Functions:

- A function is a group of statements made to execute them more than once in a program. A function has a name.
- Functions can compute a result value and can have parameters that serves as function inputs which may differ each time when function is executed.
- Functions are used *to reduce the size of code as it increases the code reusability* and *split a complex problem into multiple modules (functions) to improve manageability*.
- Sequential codes are easy for small scale programs. It becomes harder to keep track of details when code size exceeds.

Advantages:

→ Modularity

→ Abstraction

→ Code reusability

Functions: Modularity, Abstraction & Reusability

Modularity

Breaking program into smaller, self-contained parts

Abstraction

Hiding the complex details, showing only what is necessary

Code Reusability

Using the same code multiple times without rewriting

- A function is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a function.
- A function can or cannot return data as result.

Function Components:

- *Function signature*
  - *Function Name*
  - *Function Arguments (optional)*
- *Doc string*
- *Function Body*
- *Function return statement (optional)*

Syntax:

```
def function_name(arguments):
        doc string
        body
        return statement
```

```
def my_function(fname, lname):
    """
        function to print the first and last name
    """
    print(fname + ' ' + lname)


my_function('Red', 'apple')
Red apple
my_function('Green', 'chilli')
Green chilli
```

```
def my_function(country = "Pakistan"):
    print("I am from " + country)


my_function()
I am from Pakistan
my_function("Karachi")
I am from Karachi
```

```
def circle(r):
    """
        Returns the circumference of circle
    """
    return 2*3.14*r


print(circle.__doc__)

        Returns the circumference of circle


circle(7)
43.96
x = 5
circle(x)
31.400000000000002
```

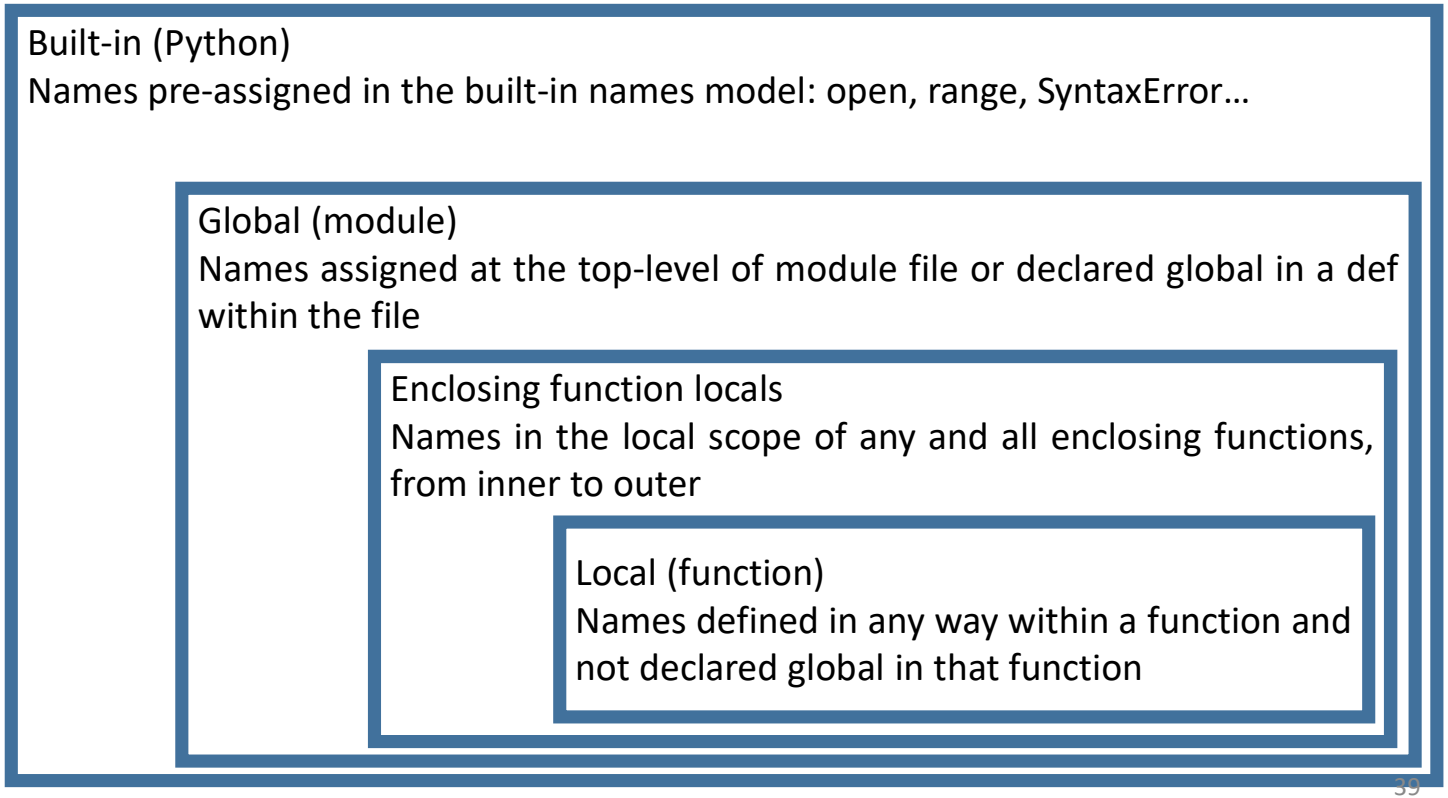
```
def circle(r):
    area = 3.14 * r **2
    circumference = 2 * 3.14 * r
    return area, circumference


circle(5)
(78.5, 31.400000000000002)
x = 7
circle(x)
(153.86, 43.96)
r = 3
circle(r)
(28.26, 18.84)

result = circle(5)
print("area = ", result[0], "and circumference = ", result[1])
area =  78.5 and circumference =  31.400000000000002
```

Scope of Variables:

Scope of variable defines the region of program where a variable is recognized and accessible.

LEGB Rule:

When Python looks for a variable, it search in this order: Local, then Enclosing, then Global and then Built-in. If variable is not found in these scopes, a ***NameError*** is raised.



Local

```
def my_function():
    local_var = 4          # THis is local variable
    print(local_var)


my_function()
4
```

Global

```
global_var = 30

def modify_global():
    global global_var
    global_var = 19
    print(global_var)


print(global_var)
30
modify_global()
19
print(global_var)
19
```

Enclosing

```
def outer_function():
    enclosing_var = 23
    def inner_function():
        print(enclosing_var)
    inner_function()


outer_function()
23
```

```
def outer():
    x =5
    def inner():
        x = 3
        print("Inner x: ", x)
    print("Outer x: ", x)
    inner()


x = 10
outer()
Outer x:  5
Inner x:  3
print("Global x:", x)
Global x: 10
```

```
def func(a,b,c):
    print(a, b, c)


# Positional mapping
func(1,2,3)
1 2 3

# keyword mapping
func(c=3, b=2, a=1)
1 2 3

# Hybird mapping
func(1, c=3, b=2)
1 2 3
```

Recursive Functions:

- Python also excepts function recursion, which means a defined function can call itself.
- Recursion is a common mathematical and programming concept. It means that a function calls itself. This has the benefit of meaning that you can loop through data to reach a result.

Components of a recursive function:

Base Case:

- Indicates the stopping condition
- Could be more than one

Recursive Call

- Moves the execution towards the base case
- Could be more than one

```
def factorial(n):
    """
        Compute the factorial of non-negative number using recursion
    """
    if n>0:
        result = n * factorial(n-1)
    else:
        result = 1
    return result

factorial(5)
120
factorial(9)
362880
factorial(0)
1
```