CH 6: Python Functions

In the previous section, we discussed about loop control and decision control. When you write a program, you need to do some task multiple times – which creates repetition of code. Instead of writing same code again and again, we can write a function, and call the function multiple times as required. This makes the code more readable, reusable and maintainable.

A function can be defined as a block of organized, reusable code that is used to perform a single, coherent action. Functions provide better modularity for your application and a high degree of code reusing. There are two kind of functions in python:

1. Build-in function – Like print, which python provide
2. User-defined function – user wrote this fuction as required.

In python, the syntax of a function is as below:

def functionname( parameters ):

"function\_docstring"

function\_suite

return [expression]

A python function consists of:

1. Function blocks begin with the keyword **def** followed by the function name and parentheses (( ) ).
2. Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
3. The first statement of a function can be an optional statement - the documentation string of the function or *docstring*.
4. The code block within every function starts with a colon (:) and is indented.
5. The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

An example of Adding two numbers is as below:

def addTwoNum(x,y): # This is a function Defination

""" Take two number and return the sum""" # This is a Doc string

z = x +y # This is Function body

return z # this is return statement

>>> print addTwoNum(1,2) # this is a function call

3

>>> print addTwoNum(8,19) # This is another function call

27

We can have multiple type of function depends on parameter type or return type, discussed as below:

**6.1 Required Argument:** Required arguments are the arguments passed to a function in correct positional order. Here, the number of arguments in the function call should match exactly with the function definition. In the example below, both x and y are required. The function call should exactly pass 2 parameters. Passing more than 2 or less than 2, will raise error.

def foo(x,y):

print x,y

>>> foo(1,2) # it;’s OK

1 2

>>> foo(1)

Traceback (most recent call last):

File "<pyshell#3>", line 1, in <module>

foo(1)

**TypeError: foo() takes exactly 2 arguments (1 given)**

>>> foo()

Traceback (most recent call last):

File "<pyshell#4>", line 1, in <module>

foo()

**TypeError: foo() takes exactly 2 arguments (0 given)**

>>> foo(1,2,3)

Traceback (most recent call last):

File "<pyshell#5>", line 1, in <module>

foo(1,2,3)

**TypeError: foo() takes exactly 2 arguments (3 given)**

>>>

**6.2 Default arguments**: A default argument is an argument that assumes a default value if any value is not provided while function call for that argument. For example, the following function take two argument x and y, but both has a default value 10 and 20 respectively. Thus if we provide less parameters while function call, default value will be taken for that parameters:

**def foo(x = 10,y = 20):**

**print x,y**

**>>> foo() # No parameter is passed, hence x and y both take default**

**10 20**

**>>> foo(30) # one parameter passed, hence x takes 30 and y take default value**

**30 20**

**>>> foo(30,40) # both parameters passed , hence both take passing value.**

**30 40**

**>>>**

**6.3 Keyword arguments:** Here the argument are named when we do a function call, it allow us to pass the parameter in different order as it is specified in function definition. In the below example, we are passing y first and x second still in function definition, y is followed by x.

**def foo(x,y):**

**print 'x: ',x,'y: ',y**

OutPut :

**>>> foo(x=10,y=30) # it in same order as in function definition**

**x: 10 y: 30**

**>>> foo(y=30,x=10) # here y is passed first, and x is passed**

**x: 10 y: 30**

**6.4. Variable-length arguments:** Sometime it requires passing variable length argent as the number of argument is not known. Python provide a way to pass variable length argument.

The function takes two formal arguments, (required),and a variable length argument and print them:

def foo(x,y,\*args):

print 'x: ',x,'y: ',y

for a in args:

print 'Extra data: ',a

>>> foo(10,20) # pass formal argument only

x: 10 y: 20

>>> foo(10,20,30) # pass one Extra argumnets

x: 10 y: 20

Extra data: 30

>>> foo (10,20,30,40,50,60) # pass 4 extra argumnets

x: 10 y: 20

Extra data: 30

Extra data: 40

Extra data: 50

Extra data: 60

There are two kind of variable length arguments:

* Non-keyword variable length argument- Denoted as \*args
* Keyword-Variable length argument – denoted as \*\*kwargs

An example of keyword-variable length argument is as below:

def foo(x,y,\*\*kwargs):

print 'x: ',x,'y: ',y

for key in kwargs:

print 'Extra data: ',key,' = ',kwargs[key]

Output

---------

>>> foo(10,30) # only formal

x: 10 y: 30

>>> foo(10,30,z=40) # one named argument z

x: 10 y: 30

Extra data: z = 40

>>> foo(10,20,z=30,w=40,u=50) # three named argunet z, w, u

x: 10 y: 20

Extra data: z = 30

Extra data: u = 50

Extra data: w = 40

We can have all of these 3 kinds of arguments at a same time: formal, no-keyword and keyword. Example below shows this:

def foo(x,y,\*args, \*\*kwargs): # It take 2 formal, non-kewd ad key-word args

print 'x: ',x,'y: ',y

for a in args:

print 'Not Named data: ',a

for key in kwargs:

print 'Named data: ',key,' = ',kwargs[key]

print args,kwargs

>>> foo(1,2) # No non-kewd and key-word args

x: 1 y: 2

() {}

>>> foo(1,2,3,4) # two formal + 2 non-keyword

x: 1 y: 2

Not Named data: 3

Not Named data: 4

(3, 4) {}

>>> foo(1,2,3,4,a=5,b=6) # two formal + two non-keyword + two keyword

x: 1 y: 2

Not Named data: 3

Not Named data: 4

Named data: a = 5

Named data: b = 6

(3, 4) {'a': 5, 'b': 6}

>>> foo(a=5,b=6,1,2,3,4) # This is invalid.

SyntaxError: non-keyword arg after keyword arg

>>>

As you can see, all non-keyword args are passed by tuple and all keyword-argument is passed as a python dict.

6.5 **Packing** and **Unpacking Arguments**: In previous example, variable number of arguments can be packed in a tuple or a dict based on whether they are keyword or not.

We create a list and pass to the function( by putting \* before list name)- this is called packing data. Inside the function, we do a unpacking the list to get the data.

def foo(\*a):

for i in a: # inside the function we are unpacking the data.

print i,

>>> x = [1,2,3,4,5] <= We have list

>>> foo(\*x) # We packed in \*x and passes to the function

1 2 3 4 5

>>>

**6.6. Lambda function: You** can use the lambda keyword to create small anonymous functions. These functions are called anonymous because they are not declared in the standard manner by using the def keyword.

The Syntax is: It is just a single line statement

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

>>> mult = lambda arg1,arg2 : arg1 \* arg2 # Define a lambda function having two argumnet and return the multiplication

>>> print '10 \* 11 =',mult(10,11) # call lambda func similar as normal function

10 \* 11 = 110

>>> print 'Mult of 8 and 9 is :',mult(8,9)

Mult of 8 and 9 is : 72

>>> >>> mult

<function <lambda> at 0x02AB31B0> # It will print lambda function

>>>

**6.7 Return Multiple Values from a function:** In python it is possible to return multiple values from a function. The following function takes two arguments and returns the addition and multiplication.

>>> def foo(x,y):

return x+y,x\*y

>>> print 'Sum and Mult of 2 and 3 is ',foo(2,3)

Sum and Mult of 2 and 3 is (5, 6)

>>> s,m = foo(10,11)

>>> s,m

(21, 110)

>>>

**6.8 Passing by Value and passing via References:**  In python, we can do passing by value and pass by reference, In pass by value got passed, thus any alternation inside the function will not affect in outside, but in pass by referees any changes made inside the function will effect outside.

We will discuss more on this in next chapter.