----:Functions:----

If a group of statements is repeatedly required then it is not recommended to write these statements every time separately. We have to define these statements as a single unit and we can call that unit any number of times based on our requirement without rewriting.This unit is nothing but function.

```
x=10
y=20
print("Addition of x & y =",x+y)
print("Addition of x & y =",x-y)
print("Addition of x & y =",x*y)

x=200
y=100
print("Addition of x & y =",x+y)
print("Addition of x & y =",x-y)
print("Addition of x & y =",x*y)

x=10
y=5
print("Addition of x & y =",x+y)
print("Addition of x & y =",x-y)
print("Addition of x & y =",x*y)

O/P:--
Addition of x & y = 30
Addition of x & y = -10
Addition of x & y = 200
Addition of x & y = 300
Addition of x & y = 100
Addition of x & y = 20000
Addition of x & y = 15
Addition of x & y = 5
Addition of x & y = 50
```

Now, repeated code can be bound into single unit that is called function.

The advantages of function:

1. **Maintaining the code is an easy way.**
2. **Code re-usability.**

Now,

```
def calculate(x, y):
   print("Addition of x & y =",x+y)
    print("Addition of x & y =",x-y)
    print("Addition of x & y =",x*y)

calculate(10,20)
calculate(200,100)
calculate(10,5)

O/P:--
Addition of x & y = 30
Addition of x & y = -10
Addition of x & y = 200
Addition of x & y = 300
Addition of x & y = 100
Addition of x & y = 20000
Addition of x & y = 15
Addition of x & y = 5
Addition of x & y = 50
```

Types of function:---

1. **In-built function :--** The functions which are coming along with Python software automatically,are called built-in functions or pre defined functions.
   Examples:--
   1. print()
   2. id()
   3. type()
   4. len()
   5. eval()
   6. sorted()
   7. count() etc…….

2. User define function:--- The functions which are defined by the developer as per the requirement are called user-defined functions.

Syntax:---

```
def fun_name(parameters….):
  ‘‘‘ doc string….’’’

  Statment1…….
  Statment2…….
  Statment3…….

  return (anything)
# call function
fun_name(arguments....)
```

Important terminology	
def-keyword	mandatory
return-keyword	optional
arguments	optional
parameters	optional
fun_name	mandatory

1. **def keyword –** Every function in python should start with the keyword ‘def’. In other words, python can understand the code as part of a function if it contains the ‘def’ keyword only.
2. **Name of the function –** Every function should be given a name, which can later be used to call it.
3. **Parenthesis** – After the name ‘()’ parentheses are required
4. **Parameters** – The parameters, if any, should be included within the parenthesis.
5. **Colon symbol ‘:’** should be mandatorily placed immediately after closing the parentheses.
6. **Body –** All the code that does some operation should go into the body of the function. The body of the function should have an indentation of one level with respect to the line containing the ‘def’ keyword.
7. **Return statement** – Return statement should be in the body of the function. It’s not mandatory to have a return statement. If we are not writing return statement then default return value is **None**
8. **Arguments**:-- At the time of calling any function, in between the parentheses we passes arguments.

Relation between parameters and arguments:--

When we are creating a function, if we are using parameters in between parenthesis, then it is compulsory to at the time of calling this function, you need to pass correspond arguments.

Parameters are inputs to the function. If a function contains parameters, then at the time of calling, compulsory we should provide values as a arguments, otherwise we will get error.

```
def calculate(x, y):
    print("Addition of x & y =",x+y)
    print("Addition of x & y =",x-y)
    print("Addition of x & y =",x*y)

calculate(10,20)
calculate(200,100)
calculate(10,5)

O/P:--
Addition of x & y = 30
Addition of x & y = -10
Addition of x & y = 200
Addition of x & y = 300
Addition of x & y = 100
Addition of x & y = 20000
Addition of x & y = 15
Addition of x & y = 5
Addition of x & y = 50
```

```
# Write a function to take number as input and print its square value
def square(x):
    print("The Square of",x,"is", x*x)
square(4)
square(5)

O/P:--
The Square of 4 is 16
The Square of 5 is 25
```

```
# Write a function to check whether the given number is even or odd?
def even_odd(num):
    if num%2==0:
        print(num,"is Even Number")
    else:
        print(num,"is Odd Number")

even_odd(10)
even_odd(15)

O/P:--
10 is Even Number
15 is Odd Number
```

```
# Write a function to find factorial of given number?
def fact(num):
    result=1
    while num>=1:
        result=result*num
        num=num-1
    return result
i=int(input("Enter any no "))
print("The Factorial of",i,"is :",fact(i))

O/P:--
Enter any no 5
The Factorial of 5 is : 120
```

Returning multiple values from a function: In other languages like C, C++ and Java, function can return almost one value. But in Python, a function can return any number of values.

```
def add_sub(a,b):
    add=a+b
    sub=a-b
    return add,sub

x,y=add_sub(100,50)
```

```
print("The Addition is :",x)
print("The Subtraction is :",y)

O/P:--
The Addition is : 150
The Subtraction is : 50

        Or

def add_sub(a,b):
    add=a+b
    sub=a-b
    return add,sub
x,y=int(input("Enter first value:")),int(input("Enter second value: "))
print("The Addition is :",x)
print("The Subtraction is :",y)

O/P:--
The Addition is : 100
The Subtraction is : 50
```

```
def calc(a,b):
    add=a+b
    sub=a-b
    mul=a*b
    div=a/b
    return add,sub,mul,div

x,y,z,p=calc(int(input("Enter first value:")),int(input("Enter second value: ")))
print("The Addition is",x)
print("The Substraction is",y)
print("The Multip is",z)
print("The Division is",p)
O/P:--
Enter first value:100
Enter second value: 10
The Addition is 110
The Substraction is 90
The Multip is 1000
The Division is 10.0
```

Types of arguments:

```
def f1(a,b):
        ------
        ------
f1(10,20)
```

There are 4 types are actual arguments are allowed in Python.

1. **positional arguments:**

```
def f1(a,b):
        ------
        ------
f1(10,20)
```

```
def square(x):
   print("The Square of",x,"is", x*x)
square(4)
square(5)

O/P:-
The Square of 4 is 16
The Square of 5 is 25
```

2. **keyword arguments:**

```
def f1(a,b):
        ------
        ------
f1(a=10,b=20)
```

```
def square(x):
   print("The Square of",x,"is", x*x)
square(x=4)
square(x=5)

O/P:--
The Square of 4 is 16
The Square of 5 is 25
```

3. **default arguments:**

```
def f1(a=0,b=0):
        ------
        ------
f1(10,20)
f1()
```

```
def square(x=0):
   print("The Square of",x,"is", x*x)
square(x=4)
square()

O/P:--
The Square of 4 is 16
The Square of 0 is 0
```

4. **Variable length arguments:**

```
def f1(*n):
        ------
        ------
f1(10)
f1(10,20)
f1(10,20,30)
```

```
def sum(*n):
   total=0
   for i in n:
      total=total+i
   print("The Sum=",total)
sum()
sum(10)
sum(10,20)
sum(10,20,30,40)

O/P:--
The Sum= 0
The Sum= 10
The Sum= 30
The Sum= 100
```

5. **key word variable length arguments:**

```
def f1(**n):
        ------
        ------
f1(n1=10, n2=20)
```

```
def display(**kwargs):
    for k,v in kwargs.items():
        print(k,"=",v)
display(n1=10,n2=20,n3=30)
print("-----------")
display(rno=100, name="Neeraj", marks=70, subject="Java")

O/P:--
n1 = 10
n2 = 20
n3 = 30
-----------
rno = 100
name = Neeraj
marks = 70
subject = Java

# ================================================
def display_info(**kwargs):
    for key, value in kwargs.items():
        print(f"{key}: {value}")

display_info(Name="Neeraj",age=37)

O/P:---
Name: Neeraj
age: 37
```

```
def args_and_kwargs(*args, **kwargs):
    print("Positional arguments:")
    for arg in args:
        print(arg)
    print("Keyword arguments:")
    for key, value in kwargs.items():
        print(f"{key}: {value}")
args_and_kwargs(1, 2, 3, name="Neeraj",age=37, quali="M.Tech")

O/P:--
Positional arguments:
1
2
3
Keyword arguments:
name: Neeraj
age: 37
quali: M.Tech
```

Types of Variables in Python

The variables based on their scope can be classified into two types:

1. **Local variables**
2. **Global variables**

Local Variables in Python:

The variables which are declared inside of the function are called local variables. Their scope is limited to the function i.e we can access local variables within the function only. If we are trying to access local variables outside of the function, then we will get an error.

```
def a():
    x=10
    return "value of Local variable is:",x
def b():
    return "value of Local variable is:",x
p=a()
print(p)
y=b()
print(y)
```

```
O/P:-
('value of Local variable is:', 10)
Traceback (most recent call last):
  File "E:\Python Core_Advance\local.py", line 10, in <module>
    y=b()
  File "E:\Python Core_Advance\local.py", line 6, in b
    return "value of Local variable is:",x
NameError: name 'x' is not defined
```

We Can't access local variable outside the function:

```
def a():
    x=10
    return "value of Local variable is:",x

def b():
    return "value of Local variable is:",x

p=a()
print(p)
print(x)

O/P:--
('value of Local variable is:', 10)
Traceback (most recent call last):
  File "E:\Python Core_Advance\local.py", line 12, in <module>
    print(x)
NameError: name 'x' is not defined
```

Global variables in Python:

The variables which are declared outside of the function are called global variables. Global variables can be accessed in all functions of that module.

```
a=11
b=12
def m():
   print("a from function m(): ",a)
   print("b from function m(): ",b)
def n():
   print("a from function n(): ",a)
   print("b from function n(): ",b)
m()
```

```
n()


O/P:--
a from function m():  11
b from function m():  12
a from function n():  11
b from function n():  12
```

GLOBAL KEYWORD IN PYTHON

The keyword global can be used for the following 2 purposes:

1. To declare a global variable inside a function
2. To make global variables available to the function.

```
def m1():
  global a
  a=2
  print("a value from m1() function: ", a)
def m2():
  print("a value from m2() function:", a)
m1()
m2()

O/P:--
a value from m1() function: 2
a value from m2() function: 2
```

global and local variables having the same name in Python

```
a=1
def m1():
  global a
  a=2
  print("a value from m1() function:", a)
def m2():
  print("a value from m2() function:", a)
m1()
m2()
```

```
O/P:--
a value from m1() function: 2
a value from m2() function: 2
```

If we use the global keyword inside the function, then the function is able to read-only global variables.
PROBLEM: This would make the local variable no more available.

globals() built-in function in python:
The problem of local variables not available, due to the use of global keywords can be overcome by using the Python built-in function called globals(). The globals() is a built-in function that returns a table of current global variables in the form of a dictionary. Using this function, we can refer to the global variable "a" as: global()["a"].

```
a=1
def m1():
  a=2
  print("a value from m1() function:", a)
  print("a value from m1() function:", globals()['a'])

m1()

O/P:--
a value from m1() function: 2
a value from m1() function: 1
```

---: Higher order function :---

A function in Python with another function as an argument or returns a function as an output is called the High order function. A function that is having another function as an argument or a function that returns another function as a return in the output

- The function can be stored in a variable.
- The function can be passed as a parameter to another function.
- The high order functions can be stored in the form of lists, hash tables, etc.
- Function can be returned from a function.

1. **Map()**
2. **Filter()**
3. **Lambda()**
4. **Reduce()**
5. **Decorators()**
6. **Generators()**

---: Map :----

Python's map() is a built-in function that enables the processing and transformation of all items in an iterable without the need for an explicit for loop, a technique referred to as mapping. This function is particularly useful when you want to apply a transformation function to each element in an iterable, producing a new iterable as a result. map() is one of the tools that facilitate a functional programming approach in Python.

map() Syntax

map(function, iterable, ...)

map() Arguments

The map() function takes two arguments:

1. function - a function
2. iterable - an iterable like sets, lists, tuples, etc

The map() function returns an object of map class. The returned value can be passed to functions like list() - to convert to list, set() - to convert to a set, and so on.

Example:-1

```
# Map() higher order function----------

my_list=[10,20,30,40]

def sqr(n):
    return n*n

x=map(sqr,my_list)
print(x)
print(list(x))

O/P:--
<map object at 0x000001EA310E3490>
[100, 400, 900, 1600]
```

Example:-2

```
my_tuple=(10,20,30,40)

def sqr(n):
    return n*n

x=map(sqr,my_tuple)
print(x)
print(tuple(x))

O/P:-
<map object at 0x0000019833A83490>
(100, 400, 900, 1600)
```

Example:-3

```
my_str="Neeraj"
def add(n):
    x=ord(n)
    return x
x=map(add,my_str)
print(x)
print(list(x))

O/P:-
<map object at 0x000001D03A4E3490>
[78, 101, 101, 114, 97, 106]
```

Example:-4

```
my_str="Neeraj"
def add(n):
    x=ord(n)
    return chr(x+5)
x=map(add,my_str)
print(x)
print(list(x))

O/P:-
<map object at 0x0000026D8F1634C0>
['S', 'j', 'j', 'w', 'f', 'o']
```

---: Filter :---

The filter function extracts elements from an iterable (such as a list or tuple) based on the results of a specified function. This function is applied to each element of the iterable, and if it returns True that element is included in the output of the filter() function.

filter() Syntax

The syntax of filter() is: **filter(function, iterable)**

filter() Arguments

The filter() function takes two arguments:

1. **function** - a function
2. **iterable** - an iterable like sets, lists, tuples etc.

The filter() function returns an iterator.

Example 1:-

```
# filter() higher order function ----------------
my_list=[60,10,70,90,55,75,10,20,40]

def fun(n):
    if n>=60:
        return True

x=filter(fun , my_list)
print(list(x))

O/P:--
[60, 70, 90, 75]
```

Example 2:-

```
def check_even(number):
    if number % 2 == 0:
        return True
    return False

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_numbers_iterator = filter(check_even, numbers)
even_numbers = list(even_numbers_iterator)
print(even_numbers)

O/P:--
[2, 4, 6, 8, 10]
```

Example 3:-

```
def check_odd(number):
    if number % 2 != 0:
        return True
    return False

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
odd_numbers_iterator = filter(check_odd, numbers)
odd_numbers = list(odd_numbers_iterator)
print(odd_numbers)

O/P:--
[1, 3, 5, 7, 9]
```

---: Lambda :---

a lambda function is a special type of function without the function name. For example, lambda : print('Hello World').

A lambda function can take any number of arguments, but can only have one expression.

Here, we have created a lambda function that prints 'Hello World'.

lambda Function Declaration: We use the **lambda keyword** instead of def to create a lambda function. Here's the syntax to declare the lambda function:

Syntex:----

lambda argument(s) : expression

argument(s) - any value passed to the lambda function

expression - expression is executed and returned

Let's see an example,

```
# without argument
greet = lambda : print('Hello World')
greet()

O/P:--
Hello World
```

greet = lambda : print('Hello World'). Here, we have defined a lambda function and assigned it to the variable named greet. In the above example, we have defined a lambda function and assigned it to the greet variable. When we call the lambda function, the print() statement inside the lambda function is executed.

```
# with argument
x=lambda p,q,r:3*p+4*q+5*r+5
print(x(10,20,30))
O/P:-
265
```

```
# with argument
user = lambda name : print('Hello', name)
user('Neeraj')

O/P:--
Hello Neeraj
```

---: Reduce :---

The reduce() function in Python is part of the functools module, which needs to be imported before it can be used.

This function performs functional computation by taking a function and an iterable (such as a list, tuple, or dictionary) as arguments. It applies the function cumulatively to the elements of the iterable, reducing it to a single value. Unlike other functions that may return multiple values or iterators, reduce() returns a

single value, which is the result of the entire iterable being condensed into a single integer, string, or boolean.

Steps of how to reduce function works:

1. The function passed as an argument is applied to the first two elements of the iterable.
2. After this, the function is applied to the previously generated result and the next element in the iterable.
3. This process continues until the whole iterable is processed.
4. The single value is returned as a result of applying the reduce function on the iterable.

```
from functools import reduce

def product(x,y):
    return x*y

ans = reduce(product, [2, 5, 3, 7])
print(ans)

O/P:--
210
```

```
import functools

my_list=(10,20,60,30,40)
def greater(a,b):
    if a>b:
        return a
    else:
        return b
x=functools.reduce(greater,my_list)
print(x)
```

```
O/P:-
60

my_list=(10,20,60,30,40)
def lowest_digit(a,b):
    if a<b:
        return a
    else:
        return b
x=functools.reduce(lowest_digit,my_list)
print(x)

O/P:-
10

my_str="Neeraj"
def greater(a,b):
    if a>b:
        return a
    else:
        return b
x=functools.reduce(greater,my_str)
print("This char have greater asci value:",x)

O/P:-
This char have greater asci value: r
```