Unit II

CHAPTER 7

PYTHON FUNCTIONS

Learning Objectives

After studying this chapter, students will be able to:

- Understand the concept of function and their types.
- Know the difference between User defined and Built in functions.
- Know how to call a function.
- Understand the function arguments.
- Know Anonymous functions.
- Know Mathematical and some String functions.

7.1 Introduction

Functions are named blocks of code that are designed to do specific job. When you want to perform a particular task that you have defined in a function, you call the name of the function responsible for it. If you need to perform that task multiple times throughout your program, you don't need to type all the code for the same task again and again; you just call the function dedicated to handling that task, and the call tells Python to run the code inside the function. You'll find that using functions makes your programs easier to write, read, test, and fix errors.

Main advantages of functions are

- *It avoids repetition and makes high degree of code reusing.*
- *It provides better modularity for your application.*

Note

Functions are nothing but a group of related statements that perform a specific task.

7.1.1 Types of Functions

Basically, we can divide functions into the following types:

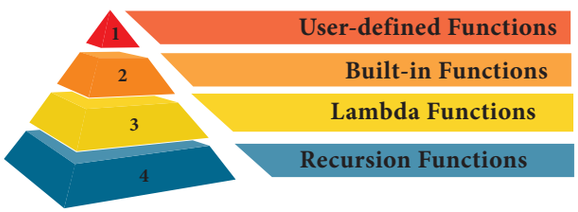


Figure – 7.1 – Types of Python Functions

Functions	Description
User-defined functions	Functions defined by the users themselves.
Built-in functions	Functions that are inbuilt with in Python.
Lambda functions	Functions that are anonymous un-named function.

Recursion functions	Functions that calls itself is known as recursive.

Table – 7.1 – Python Functions and it's Description

7.2 Defining Functions

Functions must be defined, to create and use certain functionality. There are many built-in functions that comes with the language python (for instance, the print() function), but you can also define your own function. When defining functions there are multiple things that need to be noted;

- Function blocks begin with the keyword **"def"** followed by function name and parenthesis ().
- Any input parameters or arguments should be placed within these parentheses when you define a function.
- The code block always comes after a colon (:) and is indented.
- The statement **"return [expression]"** exits a function, optionally passing back an expression to the caller. A **"return"** with no arguments is the same as return None.

Note

Python keywords should not be used as function name.

7.2.1 Syntax for User defined function

```
def <function_name ([parameter1, parameter2…] )> :
        <Block of Statements>
        return <expression / None>
```

Note

In the above Syntax, the Text which is given in square bracket [] is optional.

Block:

A block is *one or more lines of code*, grouped together so that they are treated as one big sequence of statements while execution. In Python, statements in a block are written with *indentation*. Usually, a block begins when a line is indented (by four spaces) and all the statements of the block should be at same indent level.

Nested Block:

A block within a block is called nested block. When the first block statement is indented by a single tab space, the second block of statement is indented by double tab spaces.

Here is an example of defining a function;

```
def Do_Something( ):
value =1        #Assignment Statement
return value    #Return Statement
```

Now let's check out functions in action so you can visually see how they work within a program. Here is an example for a simple function to display the given string.

Example:

```
def hello():
        print ("hello - Python")
        return
```

7.2.2 Advantages of User-defined Functions

1. Functions help us to divide a program into modules. This makes the code easier to manage.
2. It implements code reuse. Every time you need to execute a sequence of statements, all you need to do is to call the function.
3. Functions, allows us to change functionality easily, and different programmers can work on different functions.

7.3 Calling a Function

To call the hello() function from *example 7.2-1*, you use this code:

When you call the "hello()" function, the program displays the following string as output:

Output

hello – Python

Alternatively we can call the "hello()" function within the print() function as in the example given below.

Example:

```
def hello():
        print ("hello - Python")
        return
print (hello())
```

If the return has no argument, **"None"** will be displayed as the last statement of the output.

The above function will output the following.

Output:

hello – Python
None

7.4 Passing Parameters in Functions

Parameters or arguments can be passed to functions

def function_name (parameter(s) separated by comma):

Let us see the use of parameters while defining functions. The parameters that you place in the parenthesis will be used by the function itself. You can pass all sorts of data to the functions. Here is an example program that defines a function that helps to pass parameters into the function.

Example:

```
# assume w = 3 and h = 5
        def area(w,h):
                return w * h
        print (area (3,5))
```

The above code assigns the width and height values to the parameters **w** and **h**. These parameters are used in the creation of the function "area". When you call the above function, it returns the product of width and height as output.

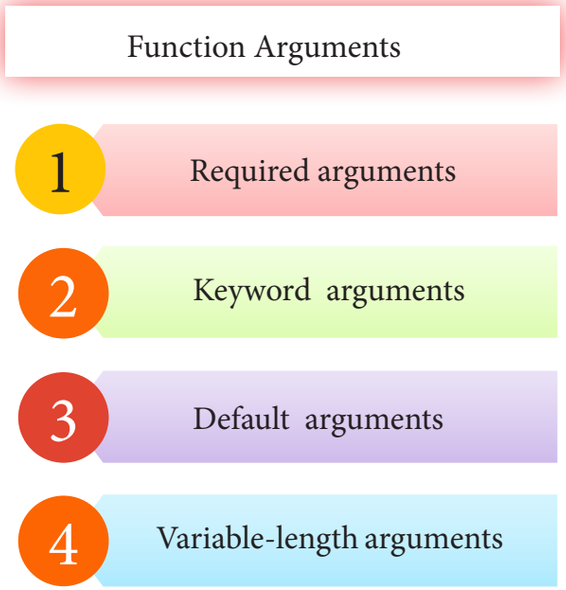
The value of 3 and 5 are passed to **w** and **h** respectively, the function will return 15 as output.

DO YOU KNOW?

We often use the terms parameters and arguments interchangeably. However, there is a slight difference between them. Parameters are the variables used in the function definition whereas arguments are the values we pass to the function parameters

7.5 Function Arguments

Arguments are used to call a function and there are primarily 4 types of functions that one can use: *Required arguments, Keyword arguments, Default arguments and Variable-length arguments.*



7.5.1 Required Arguments

"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. You need atleast one parameter to prevent syntax errors to get the required output.

Example :

```
def printstring(str):
        print ("Example - Required  arguments ")
        print (str)
        return
# Now you can call printstring() function
printstring()
```

When the above code is executed, it produces the following error.

```
Traceback (most recent call last):
File "Req-arg.py", line 10, in <module>
printstring()
    TypeError: printstring() missing 1 required positional argument: 'str'
```

Instead of **printstring()** in the above code if we use **printstring ("Welcome")** then the output is

Output:

Example - Required arguments
Welcome

7.5.2 Keyword Arguments

Keyword arguments will invoke the function after the parameters are recognized by their parameter names. The value of the keyword argument is matched with the parameter name and so, one can also put arguments in improper order (not in order).

Example:

```
def printdata (name):
        print ("Example-1 Keyword arguments")
        print ("Name :",name)
        return
# Now you can call printdata() function
printdata(name = "Gshan")
```

When the above code is executed, it produces the following output :

Output:

Example-1 Keyword arguments
Name :Gshan

Example:

```
def printdata (name):
        print ("Example-2 Keyword arguments")
        print ("Name :", name)
        return
# Now you can call printdata() function
printdata (name1 = "Gshan")
```

When the above code is executed, it produces the following result :

TypeError: printdata() got an unexpected keyword argument 'name1'

Example:

```
def printdata (name, age):
        print ("Example-3 Keyword arguments")
        print ("Name :",name)
        print ("Age :",age)
        return
# Now you can call printdata()  function
printdata (age=25, name="Gshan")
```

When the above code is executed, it produces the following result:

Output:

Example-3 Keyword arguments
Name : Gshan
Age : 25

Note

In the above program the parameters orders are changed

7.5.3 Default Arguments

In Python the default argument is an argument that takes a default value if no value is provided in the function call. The following example uses default arguments, that prints default salary when no argument is passed.

Example:

```
def printinfo( name, salary = 3500):
        print ("Name: ", name)
        print ("Salary: ", salary)
        return
printinfo("Mani")
```

When the above code is executed, it produces the following output

Output:

Name: Mani
Salary: 3500

When the above code is changed as print info("Ram",2000) it produces the following output:

Output:

Name: Ram
Salary: 2000

In the above code, the value 2000 is passed to the argument salary, the default value already assigned for salary is simply ignored.

7.5.4 Variable-Length Arguments

In some instances you might need to pass more arguments than have already been specified. Going back to the function to redefine it can be a tedious process. Variable-Length arguments can be used instead. These are not specified in the function's definition and an asterisk (*) is used to define such arguments.

Lets see what happens when we pass more than 3 arguments in the sum() function.

Example:

```
def sum(x,y,z):
        print("sum of three nos :",x+y+z)
sum(5,10,15,20,25)
```

When the above code is executed, it produces the following **result** :

TypeError: sum() takes 3 positional arguments but 5 were given

7.5.4.1 Syntax - Variable-Length Arguments

```
def function_name(*args):
        function_body
        return_statement
```

Example:

```
def printnos (*nos):
        for n in nos:
        print(n)
        return
# now invoking the printnos() function
print ('Printing two values')
printnos (1,2)
print ('Printing three values')
printnos (10,20,30)
```

Output:

```
Printing two values
        1
        2
Printing three values
        10
        20
        30
```

Evaluate Yourself

In the above program change the function name printnos as printnames in all places wherever it is used and give the appropriate data Ex. printnos (10, 20, 30) as printnames ('mala', 'kala', 'bala') and see output.

In Variable Length arguments we can pass the arguments using two methods.

1. Non keyword variable arguments
2. Keyword variable arguments

Non-keyword variable arguments are called **tuples**. You will learn more about tuples in the later chapters. The Program given is an illustration for non keyword variable argument.

Note

Keyword variable arguments are beyond the scope of this book.

The Python's print() function is itself an example of such a function which supports variable length arguments.

7.6 Anonymous Functions

What is anonymous function?

In Python, anonymous function is a function that is defined without a name. While normal functions are defined using the **def** keyword, in Python anonymous functions are defined using the **lambda** keyword. Hence, anonymous functions are also called as **lambda** functions.

What is the use of lambda or anonymous function?

- Lambda function is mostly used for creating small and one-time anonymous function.
- Lambda functions are mainly used in combination with the functions like filter(), map() and reduce().

Note

filter(), map() and reduce() functions are beyond the scope of this book.

Lambda function can take any number of arguments and must return one value in the form of an expression. Lambda function can only access global variables and variables in its parameter list.

7.6.1 Syntax of Anonymous Functions

The syntax for anonymous functions is as follows:

lambda [argument(s)] :expression

Example:

```
sum = lambda arg1, arg2: arg1 + arg2
print ('The Sum is :', sum(30,40))
print ('The Sum is :', sum(-30,40))
```

Output:

```
The Sum is : 70
The Sum is : 10
```

The above lambda function that adds argument ***arg1*** with argument ***arg2*** and stores the result in the variable sum. The result is displayed using the print().

7.7 The return Statement

- The return statement causes your function to exit and returns a value to its caller. The point of functions in general is to take inputs and return something.
- The return statement is used when a function is ready to return a value to its caller. So, only one return statement is executed at run time even though the function contains multiple return statements.
- Any number of 'return' statements are allowed in a function definition but only one of them is executed at run time.

7.7.1 Syntax of return

return [expression list]

This statement can contain expression which gets evaluated and the value is returned. If there is no expression in the statement or the return statement itself is not present inside a function, then the function will return the None object.

Example :

```
# return statment
def usr_abs (n):
        if n>=0:
                return n
        else:
                return –n
# Now invoking the function
x=int (input("Enter a number :")
print (usr_abs (x))
```

Output 1:

```
Enter a number : 25
        25
```

Output 2:

```
Enter a number : -25
        25
```

7.8 Scope of Variables

Scope of variable refers to the part of the program, where it is accessible, i.e., area where you can refer (use) it. We can say that scope holds the current set of variables and their values. We will study two types of scopes - **local scope** and **global scope**.

7.8.1 Local Scope

A variable declared inside the function's body is known as local variable.

Rules of local variable

- A variable with local scope can be accessed only within the function that it is created in.
- When a variable is created inside the function the variable becomes local to it.
- A local variable only exists while the function is executing.
- The formal parameters are also local to function.

Example : Create a Local Variable

```
def loc():
        y=0 # local scope
        print(y)
loc()
```

Output:

```
0
```

Example : Accessing local variable outside the scope

```
def loc():
        y = "local"
loc()
print(y)
```

When we run the above code, the output shows the following error:

The above error occurs because we are trying to access a local variable **'y'** in a global scope.

NameError: name 'y' is not defined

7.8.2 Global Scope

A variable, with global scope can be used anywhere in the program. It can be created by defining a variable outside the scope of any function/block.

Rules of global Keyword

The basic rules for ***global*** keyword in Python are:

- When we define a variable outside a function, it's global by default. You don't have to use global keyword.
- We use global keyword to read and write a global variable inside a function.
- Use of global keyword outside a function has no effect

Use of global Keyword

Example : Accessing global Variable From Inside a Function

```
c = 1           # global variable
def add():
        print(c)
add()
```

Output:

```
1
```

Example : Accessing global Variable From Inside a Function

```
c = 1            # global variable
def add():
        print(c)
        add()
```

Output:

```
1
```

Example : Modifying Global Variable From Inside the Function

```
c = 1            # global variable
def add():
        c = c + 2 # increment c by 2
        print(c)
add()
```

Output:

```
UnboundLocal Error: local variable 'c' referenced before assignment
```

Note

Without using the global keyword we cannot modify the global variable inside the function but we can only access the global variable.

Example : Changing Global Variable From Inside a Function using global keyword

```
x = 0                        # global variable
def add():
        global x
        x = x + 5            # increment by 5
        print ("Inside add() function  x value is :", x)
add()
print ("In main x value is :", x)
```

Output:

```
Inside add() function  x value is : 5
In main x value is : 5
```

In the above program, **x** is defined as a **global** variable. Inside the **add()** function, **global** keyword is used for **x** and we increment the variable **x** by **5**. Now We can see the change on the **global** variable **x** outside the function i.e the value of **x** is **5**.

7.8.3 Global and local variables

Here, we will show how to use global variables and local variables in the same code.

Example : Using Global and Local variables in same code

```
x=8                    # x is a global variable
def loc():
        global x
        y = "local"
        x = x * 2
        print(x)
        print(y)
loc()
```

Output:

```
16
local
```

In the above program, we declare ***x*** as global and ***y*** as local variable in the function **loc()**.

After calling the function **loc()**, the value of ***x*** becomes 16 because we used **x=x * 2.** After that, we print the value of local variable ***y*** i.e. local.

Example : Global variable and Local variable with same name

```
x = 5
def loc():
        x = 10
        print ("local x:", x)
loc()
print ("global x:", x)
```

Output:

```
local x: 10
global x: 5
```

In above code, we used same name 'x' for both global variable and local variable. We get different result when we print same variable because the variable is declared in both scopes, i.e. the local scope inside the function loc() and global scope outside the function loc().

The output :- local x: 10, is called local scope of variable.

The output:- global x: 5, is called global scope of variable.

7.9 Functions using libraries

7.9.1 Built-in and Mathematical functions

Function	Description	Syntax	Example
abs ( )	Returns an absolute value of a number. The argument may be an integer or a floating point number.	abs (x)	x=20 y=-23.2 print('x = ', abs(x)) print('y = ', abs(y)) **Output:** x = 20 y = 23.2
ord ( )	Returns the ASCII value for the given Unicode character. This function is inverse of chr() function.	ord (c)	c= 'a' d= 'A' print ('c = ',ord (c)) print ('A = ',ord (d)) **Output:** c = 97 A = 65
chr ( )	Returns the Unicode character for the given ASCII value. This function is inverse of ord() function.	chr (i)	c=65 d=43 print (chr (c)) prin t(chr (d)) **Output:** A +
bin ( )	Returns a binary string prefixed with "0b" for the given integer number. **Note:** format () can also be used instead of this function.	bin (i)	x=15 y=101 print ('15 in binary : ',bin (x)) print ('101 in binary : ',bin (y)) **Output:** 15 in binary : 0b1111 101 in binary : 0b1100101

type ( )	Returns the type of object for the given single object. **Note:** This function used with single object parameter.	type (object)	x= 15.2 y= 'a' s= True print (type (x)) print (type (y)) print (type (s)) **Output:** <class 'float'> <class 'str'> <class 'bool'>
id ( )	id( ) Return the "identity" of an object. i.e. the address of the object in memory. **Note:** the address of x and y may differ in your system.	id (object)	x=15 y='a' print ('address of x is :',id (x)) print ('address of y is :',id (y)) **Output:** address of x is : 1357486752 address of y is : 13480736
min ( )	Returns the minimum value in a list.	min (list)	MyList = [21,76,98,23] print ('Minimum of MyList :', min(MyList)) **Output:** Minimum of MyList : 21
max ( )	Returns the maximum value in a list.	max (list)	MyList = [21,76,98,23] print ('Maximum of MyList :', max(MyList)) **Output:** Maximum of MyList : 98
sum ( )	Returns the sum of values in a list.	sum (list)	MyList = [21,76,98,23] print ('Sum of MyList :', sum(MyList)) **Output:** Sum of MyList : 218

format ( )	Returns the output based on the given format 1. Binary format. Outputs the number in base 2. 2. Octal format. Outputs the number in base 8. 3. Fixed-point notation. Displays the number as a fixed-point number. The default precision is 6.	format (value [, format_ spec])	x= 14 y= 25 print ('x value in binary :',format(x,'b')) print ('y value in octal :',format(y,'o')) print('y value in Fixed-point no ',format(y,'f')) **Output:** x value in binary : 1110 y value in octal : 31 y value in Fixed-point no : 25.000000
round ( )	Returns the nearest integer to its input. 1. First argument (number) is used to specify the value to be rounded.	round (number [,ndigits])	x= 17.9 y= 22.2 z= -18.3 print ('x value is rounded to', round (x)) print ('y value is rounded to', round (y)) print ('z value is rounded to', round (z))

	2. Second argument (ndigits) is used to specify the number of decimal digits desired after rounding.		**Output:1** x value is rounded to 18 y value is rounded to 22 z value is rounded to -18 n1=17.89 print (round (n1,0)) print (round (n1,1)) print (round (n1,2)) **Output:2** 18.0 17.9 17.89
pow ( )	Returns the computation of ab i.e. (a**b) a raised to the power of b.	pow (a,b)	a= 5 b= 2 c= 3.0 print (pow (a,b)) print (pow (a,c)) print (pow (a+b,3)) **Output:** 25 125.0 343

Mathematical Functions

Note

Specify **import math** module before using all mathematical functions in a program

Function	Description	Syntax	Example
floor ( )	Returns the largest integer less than or equal to x	math.floor (x)	import math x=26.7 y=-26.7 z=-23.2 print (math.floor (x)) print (math.floor (y)) print (math.floor (z)) **Output:** 26 -27 -24

ceil ( )	Returns the smallest integer greater than or equal to x	math.ceil (x)	import math x= 26.7 y= -26.7 z= -23.2 print (math.ceil (x)) print (math.ceil (y)) print (math.ceil (z)) **Output:** 27 -26 -23
sqrt ( )	Returns the square root of x Note: x must be greater than 0 (zero)	sqrt (x)	import math a= 30 b= 49 c= 25.5 print (math.sqrt (a)) print (math.sqrt (b)) print (math.sqrt (c)) **Output:** 5.477225575051661 7.0 5.049752469181039

7.9.2 Composition in functions

What is Composition in functions?

The value returned by a function may be used as an argument for another function in a nested manner. This is called **composition**. For example, if we wish to take a numeric value or an expression as a input from the user, we take the input string from the user using the function **input()** and apply **eval()** function to evaluate its value, for example:

Example :

```
# This program explains composition
        >>> n1 = eval (input ("Enter a number: "))
Enter a number: 234
        >>> n1
        234
>>> n2 = eval (input ("Enter an arithmetic expression: "))
        Enter an arithmetic expression: 12.0+13.0 * 2
>>> n2
        38.0
```

7.10 Python recursive functions

When a function calls itself is known as recursion. Recursion works like loop but sometimes it makes more sense to use recursion than loop. You can convert any loop to

recursion.

A recursive function calls itself. Imagine a process would iterate indefinitely if not stopped by some condition! Such a process is known as infinite iteration. The condition that is applied in any recursive function is known as base condition. A base condition is must in every recursive function otherwise it will continue to execute like an infinite loop.

Overview of how recursive function works

1. Recursive function is called by some external code.
2. If the base condition is met then the program gives meaningful output and exits.
3. Otherwise, function does some required processing and then calls itself to continue recursion.

Here is an example of recursive function used to calculate factorial.

Example :

```
def fact(n):
        if n == 0:
                return 1
        else:
                return n * fact (n-1)
print (fact (0))
print (fact (5))
```

Output:

```
1
120
```

print(fact (2000)) will give Runtime Error after maximum recursion depth exceeded in comparison. This happens because python stops calling recursive function after 1000 calls by default. It also allows you to change the limit using sys.setrecursionlimit (limit_value).

Example:

```
import sys
sys.setrecursionlimit(3000)
def fact(n):
        if n == 0:
                return 1
         else:
                return n * fact(n-1)
print(fact (2000))
```

Points to remember:

- Functions are named blocks of code that are designed to do one specific job.
- Types of Functions are User defined, Built-in, lambda and recursion.
- Function blocks begin with the keyword “def” followed by function name and parenthesis ().
- A “return” with no arguments is the same as return None. Return statement is optional in python.
- In Python, statements in a block should begin with indentation.
- A block within a block is called nested block.
- Arguments are used to call a function and there are primarily 4 types of functions that one can use: Required arguments, Keyword arguments, Default arguments and Variable-length arguments.
- Required arguments are the arguments passed to a function in correct positional order.
- Keyword arguments will invoke the function after the parameters are recognized by their parameter names.
- A Python function allows to give the default values for parameters in the function definition. We call it as Default argument.
- Variable-Length arguments are not specified in the function’s definition and an asterisk (*) is used to define such arguments.
- Anonymous Function is a function that is defined without a name.
- Scope of variable refers to the part of the program, where it is accessible, i.e., area where you can refer (use) it.
- The value returned by a function may be used as an argument for another function in a nested manner. This is called composition.
- A function which calls itself is known as recursion. Recursion works like a loop but sometimes it makes more sense to use recursion than loop.

Hands on Experience

1. Try the following code in the above program

Slno	code	Result
1	printinfo(“3500”)	
2	printinfo(“3500”,”Sri”)	
3	printinfo(name=”balu”)	
4	printinfo(“Jose”,1234)	
5	printinfo(“ ”,salary=1234)	

2. Evaluate the following functions and write the output

Slno	Function	Output
1	eval(‘25*2-5*4')	
2	math.sqrt(abs(-81))	
3	math.ceil(3.5+4.6)	
4	math.floor(3.5+4.6)	

3. Evaluate the following functions and write the output

Slno	function	Output
1	1) abs(-25+12.0)) 2) abs(-3.2)	
2	1) ord('2') 2) ord('$')	
3	type('s')	
4	bin(16)	
5	1) chr(13) 2) print(chr(13))	
6	1) round(18.2,1) 2) round(18.2,0) 3) round(0.5100,3) 4) round(0.5120,3)	

7	1) format(66, 'c') 2) format(10, 'x') 3) format(10, 'X') 4) format(0b110, 'd') 5) format(0xa, 'd')	
8	1) pow(2,-3) 2) pow(2,3.0) 3) pow(2,0) 4) pow((1+2),2) 5) pow(-3,2) 6) pow(2*2,2)	

Evaluation

Part - I

Choose the best answer: **(1 Mark)**

1. A named blocks of code that are designed to do one specific job is called as

   (a) Loop (b) Branching

   (c) Function (d) Block

2. A Function which calls itself is called as

   (a) Built-in (b) Recursion

   (c) Lambda (d) return

3. Which function is called anonymous un-named function

   (a) Lambda (b) Recursion

   (c) Function (d) define

4. Which of the following keyword is used to begin the function block?

   (a) define (b) for

   (c) finally (d) def

5. Which of the following keyword is used to exit a function block?

   (a) define (b) return

   (c) finally (d) def

6. While defining a function which of the following symbol is used.

   (a) ; (semicolon) (b) . (dot)

   (c) : (colon) (d) $ (dollar)

7. In which arguments the correct positional order is passed to a function?

   (a) Required (b) Keyword

   (c) Default (d) Variable-length

8. Read the following statement and choose the correct statement(s).

(I) In Python, you don't have to mention the specific data types while defining function.

(II) Python keywords can be used as function name.

   (a) I is correct and II is wrong

   (b) Both are correct

   (c) I is wrong and II is correct

   (d) Both are wrong

9. Pick the correct one to execute the given statement successfully.

   if ____ : print(x, " is a leap year")

   (a) x%2=0 (b) x%4==0

   (c) x/4=0 (d) x%4=0

10. Which of the following keyword is used to define the function testpython(): ?

   (a) define (b) pass

   (c) def (d) while

Part - II

Answer the following questions: **(2 Marks)**

1. What is function?
2. Write the different types of function.
3. What are the main advantages of function?
4. What is meant by scope of variable? Mention its types.
5. Define global scope.
6. What is base condition in recursive function
7. How to set the limit for recursive function? Give an example.

Part - III

Answer the following questions: **(3 Marks)**

1. Write the rules of local variable.
2. Write the basic rules for global keyword in python.
3. What happens when we modify global variable inside the function?
4. Differentiate ceil() and floor() function?
5. Write a Python code to check whether a given year is leap year or not.
6. What is composition in functions?
7. How recursive function works?
8. What are the points to be noted while defining a function?

Part - IV

Answer the following questions: **(5 Marks)**

1. Explain the different types of function with an example.
2. Explain the scope of variables with an example.
3. Explain the following built-in functions.
   (a) id()
   (b) chr()
   (c) round()
   (d) type()
   (e) pow()
4. Write a Python code to find the L.C.M. of two numbers.
5. Explain recursive function with an example.

Reference Books

1. *Python Tutorial book from tutorialspoint.com*
2. *Python Programming: A modular approach by Pearson – Sheetal, Taneja*
3. *Fundamentals of Python –First Programs by Kenneth A. Lambert*