Functions in Python

A function can be defined as the organized block of reusable code, which can be called whenever required.

There are mainly two types of functions.

- **Built-in functions** - The built-in functions are those functions that are **pre-defined** in Python.
- **User-define functions** - The user-defined functions are those define by the **user** to perform the specific task.

Advantage of Functions in Python

There are the following advantages of Python functions.

- **Reusability** is the main achievement of Python functions.
- Using functions, we can avoid rewriting the same logic/code again and again in a program.
- We can call Python functions multiple times in a program and anywhere in a program.
- We can track a large Python program easily when it is divided into multiple functions.

Syntax of functions in Python:

Functions declaration:
def my_function(parameters): function_block # **return** expression

Let's understand the syntax of functions definition.

- The **def** keyword, along with the function name is used to define the function.
- The identifier rule must follow the function name.
- A function accepts the parameter (argument), and they can be optional.
- The function block is started with the colon (:), and block statements must be at the same indentation.

- The **return** statement is used to return the value. A function can have only one **return**

Example:

```
#called function
def  display():
        print("Welcome")

# calling function
display()
display()
display()
```

Output:

```
Welcome
welcome
welcome
```

Based on the data flow between the calling function and called function, the functions are classified as follows...

- Function without Parameters and without Return value
- Function with Parameters and without Return value
- Function without Parameters and with Return value
- Function with Parameters and with Return value

Example : write a program to find the sum of two integers.

Expected output:

```
Enter any two integers:
10
20
Sum = 30
```

Function without Parameters and without Return value

```
def sum():
   print("Enter any Two Integers")
   a = int(input())
   b = int(input())
   print("Sum = ",(a+b))

#calling the functions
sum()
```

Function with Parameters and without Return value

```
def sum(x,y):
   print("Sum = ",(x+y))


print("Enter any Two Integers")
a=int(input())
b=int(input())
sum(a,b)   # calling function using two arguements
```

Function without Parameters and with Return value

```
def sum():    # sum() read two values and perform sum and return result
        print("Enter any two integers")
        a=int(input())
        b=int(input())
        return (a+b)

print("Sum", sum())    # it calls the sum() functions,
```

Function with Parameters and with Return value

```
        def sum(a,b):
           return (a+b)

        print("Enter any two integers")
        x = int(input())
        y = int(input())

        print("Sum = ", sum(x,y))
```

Returning multiple values from a function

A function can returns a single value in the programming languages like C, C++ and JAVA. But, in python, a function can return multiple values. When a function calculates multiple results and wants to return the results, we can use return statement as:

return a, b, c

Here, three values which are in „a", „b" and „c" are returned. These values are returned by the function as a tuple. To grab these values, we can three variables at the time of calling the function as:

x, y, z = functionName( )

Here, x,y and z are receiving the three values returned by the function.

Write a program for arithmetic operator

```
# arithmetic function  takes two integers and perform all arithmetic operator

def arithmatic(a,b):
    return (a+b),(a-b),(a*b),(a/b),(a%b),(a//b),(a**b)




print("Enter Any Two Integer")
x = int(input())
y = int(input())


# calling function ,
sum,sub,mul,div,mod,floor_div,power  = arithmatic(x,y)

print("Sum =",sum)
print("Subtraction =",sub)
print("Multiplication = ",mul)
print("Division=",div)
print("Modulus =",mod)
print("Floor Div=",floor_div)
print("Power =",power)
```

In Python, there are different ways to pass arguments to a function. They are as follows.

- **Positional Arguments (or) Required Arguments**
- **Default Arguments**
- **Keyword Arguments**
- **Variable-length Arguments**

Positional Arguments (or) Required Arguments:

The positional arguments are the arguments passed to a function in the same positional order as they defined in the function definition.

Example : Positional Arguments

```
def display(a,b,c,d,e):
   print(a)
   print(b)
   print(c)
   print(d)
   print(e)


  # 10,20,30,40,50 are the positional arguments, they are assign to a,b,c,d,e respectively

display(10,20,30,40,50)

print("Task Has Completed")
```

Default Arguments

Python allows us to initialize the arguments at the function definition.

Example :

```
display(a=0,b=0,c=0):
        pass
```

If the value of any of the arguments is not provided at the time of function call, then that argument can be initialized with the value given in the definition even if the argument is not specified at the function call.

Example :

```
def display(a=0,b=0,c=0,d=0,e=0):
   print(a)
   print(b)
   print(c)
   print(d)
   print(e)

display()  # all are 0 s

display(10) # a = 10 and b,c,d,e, are 0s

display(10,20) # a =10, b=20 and c,d, and e are 0
```

```
display(10,20,30) # output: 10 20 30 and d,e are 0

display(10,20,30,40)# output : 10, 20 30 40 and e is 0

display(10,20,30,40,50) # output : 10,20,30 , 40 and 50

display(10,20,30,40,50,60) # error, we can no send more than 5 arguments
```

Keyword arguments:

Python allows us to call the function with the keyword arguments. This kind of function call will enable us to pass the arguments in the random order.

The name of the arguments is treated as the keywords and matched in the function calling and definition. If the same match is found, the values of the arguments are copied in the function definition.

Example:

```
def display(a=0,b=0,c=0,d=0,e=0):
    print(a)
    print(b)
    print(c)
    print(d)
    print(e)

display(a=10,c=20) # passing arguments as key and value pair

display(d=20,a=10) # passing arguments to only d and a as 20 and 10
```

Variable length arguments / unlimited number of arguments / arbitrary arguments

sometimes we may not know the number of arguments to be passed in advance. In such cases, Python provides us the flexibility to offer the comma-separated values which are internally treated as tuples at the function call. By using the variable-length arguments, we can pass any number of arguments.

However, at the function definition, we define the variable-length argument using the ***args** (star) as *<variable - name >.

Example :

```
def sum_n(*a):
    sum =0
    for i in a:
        sum+=i
```

```
    print("SUm = ",sum)

sum_n(10,20,30,40,50,60)      # we can supply any no of arguments – no limit
```

Note:

def fun_name(*a) – valid

def fun_name (x,*a) – valid , one mandatory positional arguments

def fun_name(x,y=0,*a) – valid, one mandatory, secnd default, 3rd arbitrary

def fun_name (*a, x) – valid , but , x should be keyword argument and mandatory value, and x must be supply as a positional arguments(we need to call at end)

def fun_name(*a,x=0) , valid , but x is default argument, should supply value as keyword argument