Scope and Lifetime of variables
All variables in a program may not be accessible at all locations in that program. This depends on where you have declared a variable.

Local Variables
1 The variables defined within the function has a local scope and hence they are called local variables
2 Local scope means they can be accessed within the function only

Global variables
1 The variables defined outside the function has a global scope and hence they are called global variables
2 Global scope means they can be accessed within the function as well as outside the function
3 The value of a global variable can be used by referring the variable as global inside a function

Example:
```
total = 0; #Global Variable
def sum( arg1, arg2 ):
    total = arg1 + arg2; # Here total is local variable.
    print("Inside the function local total : ", total)
    return total;
sum( 10, 40 );
print("Outside the function global total : ", total)
```

Example:
```
x=40#Global Scope
print(id(x))
def Scope_Variables1():
    x=200#Local Scope
    print(id(x))
    print(x+x)#Local+Local
    return
Scope_Variables1()
def Scope_Variables2():
    z=600#Local SCope
    print(id(z))
    print(x+z)#Global+Local
    return
Scope_Variables2()
```

global
It is used to declare that a variable inside the function is global (outside the function). If we need to read the value of a global variable, it is not necessary to define it as global. If we need to modify the value of a global variable inside a function, then we must declare it with global.

Example:
```
x=100
def MyFun():
    global x
    x=5
    y=20
```

```
    print(x+y)
def MyFun1():
    y=10
    print(x+y)
MyFun()
MyFun1()
```

Example:Output of the following Script..!!

```
x=1
def Compute():
    global x
    for i in (1,2,3):
        x+=1
Compute()
print(x)
```

Example: Output of the following Script..!!

```
def f():pass
print(type(f()))
```

Function Aliasing:
For the existing function we can give other name, which is nothing but function aliasing.

Example:

```
def Hello_World(name):
    print("Great:",name)
MyWorld=Hello_World
Hello_World('Machine Learning')
MyWorld("PYTHON")
print(id(MyWorld))
print(id(Hello_World))
```

NOTE:
If we delete one name still we can access that function by using Alias Name

Example:

```
def Hello_World(name):
    print("Good Morning:",name)
MyWorld=Hello_World
Hello_World('Machine Learning')
del Hello_World
MyWorld("PYTHON")
```

NOTE:
Fun_One = Outer _Fun ==> It is Function Aliasing.
Fun_One = Outer_Fun() ==> It is calling Outer_Fun() Function

Function is an object:
In Python every thing is an object. Functions are also objects.

Example:

```
def MyFun():
    print("Say Hey PYTHON")
print(MyFun)
print(id(MyFun))
```

Nested Functions or Inner Functions:
Functions can be defined within the scope of another function.

Example:

```
def Outer(): #outer function
    print ("Hello Welcome TO")
    def Inner(): #inner function
        print ("Nested Functions")
    Inner()
Outer()
```

Example:

```
def Outer_Fun():
    print("Say Hey Outer Function")
def Inner_Fun():
    print("Bye Hey Inner Function")
print("OuterOneCallingInnerFunction")
Inner_Fun()
Outer_Fun()
```

Example:

```
def Outer_Fun():
    print("Hey Outer")
    def inner():
        print("Bye Inner")
    print("Outer Returns Inner")
    return inner
Outer_Fun()
```

Example:

```
def Outer_Fun():
    print("Hey Outer")
    def inner():
        print("Bye Inner")
    print("Outer Returns Inner")
    return inner
Inn=Outer_Fun()
Inn()
```

Example:

```
def Outer(): #outer function
    x = 1 #variable defined in Outer function
    def Inner(a): #inner function
        print (a+x) #able to acces the variable of outer function
    Inner(2)
Outer()
```

Example:

```
def Outer(): #outer function
    x = 1 # variable defined in the outer function
    def Inner(a): #inner function
       #will create a new variable in the inner function
       #instead of changing the value of x of outer function
        x = 4
        print (a+x)
    print (x) # prints the value of x of outer function
```

```
    Inner(2)
Outer()
```

nonlocal
It is used to declare that a variable inside a nested function is not local to it. If we need to modify the value of a non-local variable inside a nested function, then we must declare it with nonlocal.

Example:
```
def Outer_Function():
    a = 5
    def Inner_Function():
        nonlocal a
        a = 10
        print("Inner Function: ",a)
    Inner_Function()
    print("Outer Function: ",a)
Outer_Function()
```

Example:
```
def Outer_Function():
    a = 5
    def Inner_Function():
        a = 10
        print("Inner Function: ",a)
    Inner_Function()
    print("Outer Function: ",a)
Outer_Function()
```

PYTHON Recursive Functions:
A function calls itself one or more times in its body.
OR
If a function calls itself is called as recursive function

Example:
```
def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n-1)
print(factorial(5))
```

Example:
```
def counter(c):
    if c<=0:h
        return c
    else:
        return c + counter(c-1)
print(counter(5))
```

Example:Output of the following Script
```
def MyFunc(*args):
    for i in args:
        print(i*2,end=" ")
input_num=[1,2,3,4,5]
MyFunc(*input_num)
```

Example:Output of the following Script

```
def MyFunc(*args):
    PyList=[]
    for i in args:
        PyList.append(i*2)
    return PyList
input_num=[1,2,3,4,5]
print(MyFunc(*input_num))
```

Example:

```
# Sorting the elements of the list using a certain function
def func(x):
    return len(x)
Cars = ["Ford", "BMW", "Ferari", "Mitsubishi", "Audi"]
Cars.sort(reverse = True, key = func)
print(Cars)
# prints the elements of the list in the descreasing order of
character length
```