Python Functions:

A Function is a self block of code.

A Function can be called as a section of a program that is written once and can be executed whenever required in the program, thus making code reusability.

A Function is a subprogram that works on data and produce some output.

**Types of Functions:**

There are two types of Functions.

a) Built-in Functions: Functions that are predefined. We have used many predefined functions in Python.

b) User- Defined: Functions that are created according to the requirements.

Defining a Function:

A Function defined in Python should follow the following format:

1) Keyword def is used to start the Function Definition. Def specifies the starting of Function block.

2) def is followed by function-name followed by parenthesis.

3) Parameters are passed inside the parenthesis. At the end a colon is marked.

**Syntax:**

**def** <function\_name>([parameters]):

</function\_name>

eg:

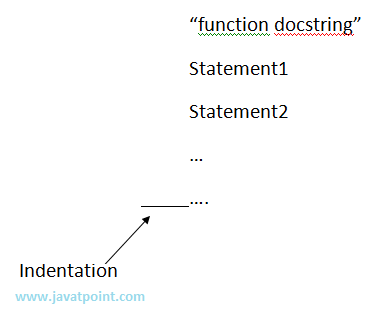
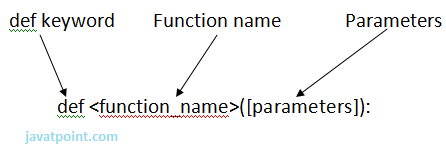
1. **def** sum(a,b):

4) Before writing a code, an Indentation (space) is provided before every statement. It should be same for all statements inside the function.

5) The first statement of the function is optional. It is ? Documentation string? of function.

6) Following is the statement to be executed.

**Syntax:**



Invoking a Function:

To execute a function it needs to be called. This is called function calling.

Function Definition provides the information about function name, parameters and the definition what operation is to be performed. In order to execute the Function Definition it is to be called.

**Syntax:**

1. <function\_name>(parameters)
2. </function\_name>

eg:

1. sum(a,b)

here sum is the function and a, b are the parameters passed to the Function Definition.

Let?s have a look over an example:

**eg:**

#Providing Function Definition

**def** sum(x,y):

     "Going to add x and y"

      s=x+y

**print** "Sum of two numbers is"

**print** s

       #Calling the sum Function

      sum(10,20)

      sum(20,30)

**Output:**

1. >>>
2. Sum of two numbers **is**
3. 30
4. Sum of two numbers **is**
5. 50
6. >>>

NOTE: Function call will be executed in the order in which it is called.

return Statement:

return[expression] is used to send back the control to the caller with the expression.

In case no expression is given after return it will return None.

In other words return statement is used to exit the Function definition.

Eg:

1. **def** sum(a,b):
2. "Adding the two values"
3. **print** "Printing within Function"
4. **print** a+b
5. **return** a+b
6. **def** msg():
7. **print** "Hello"
8. **return**
10. total=sum(10,20)
11. **print** ?Printing Outside: ?,total
12. msg()
13. **print** "Rest of code"

Output:

1. >>>
2. Printing within Function
3. 30
4. Printing outside:  30
5. Hello
6. Rest of code
7. >>>

Argument and Parameter:

There can be two types of data passed in the function.

1) The First type of data is the data passed in the function call. This data is called ?arguments?.

2) The second type of data is the data received in the function definition. This data is called ?parameters?.

Arguments can be literals, variables and expressions.

Parameters must be variable to hold incoming values.

Alternatively, arguments can be called as actual parameters or actual arguments and parameters can be called as formal parameters or formal arguments.

Eg:

**def** addition(x,y):

**print** x+y

x=15

addition(x ,10)

addition(x,x)

y=20

addition(x,y)

Output:

1. >>>
2. 25
3. 30
4. 35
5. >>>

Passing Parameters

Apart from matching the parameters, there are other ways of matching the parameters.

Python supports following types of formal argument:

1) Positional argument (Required argument).

2) Default argument.

3) Keyword argument (Named argument)

**Positional/Required Arguments:**

When the function call statement must match the number and order of arguments as defined in the function definition it is Positional Argument matching.

**Eg:**

#Function definition of sum

**def** sum(a,b):

            "Function having two parameters"

         c=a+b

**Print**(c)

sum(10,20)

sum(20)

**Output:**

1. >>>
2. 30
4. Traceback (most recent call last):
5. File "C:/Python27/su.py", line 8, **in** <module>
6. sum(20)
7. TypeError: sum() takes exactly 2 arguments (1 given)
8. >>>
9. </module>

**Explanation:**

1) In the first case, when sum() function is called passing two values i.e., 10 and 20 it matches with function definition parameter and hence 10 and 20 is assigned to a and b respectively. The sum is calculated and printed.

2) In the second case, when sum() function is called passing a single value i.e., 20 , it is passed to function definition. Function definition accepts two parameters whereas only one value is being passed, hence it will show an error.

Default Arguments

Default Argument is the argument which provides the default values to the parameters passed in the function definition, in case value is not provided in the function call.

**Eg:**

#Function Definition

**def** msg(Id,Name,Age=21):

            "Printing the passed value"

**print** Id

**print** Name

**print** Age

**return**

#Function call

msg(Id=100,Name='Ravi',Age=20)

msg(Id=101,Name='Ratan')

**Output:**

1. >>>
2. 100
3. Ravi
4. 20
5. 101
6. Ratan
7. 21
8. >>>

**Explanation:**

1) In first case, when msg() function is called passing three different values i.e., 100 , Ravi and 20, these values will be assigned to respective parameters and thus respective values will be printed.

2) In second case, when msg() function is called passing two values i.e., 101 and Ratan, these values will be assigned to Id and Name respectively. No value is assigned for third argument via function call and hence it will retain its default value i.e, 21.

Keyword Arguments:

Using the Keyword Argument, the argument passed in function call is matched with function definition on the basis of the name of the parameter.

**Eg:**

**def** msg(id,name):

         "Printing passed value"

**print** id

**print** name

**return**

msg(id=100,name='Raj')

msg(name='Rahul',id=101)

**Output:**

1. >>>
2. 100
3. Raj
4. 101
5. Rahul
6. >>>

**Explanation:**

1) In the first case, when msg() function is called passing two values i.e., id and name the position of parameter passed is same as that of function definition and hence values are initialized to respective parameters in function definition. This is done on the basis of the name of the parameter.

2) In second case, when msg() function is called passing two values i.e., name and id, although the position of two parameters is different it initialize the value of id in Function call to id in Function Definition. same with name parameter. Hence, values are initialized on the basis of name of the parameter.

Anonymous Function:

Anonymous Functions are the functions that are not bond to name.

Anonymous Functions are created by using a keyword "lambda".

Lambda takes any number of arguments and returns an evaluated expression.

Lambda is created without using the def keyword.

**Syntax:**

**lambda** arg1,args2,args3,?,argsn :expression

**Output:**

1. #Function Definiton
2. square=**lambda** x1: x1\*x1
4. #Calling square as a function
5. **print** "Square of number is",square(10)

**Output:**

1. >>>
2. Square of number **is** 100
3. >>>

Difference between Normal Functions and Anonymous Function:

Have a look over two examples:

**Eg:**

Normal function:

1. #Function Definiton
2. **def** square(x):
3. **return** x\*x
5. #Calling square function
6. **print** "Square of number is",square(10)

Anonymous function:

1. #Function Definiton
2. square=**lambda** x1: x1\*x1
4. #Calling square as a function
5. **print** "Square of number is",square(10)

**Explanation:**

Anonymous is created without using def keyword.

lambda keyword is used to create anonymous function.

It returns the evaluated expression.

Scope of Variable:

Scope of a variable can be determined by the part in which variable is defined. Each variable cannot be accessed in each part of a program. There are two types of variables based on Scope:

1) Local Variable. – Only access inside function only

2) Global Variable. – Access both inside and outside functions

**1) Local Variables:**

Variables declared inside a function body is known as Local Variable. These have a local access thus these variables cannot be accessed outside the function body in which they are declared.

**Eg:**

**def** msg():

           a=10

**print** "Value of a is",a

**return**

msg()

**print** a #it will show error since variable is local

**Output:**

1. >>>
2. Value of a **is** 10
4. Traceback (most recent call last):
5. File "C:/Python27/lam.py", line 7, **in** <module>
6. **print** a #it will show error since variable is local
7. NameError: name 'a' **is** **not** defined
8. >>>
9. </module>

**b) Global Variable:**

Variable defined outside the function is called Global Variable. Global variable is accessed all over program thus global variable have widest accessibility.

**Eg:**

1. b=20
2. **def** msg():
3. a=10
4. **print** "Value of a is",a
5. **print** "Value of b is",b
6. **return**
8. msg()
9. **print** b

**Output:**

1. >>>
2. Value of a **is** 10
3. Value of b **is** 20
4. 20
5. >>>