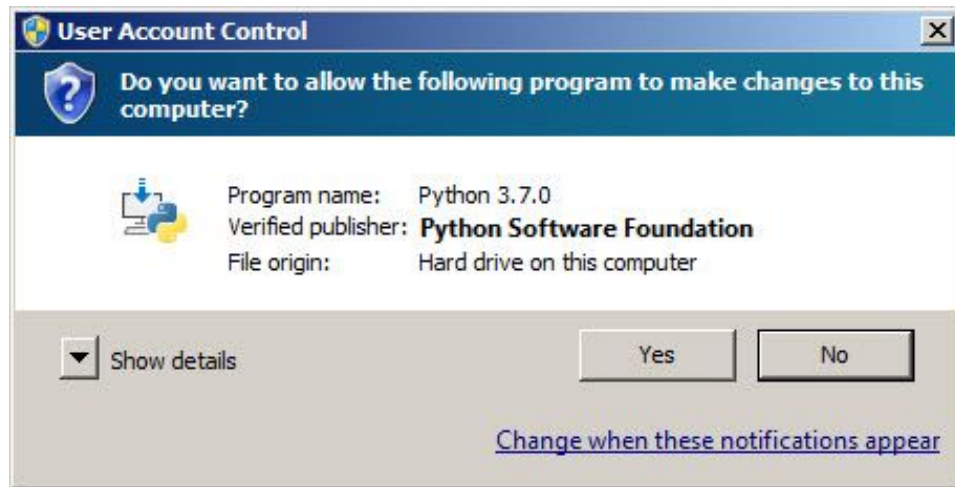
Ensure that the **Install launcher for all users (recommended)** and the **Add Python 3.7 to PATH** checkboxes at the bottom are checked.

If the Python Installer finds an earlier version of Python installed on your computer, the **Install Now** message may instead appear as **Upgrade Now** (and the checkboxes will not appear).

3. Highlight the **Install Now** (or **Upgrade Now**) message, and then click it.

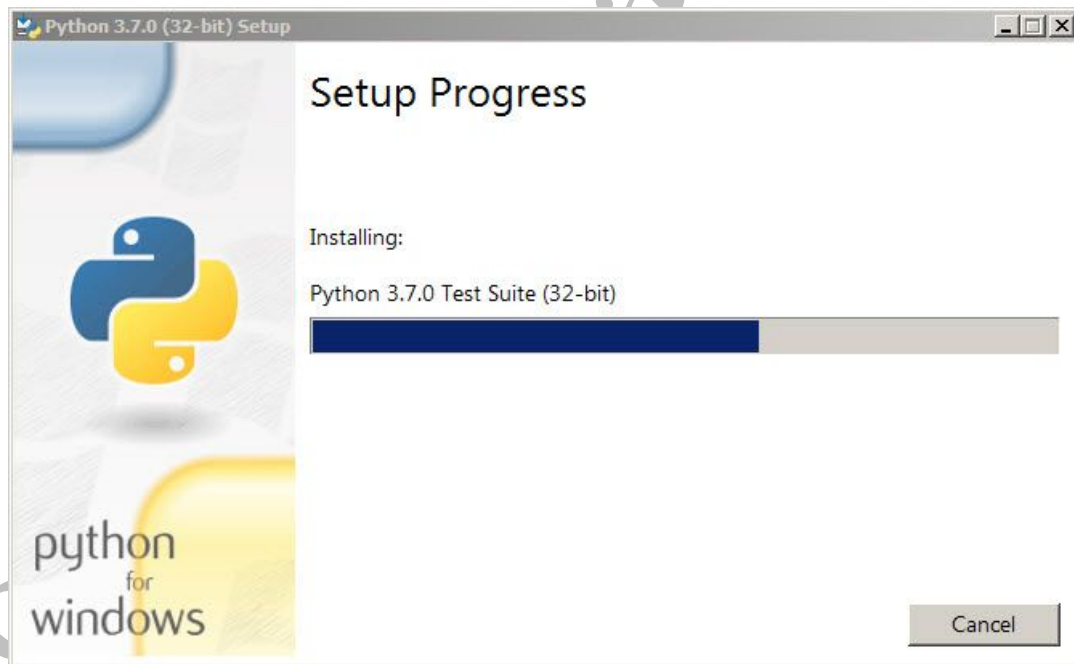
A **User Account Control** pop-up window will appear, posing the question

“Do you want to allow the following program to make changes to this computer?”



4. Click the **Yes** button.

A new **Python 3.7.0 (32-bit) Setup** pop-up window will appear with a **Setup Progress** message and a progress bar.



During installation, it will show the various components it is installing and move the progress bar towards completion. Soon, a new **Python 3.7.0 (32-bit) Setup** pop-up window will appear with a **Setup was successfully** message.



5. Click the **Close** button.

Python should now be installed.

✓ Verifying

To try to verify installation,

1. Navigate to the directory :

C:\Users\Pattis\AppData\Local\Programs\Python\Python37-32 (or to whatever directory Python was installed: see the pop-up window for Installing step 3).

2. Double-click the icon/file **python.exe**.

The following pop-up window will appear.

A screenshot of a command prompt window. The title bar shows the path "C:\Users\Pattis\AppData\Local\Programs\Python\Python37-32\python.exe". The window content shows the Python 3.7.0 shell prompt: "Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32". Below this, it says "Type 'help', 'copyright', 'credits' or 'license' for more information." and the prompt ">>>" is visible.

A pop-up window with the title :

C:\Users\Pattis\AppData\Local\Programs\Python\Python37-32 appears, and inside the window; on the first line is the text **Python 3.7.0 ...** (notice that it should also say 32 bit). Inside the window, at the bottom left, is the prompt **>>>**: type **exit()** to this prompt and press **enter** to terminate Python.

Note: you need to download/install Java even if you are using Eclipse only for Python.

➤ The basic elements of Python :

- Python programs can be written using any text editor and should have the extension **.py**.
- Python programs do not have a required first or last line, but can be given the location of python as their first line: `#!/usr/bin/python` and become executable.
- Otherwise, python programs can be run from a command prompt by typing `python file.py`.
- **There are no braces {} or semicolons ; in python.**
- It is a very high level language. Instead of braces, blocks are identified by having the same indentation.

❖ Data Types and Variable :

There are 5 (Five) standard data types in PYTHON.

1. Number 2. String 3. List 4. Tuple 5. Dictionary

1. Number : Number data types are used to store numeric values. There are **four** different numerical types in Python:

- int (plain integers):** Plain integers are just positive or negative whole numbers.
- long (long integers):** Long integers are integers of infinite size. They look just like plain integers except they're followed by the letter "L" (ex: 150L).
- float (floating point real values):** Floats represent real numbers, but are written with decimal points (or scientific notation) to divide the whole number into fractional parts.
- complex (complex numbers):** Complex numbers are specified as `<real part>+<imaginary part>j`

Example	Out Put
<pre>i = 100 #An Integer Value j = 50.5 #A Float Value print(i + j) print(5 + 3j) print(5j + 3j)</pre>	<pre>150.5 5 + 3j 8j</pre>

2. String : Strings are sequences of character data. The string type in Python is called **str**. String literals may be delimited using either single or double quotes. All the characters between the opening delimiter and matching closing delimiter are part of the string.

Example	Out Put
<pre>>>>print("Hello Students") >>>type("I am a string.")</pre>	<pre>Hello Students <class 'str'></pre>

3. List : List can contain a series of values. List variables are declared by using brackets `[]` following the variable name. All lists in Python are zero-based indexed. When

referencing a member or the length of a list the number of list elements is always the number shown plus one.

Example
<pre>A = [] # This is a blank list variable B = [1, 23, 45, 67] # this list creates an initial list of 4 numbers. C = [2, 4, 'john'] # lists can contain different variable types.</pre>

4. **Tuple** : Tuples are a group of values like a list and are manipulated in similar ways. But, tuples are fixed in size once they are assigned. In Python the fixed size is considered immutable as compared to a list that is dynamic and mutable. Tuples are defined by parenthesis ().

Advantages of Tuples over lists:

- Elements to a tuple. Tuples have no append or extend method.
- Elements cannot be removed from a tuple.
- You can find elements in a tuple, since this doesn't change the tuple.
- You can also use the in operator to check if an element exists in the tuple.
- Tuples are faster than lists. If you're defining a constant set of values and all you're ever going to do with it is iterating through it, use a tuple instead of a list.

Example	Out Put
<pre>myTuple = ('One', 'Two', 'Three', 'Four') print(myTuple[0]) print(myTuple[2])</pre>	One Three

5. **Dictionary** : Dictionaries in Python are lists of Key:Value pairs. This is a very powerful datatype to hold a lot of related information that can be associated through keys. The main operation of a dictionary is to extract a value based on the key name. Unlike lists, where index numbers are used, dictionaries allow the use of a key to access its members. Dictionaries can also be used to sort, iterate and compare data. Dictionaries are created by using **braces** ({ }) with pairs separated by a **comma** (,) and the key values associated with a **colon** (:). In Dictionaries the Key must be unique.

Example
<pre>my_Dict = {'Bhavans': 111, 'HJD': 222} # set the value associated with the 'BCA' key to 4 my_Dict['BCA'] = 4 print (my_Dict ['Bhavans']) # print the value of the ' Bhavans ' key. print (my_Dict()) # print out a list of keys in the dictionary print ('BCA' in my_Dict) # test to see if 'BCA' is in the</pre>

```
dictionary. This returns true.
```

- **Boolean** is also a data type in Python : Boolean values are the two constant objects **"False"** and **"True"**. They are used to represent truth values (other values can also be considered false or true). In numeric contexts (for example, when used as the argument to an arithmetic operator), they behave like the **integers 0 and 1**, respectively. The built-in function `bool()` can be used to cast any value to a Boolean, if the value can be interpreted as a truth value. They are written as `False` and `True`, respectively.

Example	Out Put
<pre>x = bool(0) y = bool(1) print(x) print(y)</pre>	<pre>False True</pre>

➤ Branching Statements :

Branching statements are used to change the normal flow of execution based on some condition. The return statement is used to explicitly return from a method. It is also known as Conditional statements or Decision Making Statements. Decision structures evaluate multiple expressions which produce `TRUE` or `FALSE` as outcome. Python programming language assumes any **non-zero** and **non-null** values as **TRUE**, and if it is either **zero** or **null**, then it is assumed as **FALSE** value. Conditional Statement in Python performs different computations or actions depending on whether a specific Boolean constraint evaluates to true or false. Conditional statements are handled by IF statements in Python.

1. **IF Statement** : It is used for decision making. It will run the body of code only when IF statement is true. **IF statements enclosed with COLON (:)**

SYNTAX :

```
if expression :
    Statement
```

Example	Out Put
<pre>print("This is Example of IF Statement") x = "BCA" y = 6 if(x == "BCA") : print("Hello Students") if(y == 6) : print ("Hello 6th SEM Students")</pre>	<pre>This is Example of IF Statement Hello BCA Students Hello 6th sem Students</pre>

2. **ELSE Statement** : An **else** statement can be combined with an **if** statement. An **else** statement contains the block of code that executes if the conditional expression in the **if** statement resolves to 0 or a FALSE value. The **else** statement is an optional statement and there could be at most only one else statement following **if**.

SYNTAX:

```

if expression :
    Statement
else :
    Statement

```

Example	Out Put
<pre> print("This is Example of IF - ELSE Statement") x = "BCA" if(x == "BCA") : print("\nHello BCA Students") else : print ("\nHello Folks") </pre>	<pre> This is Example of IF - ELSE Statement Hello BCA Students </pre>

3. **ELIF Statement** : The **elif** statement allows you to check multiple expressions for **TRUE** and execute a block of code as soon as one of the conditions evaluates to **TRUE**. Similar to the **else**, the **elif** statement is optional. However, unlike **else**, for which there can be at most one statement, there can be an arbitrary number of **elif** statements following an **if**.

SYNTAX:

```

if expression :
    Statement
elif expression :
    Statement
else :
    Statement

```

Example	Out Put
<pre> x = 2 if x == 1: print ("First value") print ("x = ",x) elif x == 2: print ("Second value") print ("x = ",x) elif x == 3: print ("Third value") print ("x = ",x) else: print ("False expression value") print ("x = ",x) print ("End of IF - ELIF - ELSE Statements! ") </pre>	<pre> Second Value End of IF - ELIF - ELSE Statements! </pre>

4. SWITCH Statement : A **SWITCH** statement is a multiway branch statement that compares the value of a variable to the values specified in case statements. Python language doesn't have a **SWITCH** statement. Python uses **dictionary mapping** to implement **SWITCH** statement in Python

Example
<pre>#myDict is Dictionary myDict = { 1:"Sunday", 2:"Monday", 3:"Tuesday", 4:"Wednesday", 5:"Thursday", 6:"Friday", 7:"Saturday" } i = input("Enter WeekDay Number : ") print(i, " Day is : ",myDict.get(int(i),"Invalid Day"))</pre>
Out Put
<pre>Enter WeekDay Number : 3 3 day is : TuesDay</pre>

➤ Iteration Statements :

Iteration(**LOOP**) is a general term for taking each item of something, one after another. Any time you use a loop, explicit or implicit, to go over a group of items, that is iteration. In Python, iterable and iterator have specific meanings.

There are Two Types of Loops in Python :

1. **FOR Loop**
2. **WHILE Loop**

1. **FOR Loop :** Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable. For loop is used to iterate over elements of a sequence. It is often used when you have a piece of code which you want to repeat "n" number of time.

SYNTAX :

```
for iterating_var in sequence:
    statements(s)
```

Example	Out Put
<pre>for x in range(1,5): print (x)</pre>	<pre>1 2 3 4 5</pre>

2. **WHILE** Loop : While Loop is used to repeat a block of code. Instead of running the code block once, it executes the code block multiple times until a certain condition is met.

SYNTAX :

```
while expression
    Statement
```

Example	Out Put
<pre>x = 1 while (x <= 5) : print(x) x = x + 1</pre>	1 2 3 4 5

➤ **Functions and Scoping :**

- **FUNCTION** : A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

How to Define a Function? :

- ✓ Function blocks begin with the keyword **def** followed by the function name and parentheses (()).
- ✓ Any input parameters or arguments should be placed within these parentheses.
- ✓ The first statement of a function can be an optional statement - the documentation string of the function or **docstring**.
- ✓ The code block within every function starts with a **colon " : "** and is indented.
- ✓ The statement **return [expression]** exits a function, optionally passing back an expression to the caller. A return statement with **no arguments** is the same as return **None**.

Syntax :

```
def functionname( parameters ) :
    Statements / "function_docstring"
    return [expression]
```

Example	Out Put
<pre>#Define a function def printMe(str) : #This prints a passed string into this function print str return #Calling function printMe("Hello Students")</pre>	Hello Students

- **SCOPE** : Variables can only reach the area in which they are defined, which is called scope. Think of it as the area of code where variables can be used. Python supports global variables (usable in the entire program) and local variables. By default, all variables declared in a function are **local variables**. To access a global variable inside a function, it's required to **explicitly** define '**global variable**'.
- In Python, a variable declared outside of the function or in global scope is known as global variable. This means, global variable can be accessed inside or outside of the function.
- A variable declared inside the function's body or in the local scope is known as local variable.

Example	Out Put
<pre> myStr = "This is Global Variable" def myVar(): myStr = "This is Local Variable" print (myStr) myVar() print (myStr) </pre>	<pre> This is Local Variable This is Global Variable </pre>

Example	Out Put
<pre> a = 0 # Uses global because there is no local 'a' def f(): print ("Inside f():", a) # Variable 'a' is redefined as a local def g(): a = 2 print ("Inside g():",a) # Uses global keyword to modify global 'a' def h(): global a a = 3 print ("Inside h():",a) # Global scope print ("\nglobal : ",a) f() print ("\nglobal : ",a) g() print ("\nglobal : ",a) h() print ("\nglobal : ",a) </pre>	<pre> global : 0 Inside f(): 0 global : 0 Inside g(): 2 global : 0 Inside h(): 3 global : 3 </pre>

➤ Recursion :

Recursion means that the function will continue to call itself and repeat its behavior until some condition is met to return a result. All recursive functions share a common structure made up of two parts: **base case** and **recursive case**.

Example	Out Put
<pre>def factorial_recursive(n): # Base case: 1! = 1 if n == 1: print (n,end = ' = ') return 1 # Recursive case: n! = n * (n-1)! else: print (n, end = ' X ') return n * factorial_recursive(n-1) i = input("Enter Any Number : ") print (factorial_recursive(int(i)))</pre>	<pre>Enter Any Number : 3 3 x 2 x 1 = 6 Enter Any Number : 5 5 x 4 x 3 x 2 x 1 = 120</pre>

➤ Python Modules :

Modules refer to a file containing Python statements and definitions. A file containing Python code, for e.g.: **"HiralPandya.py"**, is called a module and its module name would be **"HiralPandya"**. Python uses modules to break down large programs into small manageable and organized files. Furthermore, modules provide reusability of code. Programmers can define most used functions in a module and import it, instead of copying definitions into different programs.

A module allows to logically organize Python code. Grouping related code into a module makes the code easier to understand and use. A module is a Python object with arbitrarily named attributes that can be bound and referenced. Simply, a module is a file consisting of Python code. A module can define functions, classes and variables. A module can also include runnable code.

Let's First Create a Module named **"myAddMod.py"** on path

"C:\Python\Python3.7\Lib\" (it's a path where **"Lib"** folder is installed.)

```
def myAdd(fVal, sVal):  
    return (fVal + sVal)
```

Now, Let's use our First Module :

Example	Out Put
<pre>import myAddMod print (myAddMod.myAdd(5,6))</pre>	11

➤ Python Files / Files Handling :

File is a named location on disk to store related information. It is used to permanently store data in a non-volatile memory (e.g. hard disk). Random access memory (RAM) is volatile which loses its data when computer is turned off, we use files for future use of the data. When we want to read from or write to a file we need to open it first. When we are done, it needs to be closed, so that resources that are tied with the file are freed.

How to open a file? :

Python has a built-in function `open()` to open a file. This function returns a file object, also called a handle, as it is used to read or modify the file accordingly.

```
>>> f = open("test.txt") # open file in current directory
>>> f = open("C:\\Python\\README.txt") # specifying full path
```

We can specify the mode while opening a file. In mode, we specify whether we want to read 'r', write 'w' or append 'a' to the file. We also specify if we want to open the file in text mode or binary mode.

The default is reading in text mode. In this mode, we get strings when reading from the file. On the other hand, binary mode returns bytes and this is the mode to be used when dealing with non-text files like image or exe files.

Python File Modes

Mode	Description
'r'	Open a file for reading. (default)
'w'	Open a file for writing. Creates a new file if it does not exist or truncates the file if it exists.
'x'	Open a file for exclusive creation. If the file already exists, the operation fails.
'a'	Open for appending at the end of the file without truncating it. Creates a new file if it does not exist.
't'	Open in text mode. (default)
'b'	Open in binary mode.
'+'	Open a file for updating (reading and writing)

Example :

```
with open("C:\\Python\\test.txt", 'w', encoding = 'utf-8') as f:
    f.write("Hello\n")
    f.write("HJD\n")
    f.write("Students\n")
f.close()
```

➤ Lists and Mutability :

Mutable means Changeable. Everything in Python is an object. You have to understand that Python represents all its data as objects. An object's mutability is determined by its type. Some of these objects like lists and dictionaries are mutable, meaning you can change their content without changing their identity. Other objects like **integers, floats, strings** and **tuples** are objects that **cannot** be changed. Strings are immutable in Python, which means you cannot change an existing string.

Example	Out Put
<pre>myList = [10, 20, 30, 40, 50] print (myList) myList[0] = 5 print (myList)</pre>	<pre>[10,20,30,40,50] [5,20,30,40,50]</pre> <p>First Element changed from 10 to 5</p>
<pre>myTuple = (10, 20, 30, 40, 50) print (myTuple) myTuple [0] = 5 print (myTuple)</pre>	<pre>(10, 20, 30, 40, 50)</pre> <p>Traceback (most recent call last): File "C:\Test.py", line 3, in <module> myTuple[0] = 5 TypeError: 'tuple' object does not support item assignment</p>

➤ Functions as Objects :

First class objects in a language are handled uniformly throughout. They may be stored in data structures, passed as arguments, or used in control structures. A programming language is said to support first-class functions if it treats functions as first-class objects. Python supports the concept of First Class functions.

One interesting consequence of the Python world-view is that a function is an object of the class **function**, a subclass of **callable**. The common feature that all **callables** share is that they have a very simple interface: they can be called. Other **callables** include the built-in functions and generator functions.

Properties of first class functions:

- A function is an instance of the Object type.
- You can store the function in a variable.
- You can pass the function as a parameter to another function.
- You can return the function from a function.
- You can store them in data structures such as hash tables, lists.

Example	Out Put
<pre>def myFunc(text): return text.upper() print (myFunc("hiral pandya") myFuncObj = myFunc print (myFuncObj("python"))</pre>	<p>HIRAL PANDYA</p> <p>PYTHON</p>

Functions can be passed as arguments to other functions: Because functions are objects we can pass them as arguments to other functions. Functions that can accept other functions as arguments are also called **higher-order functions**.

Example
<pre>def myUpper(text): return text.upper() def myLower(text): return text.lower() def myHOF(func): # storing the function in a variable myVar = func("Hi, This is created by a function passed as an argument.") print (myVar) myHOF(myUpper) myHOF(myLower)</pre>
OutPut
<p>HI, THIS IS CREATED BY A FUNCTION PASSED AS AN ARGUMENT.</p> <p>hi, this is created by a function passed as an argument.</p>