Variable Scope:-

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

The scope of a variable determines the portion of the program where you can access a particular identifier. There are two basic scopes of variables in Python −

- Global variables
- Local variables

Global vs. Local variables

Variables that are defined inside a function body have a local scope, and those defined outside have a global scope.

This means that local variables can be accessed only inside the function in which they are declared, whereas global variables can be accessed throughout the program body by all functions. When you call a function, the variables declared inside it are brought into scope.

```
Ex.

 total = 0; # This is global variable.

# Function definition is here

def sum( arg1, arg2 ):

   # Add both the parameters and return them."

   total = arg1 + arg2; # Here total is local variable.

   print "Inside the function local total: ", total

   return total;

# Now you can call sum function

sum(10, 20);

print "Outside the function global total : ", total
```

Output

When the above code is executed, it produces the following result −

```
Inside the function local total : 30
Outside the function global total : 0
```

Void Function returns None.

In Python, void functions are slightly different than functions found in C, C++ or Java. If the function body doesn't have any return statement then a special value None is returned when the function terminates. In Python, None is a literal of type NoneType which used to denote absence of a value. It is commonly assigned to a variable to indicate that the variable does not points to any object.

The following program demonstrates that the void functions return None.

```
def add(num1, num2):
  print("Sum is", num1 + num2)

return_val = add(100, 200)
print(return_val)
```

Output:

Sum is 300
None

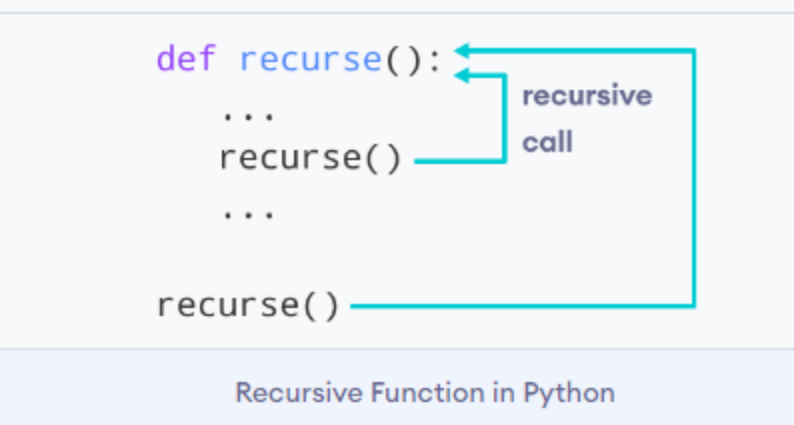
What is recursion?

Recursion is the process of defining something in terms of itself.

Python Recursive Function

In Python, we know that a function can call other functions. It is even possible for the function to call itself. These types of construct are termed as recursive functions.

The following image shows the working of a recursive function called recurse.



Recursive Function in Python

Example

```
def factorial(x):
    """This is a recursive function
    to find the factorial of an integer"""

    if x == 1:
        return 1
    else:
        return (x * factorial(x-1))


num = 3
print ("The factorial of", num, "is", factorial(num))
```

Output

```
The factorial of 3 is 6
```

In the above example, factorial () is a recursive function as it calls itself.

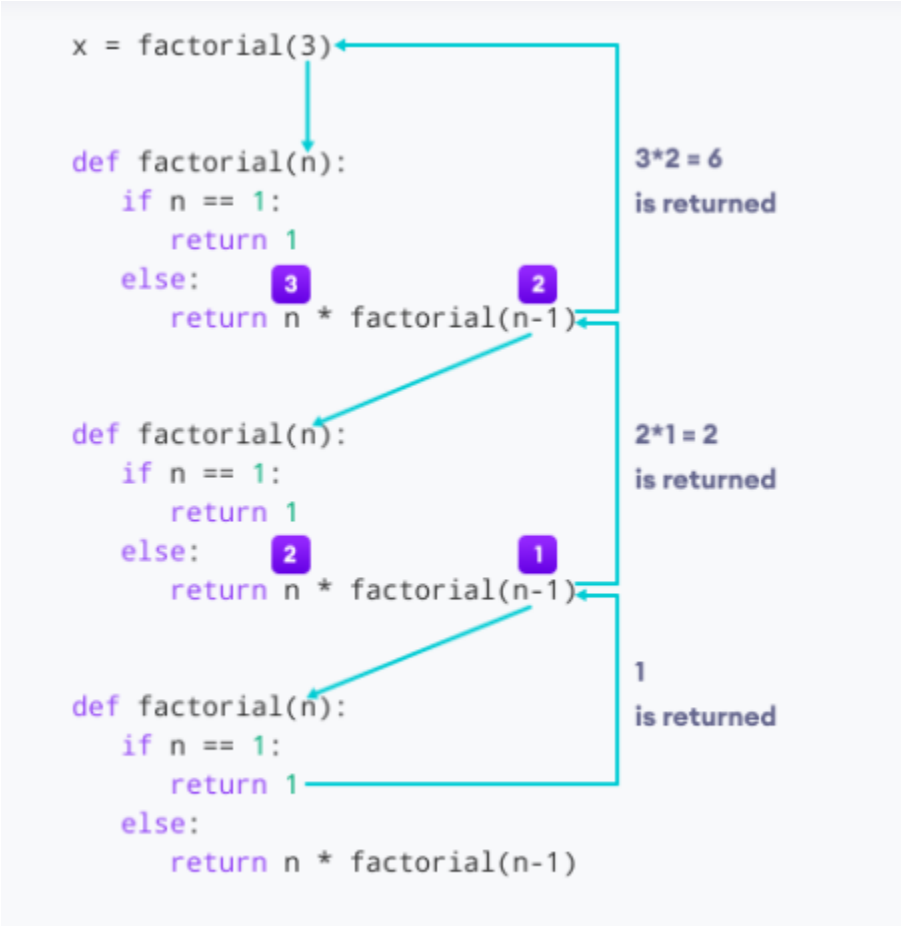
When we call this function with a positive integer, it will recursively call itself by decreasing the number.

Each function multiplies the number with the factorial of the number below it until it is equal to one. This recursive call can be explained in the following steps.

```
factorial(3)          # 1st call with 3
```

```
3 * factorial(2)      # 2nd call with 2
3 * 2 * factorial(1)  # 3rd call with 1
3 * 2 * 1             # return from 3rd call as number=1
3 * 2                 # return from 2nd call
6                     # return from 1st call
```

Let's look at an image that shows a step-by-step process of what is going on:



Working of a recursive factorial function

Advantages of Recursion

1. Recursive functions make the code look clean and elegant.
2. A complex task can be broken down into simpler sub-problems using recursion.
3. Sequence generation is easier with recursion than using some nested iteration.

Disadvantages of Recursion

1. Sometimes the logic behind recursion is hard to follow through.
2. Recursive calls are expensive (inefficient) as they take up a lot of memory and time.
3. Recursive functions are hard to debug.

Python Modules

What are modules in Python?

A module is an ordinary Python file that ends with .py extension. We can define a group of classes, functions, variables, constants and so on inside a module. To reuse the classes or functions defined inside the module, we have to import the module in our program using the import statement. The syntax of import statement is as follows:

import module_name

where module_name is the name of the file without .py extension

A file containing Python code, for example: example.py, is called a module, and its module name would be example.

We use modules to break down large programs into small manageable and organized files.

Furthermore, modules provide reusability of code.

We can define our most used functions in a module and import it, instead of copying their definitions into different programs.

Let us create a module. Type the following and save it as example.py.

```
# Python Module example

def add(a, b):
   """This program adds two
   numbers and return the result"""

   result = a + b
```

```
    return result
```

Here, we have defined a function add() inside a module named example. The function takes in two numbers and returns their sum.

How to import modules in Python?

We can import the definitions inside a module to another module or the interactive interpreter in Python.

We use the import keyword to do this. To import our previously defined module example, we type the following in the Python prompt.

```
>>> import example
```

This does not import the names of the functions defined in example directly in the current symbol table. It only imports the module name example there.
Using the module name we can access the function using the dot. Operator. For example:

```
>>> example.add(4,5.5)
9.5
```

Python has tons of standard modules. You can check out the full list of Python standard modules and their use cases. These files are in the Lib directory inside the location where you installed Python.
Standard modules can be imported the same way as we import our user-defined modules.

There are various ways to import modules. They are listed below..

Python import statement

We can import a module using the import statement and access the definitions inside it using the dot operator as described above. Here is an example.

```
# import statement example
# to import standard module math

import math
print("The value of pi is", math.pi)
```

When you run the program, the output will be:

```
The value of pi is 3.141592653589793
```

Packages : A package is a collection of Python modules. A package is a directory of Python. A package from a directory contains a bunch of Python scripts. We can use packages as follows : Create a new folder named E:\MyApp. Inside MyApp, create a subfolder with the name 'MyPack'. Create an empty __init__.py file in the "MyPack" folder. Using a Python-aware editor like IDLE, create modules greet.py and functions.py

e.g. from MyPack import functions
functions.power(4,2)
Ouput : 16

e.g. from MyPack import greet
greet.SayHello("Rohan")
Output : Hello Rohan

e.g. from MyPack.functions import div
div(500,5)
Output : 100.0

Why to use __init__.py The package folder contains a special file called __init__.py, which stores the package's content. It serves two purposes: 1.The Python interpreter recognizes a folder as the package if it contains __init__.py file. 2.__init__.py exposes specified resources from its modules to be imported.