Chapter - 6 : Functions

Function Arguments

A function can be called by using the following types of formal arguments −

- Required arguments
- Keyword arguments(**kwargs**)
- Default arguments
- Variable-length arguments or Arbitrary Arguments(***args**)
- Arbitrary Keyword Arguments (****kwargs**)

Variable-length Arguments or Arbitrary Arguments(*args)

- If you do not know how many arguments that will be passed into your function, add an asterisk (*) before the parameter name in the function definition.
- This variable name holds the values of all non keyword variable arguments.
- The function will receive a **tuple** of arguments, and can access the items accordingly. This **tuple** remains empty if no additional arguments are specified during the function call.

Example -1

```
def my_function(*kids):
    print("The youngest child is " + kids[2])

my_function("Ajay", "Vijay", "Sanjay")
```

Output- The youngest child is Sanjay

Example -2

```
def printno( arg1, *vartuple ):
    "This prints a variable passed arguments"
    print ("Output is: ")
    print (arg1)

    for var in vartuple:
       print (var)
```

```
printno( 10 )
printno( 70, 60, 50 )
```

Output is:
10
Output is:
70
60
50

Arbitrary Keyword Arguments (**kwargs)

- If you do not know how many keyword arguments that will be passed into your function, add two asterisk(**) before the parameter name in the function definition.
- This way the function will receive a **dictionary** of arguments, and can access the items accordingly.

```
def my_function(**kid):
  print("His last name is " + kid["lname"])

my_function(fname = "Ajay", lname = "Kumar")
```

Recursion

Python also accepts function recursion, which means a defined function can call itself.

```
def fact(k):
   "This function returns the factorial of a number"

    If (k > 0):
       f = k * fact(k - 1)
    else:
        f = 1
    return f

print("\n\n Recursion Example Results")
r=fact(5)
print("factorial=", r)
```

The Anonymous Functions

- The functions are called anonymous when it is not declared in the standard manner by using the **def** keyword.
- The **lambda** keyword used to create small anonymous functions.
- Lambda forms can take any number of arguments but return just one value in the form of an expression. They cannot contain commands or multiple expressions.
- An anonymous function cannot be a direct call to print because lambda requires an expression.
- Lambda functions have their own local namespace and cannot access variables other than those in their parameter list and those in the global namespace.

Syntax

lambda [arg1 [,arg2,.....argn]] : expression

```
# Function definition is here
sum = lambda arg1, arg2: arg1 + arg2

# Now you can call sum as a function
print ("Value of total : ", sum( 10, 20 ))
print ("Value of total : ", sum( 20, 20 ))
```

Scope 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.

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 we call a function, the variables declared inside it are brought into scope.

Example 1 : In this example TOTAL is the Local variable in SUM ( ) function.

```
total = 0   # This is global variable.

# Function definition i
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)

# Now you can call sum function
sum( 10, 20 )
print ("Outside the function global total : ", total )
```

Output

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

Example 2 : In this example TOTAL is the GLOBAL variable in SUM ( ) function. GLOBAL keyword is used link the variable defined within the function with the global scope.

```
total = 0   # This is global variable.
# Function definition
def sum( arg1, arg2 ):
   # Add both the parameters and return them."
global total
 total = arg1 + arg2;  # Here total is local variable.
 print ("Inside the function local total : ", total)

# Now you can call sum function
sum( 10, 20 )
print ("Outside the function global total : ", total )
```

Output

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