Functions in Python 3

Function is defined as the named group of statements which can be used to perform a specific task. A function in Python can contain one or more Python statements.

When we call a function, all the statements written within the function body get executed on after another.

Advantages of a function in Python 3

- It is easy to locate and correct errors in a function.
- Function reduces complexity of a program.
- It is easy to perform testing of a program.
- They are used to divide a bigger problem into smaller sections.
- A function once defined can be used by other functions as well as programs so a programmer can build a new application using existing functions .
- It helps in avoiding repetition of code which saves time and space.

Defining a function in Python

A function must be defined before it can be called within a program.

Syntax of a Function:

```
def  Func_name(Arguments):
    Local Variables
    Statement_Block
```

```
return(Expression)
```

def specifies that a function definition is starting.

Func_name refers to the name of function defined by the programmer. It should follow all the rules of a valid identifier of Python.

Arguments is the set of variables declared within the pair of parenthesis after the function name.

Each of these variables are known as function arguments/parameters. Arguments are used to receive values from another functions.

Local Variables: In this part, we declare local variables of function. These variables are usable only within the function only.

These variables automatically get destroyed when program control leaves the function. Indent must be given before local variables.

Statement-Block: It is set of valid Python statements which would execute when we call the function within another function.

Indent must be given before statements that are part of Statement-Block.

return is used when function returns some value. It appears with an expression within a pair of parenthesis after it.

Indent must be given before return statement.

Expression: It is any valid expression of Python that is to be returned by the function.

Program for defining function in Python

```
def message():
    print("Hello Python")
def shownumber(num):
    print(num)
```

Two functions namely **message()** and **shownumber()** have been defined in above program.

Calling a Function in Python (Using a Function in Python)

A function can be called by specifying its name followed by the required number of actual arguments separated by commas enclosed in parenthesis.

If no arguments are required then there must be empty pair of parentheses after function name.

To call function message() defined above, we can write the following statement

message()

Output

Hello Python

To call function shownumber() defined above, we can write the following statement.

shownumber(10)

Output

10

On calling function shownumber(10), value 10 is sent as argument to function. 10 will go and store in argument named num. Value of num is displayed in function body.

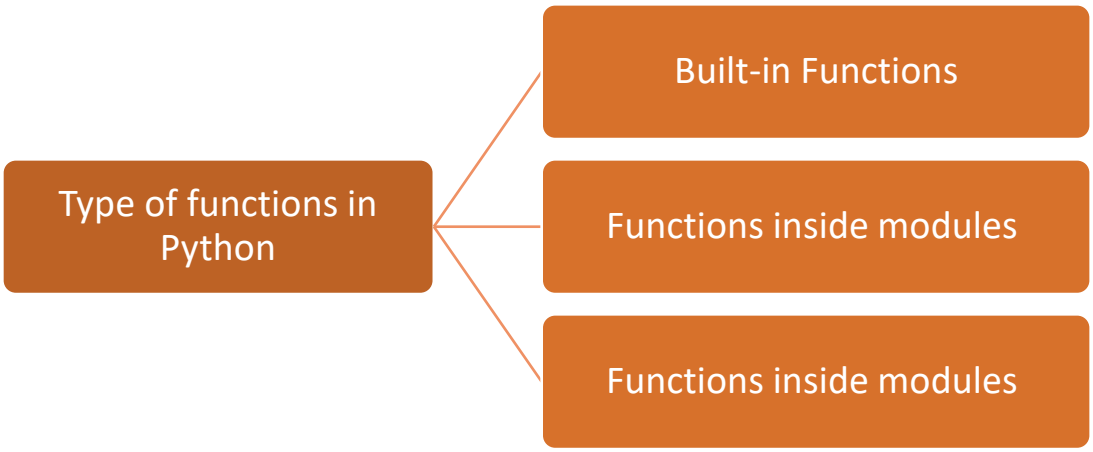
Complete Program

Defining and calling function in Python

```
def message():
    print("Hello Python")
def shownumber(num):
print(num)
message()
shownumber(10)
```

Output

```
Hello Python
10
```



1. Built-in Functions in Python

They are predefined functions provided by Python. They are also known as library functions. These functions are basically used to perform basic input/output operations and many other operations which can't be performed by any user defined function.

Built in functions can be called in any program The most commonly used built in functions are input(), id(), type(), int(), float(), bool(), len()etc.

2. Functions inside modules in Python

These functions are defined within modules and can be used only when we import a particular module in our program.

For example, to use pre-defined functions sqrt(), we need to import the module **math** as it contains definition of sqrt() function in it.

3. User defined function in Python

It is a function defined by a programmer depending upon his own requirement. User defined functions are mainly defined for those operations which are repeatedly required in a program.

A user defined function is a self-contained group of statements which can perform a specific task. Once a user defined function has been defined, it can be called within another function in the same program or another program.

A user defined function can also take values from other functions and can also return some value.

There are two terms associated with functions:

Calling function: The function which calls another function within its body is known as calling function.

Called function: The function which is called by another function is known as called function. It is the called function which contains the actual set of statements which would be executed when function call is made.

Types of user defined function in Python

There are four types of user defined function in Python

(i) Function without parameters and without return value

This type of function has no parameters so it doesn't receive any values from the calling function. So In we don't specify anything within the parenthesis written after the function name. This type of function doesn't contain return statement.

Features:

- Function contains no parameters.
- Calling function does not receive any value from the function.
- There is simple passing of control from calling function to called function and vice versa but there is no passing of values between them.
- Everything is performed within the body of function.

Program of function without parameters and without return value.

```
def sum():
    a=int(input("Enter first number="))
    b=int(input("Enter second number="))
    c=a+b
    print ("sum=",c)
```

```
sum()   #Function sum() is called here
```

Output

```
Enter first number=2
Enter second number=3
```

```
sum= 5
```

Description of Above Program

In the above program, we have defined a user define function named sum() which reads values of two variables, adds them and shows their addition. This function has been called as sum().

(ii) Function with parameters and without return value

In this type of function, we need to specify parameters within the parenthesis written after the function name. parameter are generally variables declared individually within the parenthesis.

Parameters within the parenthesis while defining a function are known as formal parameters. Parameter within the parenthesis while calling the user defined function are known as actual parameters.

The formal parameters receive values from actual parameters. This function doesn't return any value so we don't write return statement.

Features:

- Function contains parameters.
- Calling function does not receive any value from the called function.
- There is passing of values from calling function to user defined function.
- **Input is received from calling function and only manipulation is performed within the user defined function.**

Program of function with parameters and without return value.

```
def sum(a,b):
    c=a+b
    print ("sum=",c)

sum(10,5)
```

Output

```
sum= 15
```

Description of Above Program

In the above program, we have defined a user defined function named sum() which has two formal parameters namely a and b. It performs addition of a and b and shows their addition.

This function has been called as sum(10,5). 10 and 5 are actual parameters. 10 will go to a and 5 will be go to b so their sum 15 will be shown in the output.

(iii) Function with parameters and with return value

In this category of function, we need to specify formal parameters within the parenthesis written after the function name.

In this type of function, input is generally received from calling function and some manipulation is performed within the user defined function and result of manipulation is returned back to the calling function.

While calling such type of function, we need to assign function calling statement to some variable or we can directly write function calling statement within the print() function.

Features:

- Function contains parameters.
- We must need to write return statement.
- There is a two-way communication of values between calling function as well as called function i.e. values are transferred from calling function to called function and again some value is returned back to the calling function.
- At a time only one value can be returned by this type of function.
- Input is received from calling function and after manipulation within the user defined function, result is returned back to the calling function.

Program of function with parameters and with return value.

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

s=sum(3,5)
print("sum of 3 and 5 is",s)
print("sum of 5 and 6 is ",sum(5,6))
```

Output

```
sum of 3 and 5 is 8
sum of 5 and 6 is 11
```

Description of Above Program

In the above program, we have defined a user defined function named sum() which has two formal parameters namely a and b. It performs addition of a and b and returns their addition by statement return(c).

This function has been called twice. Firstly as **s=sum(3,5)**. 3 goes to a and 5 goes to b so their sum 8 is stored in variable c which will be returned and be saved in variable s.

Function **sum()** is called again as **print("sum of 5 and 6 is ",sum(5,6))** within print() function. 5 and 6 go to parameters a and b respectively. Their sum is returned and shown as 11

(iv) Function without parameters and with return value

This category of function doesn't contain any parameter. Manipulations are performed within the body of user defined function and result of manipulation is returned back to the calling function.

There is one way communication in this function i.e. value is transferred from called function to calling function. While calling such type of function, we need to assign function calling statement to some variable or we can directly call function within the print function.

Features:

- Function doesn't contain parameter.
- There is a one way communication of values between called function and calling function i.e. values are transferred from called function to calling function.
- All manipulations are performed within the user defined function and result is returned back to the calling function.

Program of function without parameters and with return value.

```
def sum():
    a=int(input("Enter first value="))
    b=int(input("Enter second value="))
    c=a+b
    return(c)

s=sum()
print("Sum=",s)
print("Sum=",sum())
```

Output

```
Enter first value=2
Enter second value=3
Sum= 5
Enter first value=4
Enter second value=5
Sum= 9
```

Description of Above Program

In the above program, we have defined a user defined function named sum() It performs addition two values after reading them and result is returned back by statement return(c).

This function has been called twice. Firstly as **s=sum()**. Values of variables a and b are input and their sum is stored in variable c which will be returned and be saved in variable s.

Function **sum()** is called again as **print("sum =",sum())** within print() function. Values of variables a and b are input again and their sum is stored in variable c which will be returned and shown.

Types of function parameters in Python

In Python, we can take four different types of parameters in a user defined function. They are as follows:

1. Positional parameters in Python

A positional parameter is any parameter that's not supplied as a **key=value** pair.

Value of first actual parameter goes to first formal parameter, second actual parameter goes to second formal parameter and so on as per their positions.
In the following program, formal parameter a gets 25 and formal parameter b gets 10.

Program to demonstrate positional parameters

```
def show(a,b):
    print(a,b)

show(25,10)

#a gets 25 and b gets 10
```

Output

```
25 10
```

---

2. Passing string as function parameter in Python

A string can be passed as a parameter to a Python function as like other parameters. We simplay need to pass string value as actual parameter while calling the function.
When we pass a string as argument, it is automatically passed by value. Strings are immutable so no changes can be made to formal parameter.

Program of function with string as parameter.

```
def show(var1):
    print(var1)
show('Amit')
```

"' Amit' will be stored in formal parameter var1 and shown within function body."

Output

```
Amit
```

---

3. Passing list as function parameter in Python

A list can also be passed as a parameter to a Python function. We simply need to specify the name of list both as actual as well as formal parameter. There is no need to specify brackets. When we pass a list as an argument, it is automatically passed by reference. Lists are mutable so changes be made to elements of list in function body will direct affect actual parameters.

Example 1 – Program of function with list as parameter.

```
def show(list1):
    print(list1)

L1=[10,20,30,40]
show(L1)
```

Output

```
[10, 20, 30, 40]
```

In the above program, **L1** is the list passed as actual argument to function **show()**, **list1** is the formal parameter w receive values of

Example 2- Program of function with list as parameter.

```
def show(list1):
    list1[0]='a'

L1=[10,20,30,40]
show(L1)
print(L1)
```

Output

```
['a', 20, 30, 40]
```

In the above program, L1 is the list passed as actual argument to function show(), list1 is the formal paramet will receive values of L1.
First element of list1 i.e. list1[0] is assignmed 'a'.
After the function call statement show(L1)
When we use print(L1) statement. It will display modified value of first element of list i..e changes list1 are reflected back to list L1.

---

4. Passing tuple as function parameter in Python

A tuple can also be passed as a parameter to a Python function. We simply need to specify the name of tuple both as actual as well as formal parameter. There is no need to specify parenthesis.

Tuples are immutable so we can't make changes to tuple in formal argument.

Example – Program of function with tuple as parameter.

```
def show(tuple1):
    print(list1)

T1=[10,20,30,40]
show(T1)
```

Output

```
[10, 20, 30, 40]
```

In the above program, T1 is the list passed as actual argument to function show(), tuple1 is the formal paramet will receive values of T1

---

5. Passing dictionary as function parameter in Python

A dictionary can also be passed as a parameter to a Python function. We simply need to specify the name of dictionary both as actual as well as formal parameter. There is no need to specify braces.

When we pass a list as an argument, it is automatically passed by reference. Lists are mutable so changes be made to elements of list in function body will direct affect actual parameters.

Program of function with list as parameter.

```
def show(dict1):
print(list1)D1={1:'a',2:'b',3:'c'}
print(D1)
```

Output

```
{1: 'a', 2: 'b', 3: 'c'}
```

In the above program, dictionary D1 is passed as actual argument to function show(), dict1 is the formal para which will receive values of D1.

6. Default parameters in Python

In default parameters, we can assign some value to the formal parameters of a function but the assignment of values is possible from last argument of the function towards left.

The main use of default parameters is that we may or may not provide value for default parameter while calling the function because function will automatically take the default value of formal parameter in the function.

Example: **def show(a=5,b=10)**

In the above example, a is a default parameter having value 5 and b is also a default parameter having value 10.

Example-1 : Program of function without default parameters.

```
def show(a=5,b=10):
    print(a,b)

show() #a takes 5, b takes 10
show(25) #a takes 25, b takes 10
show(25,40) #a takes 25, b takes 40
```

Output

```
5 10
25 10
25 40
```

Example-2 Program of function without default parameters.

```
def display(fname,lname="Kumar"):
    print(fname,lname)

display("amit") #fname gets 'amit', lname gets 'Kumar'
display("amit","singh") #fname gets 'amit', lname gets 'singh'
```

Output

```
amit Kumar
amit singh
```

---

7. Required parameters in Python

In case of Required parameters, number and order of actual parameters and formal parameters must be same. The values of actual parameters are assigned to formal parameters on one-to-one basis i.e. First actual parameter in assigned to first formal parameter, second actual parameter in assigned to second formal parameter and so on.

Program of function with Required parameters.

```
def show(name,salary):
    print(name,salary)

show('Amit',50000)
```

Output

```
Amit 50000
```

---

8. Keyword arguments in Python

The keyword parameters are used to specify the names of formal parameters while calling the function. The main advantage of keywords parameters is that we can write actual parameters in any order we want.

Values of actual parameters will be passed to formal parameters on the basis of their names rather than their order.

Program of function with Keyword parameters.

```
def show(name,age,salary):
    print("name=",name)
    print("age=",age)
    print("salary=",salary)

show(age=30,salary=50000,name='Amit')
```

"'value of age will be 30 in formal parameter age,
name will receive 'Amit' and
salary will recieve 50000."'

Output

```
name= Amit
age= 30
salary= 50000
```

---

9. Variable number of parameters in Python

This technique of passing parameters to a function is very useful when we do not know the exact number of parameters that will be passed to a function. Syntax of function with variable number of parameters is:

def func_name(var1, *vartuple)

Here var1 takes very first parameter value. All parameter values passed after first parameter will be stored in *vartuple in the form of a tuple. We can perform operations on these values just as we work with tuples.

Program of function with variable number of parameters.

```
def show(var1,*var2):
```

```
    print(num1)
    print(var2)
show('Amit',1,3,4,5,6)
```

'Amit' will be stored in argument var1. Values 1,3,4,5,6 will be stored as a tuple in var2.

Output

```
Amit
(1, 3, 4, 5, 6)
```

Python Modules and Packages

Python module is defined as a collection of functions, variables, constants, classes, objects along with Python statements.

Structure of a Python Module

A Python module is a normal Python file (.py file) that can contain following things:

(i) Docstring

Docstring is text written within the pair of triple quotes. it is basically used for documentation purpose. Docstring should be the first string stored inside a module, function or class.

General conventions for Doctstring are:

1. First letter of first line is a capital letter.
2. Second line is blank line.
3. Rest of the details begin from third line.

(ii) Variables and constants

We can also define variables and constants inside a Python module depending upon the requirement.

(iii) Class

We can also create more than class inside a Python module. Class may be defined as blueprint for creating objects.

(iv) Object

Object is anything having properties and behaviour. But in Python, object is just a variable of class type which canbe used to refer to members of a class. It is also known as instance of class.

(v) Statements

We can write all valid statements of Python within a module.

(vi) Functions

Function is defined as named group of instructions. When we call function, all statements written inside the function body get executed.

Types of Module in Python

There are two types of modules in Python

- Predefined modules
- User Defined Modules

(i) Predefined Modules in Python

Predefined modules are also known as library modules. They contain library functions and classes provided by Python. These modules are used to perform basic complex mathematical operations and many other operations which can't be performed by any user defined modules. We can import predefined modules in any Python program.

(ii) User Defined Modules in Python

User Defined Modules are defined by a programmer depending upon his own requirement. Once a user defined module is defined, it can be called within another Python module or program.

Example

Module1.py

```
'''
Name: Demonstration of Python modules
What it contains: It contains two functions namely sum() and subtract()
'''
def sum(a,b):
return(a+b)

def subtract(a,b):
return(a-b)
```

In above example a user defined module named **Module1.py** has been created. We can see documentation of above program by using following command at python shell.

1. import Module1 #to import module

2. help(Module1) #To see documentation of module

```
>>> import Module1
>>> help(Module1)
Help on module Module1:
NAME
Module1
FILE
c:\python30\Module1.py
DESCRIPTION
Name: Demonstration of Python modules
What it contains: It contains two functions namely sum() and subtract()
FUNCTIONS
subtract(a, b)
sum(a, b)
```

We can also use predefined function dir() to see list of elements defined inside a module as follows:

```
>>>import Module1

>>> dir(Module1)
['__builtins__', '__doc__', '__file__', '__name__', '__package__', 'subtract', 'sum']
```

Importing Modules in a Python Program

Before using functions and other components of a module in a Python program, we need to import the module in our program.

We can use **import** statement to import modules in a program. The import statement can be used in two ways:

- To import entire module:
- To import selected objects

(i) To import entire Python module

We can use **import<module-name>**command to import entire module. The **import** statement internally executes the code written inside module file and then makes it available to our program. Syntax to import entire module is:

import module1[, module2[,…module3]]

module1,module2,module3 refer to names of different modules that we want to import in our program.

Example1:

To import predefined module **math** in a Python program we can use **import math** statement in a Python program.

Example2:

To import user defined module **Module1** in a Python program we can use **import Module1** statement in a Python program.

Example3:

To import predefined modules **math** and **random** in a Python program we can use **import math.randome** statement in a Python program.

After importing a module, you can use any function or other element of the imported module as per following syntax:

<module-name>.<function-name>()

Example1:

To call **sqrt()** function defined in module **math,** we can write **math.sqrt()** statement in Python program.

Example2:

To call **sum()** function defined in module **Module1,** we can write **Module1.sum()** in Python program.

(ii) To Imported Select Objects from a Python module

If you want to import some selected items, not all from a module, then you can use from <module> import statement as per following syntax:

from<module>import<objectname>[,<objectname>[…]|*

To import a single object from a module we don't have to prefix the module's name, we can write the name of object after keyword import.

For example to import just the constant **pi** from module **math,** you can write:

from module math import pi

Now, we can use the constant **pi.** We need not prefix it with module name. We can simply write **print(pi)** statement at Python shell to get value of pi.

To import multiple objects from a module we don't have to prefix the module's name, we can write the comma separated list of objects after keyword import. For examle, to import functions **sqrt()** and **pow()** from **math** module, we need to write **from module math import sqrt, pow**

To import all the items from a module we don't have to prefix the module's name, we can write:

from<modulename>import *

To import all the items in module **math,**we can write:

from math import *

Now, we can use all the functions, variables etc from **math** module, without having to prefix module's name to the imported item name.

Example 1

Program to import and use module Module1.py in program.

```
import Module1
print(Module1.sum(10,5))
print(Module1.subtract(10,5))
```

Output

```
15

5
```

Example 2

Program to import and use math module in program.

```
import math
print(math.sqrt(9))
print(math.pow(2,3))
```

Output

```
3.0
8.0
```

Example 3

Program to import and use specific functions of math module.

```
from math import sqrt,pow
print(sqrt(9)) #No need to prefix math
print(pow(2,3))
```

Output

```
3.0
8.0
```

Example 4

Program to import all functions of module math.

```
from math import *
print(sqrt(9)) #No need to prefix math
print(pow(2,3))
```

Output

```
3.0
8.0
```

Processing of Import <module> Command

When we use **import <module>** command, internally following things take place:

- The code of imported module is interpreted and executed.
- Defined functions and variables created in the module are now available to the program that imported module.
- For imported module, a new namespace is setup with the same name as that of the module.

Namespace in Python is a named environment containing logical grouping of related objects. For every module(.py file), Python creates a namespace having its name similar to that of module's name.

For example after importing module Module1 in our program ,all objects of module Module1 would be referred as Module1.<object-name>.

Example: if Module1 has a function defined as sum(), then it would be referred to as Module1.sum() in our program.

When we issue **from <module> import <object>** command, internally following things take place:

- The code of imported module is interpreted and executed.
- Only the asked functions and variables from the module are made available to the program.
- No new namespace is created, the imported definition is just added in the current namespace.
- That means if your program already has a variable with the same name as the one imported via module, then the variable in your program will hide imported member with same name because there cannot be two variables with the same name in one namespace. Following code fragment illustrates this.