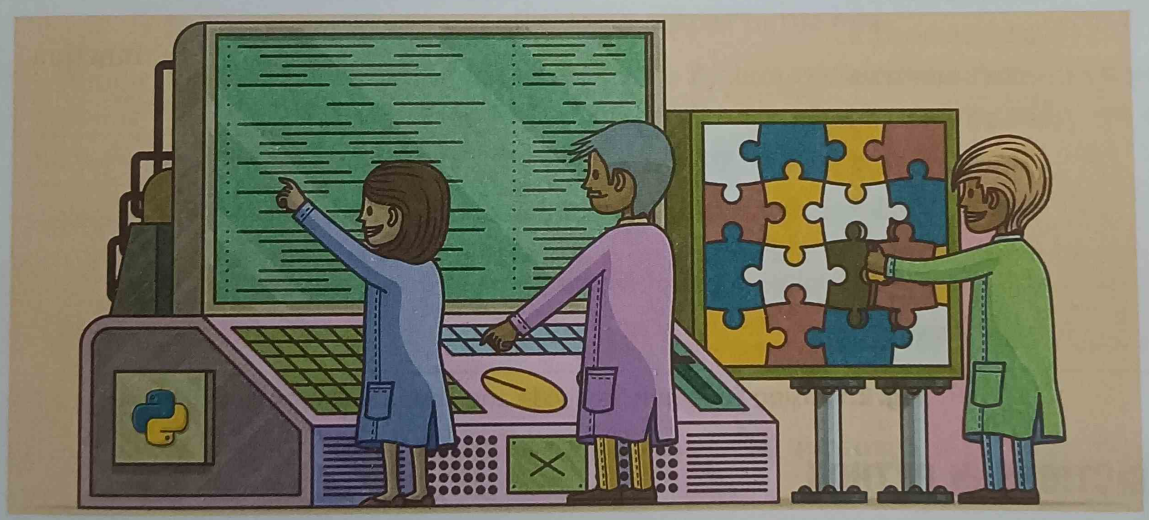
2 Functions

2.1 INTRODUCTION

So far we have introduced working modes, variables, operators, input/output operations, data types and execution of a program in Python. The codes we have studied till now have been single, conditional and iterative sequence of instructions and were small in size. We can easily work with problems whose solutions can be written in the form of small Python programs. However, as a problem becomes complex, the program also increases in size and complexity, and it becomes difficult and cumbersome for a programmer to keep track of the data and control statements in the whole program.



Thus, Python provides the feature to divide a large program into different smaller modules. These modules are known as **functions** which have the responsibility to work upon data for processing. It is difficult and time-consuming to interpret a large program every time there is a slight modification in the code. But if that program is divided into a number of small functions (also called sub-programs), then the whole task becomes easy and saves a lot of time and effort.

If required and if it becomes necessary, the sub-programs (or sub-problems) can be further divided into still smaller programs, and the process of dividing a program into a set of programs of smaller size can be continued up to any appropriate level.

Therefore, functions provide a systematic way of problem-solving by dividing the given problem into several sub-problems, finding their individual solutions, and integrating the solutions of individual problems to solve the original problem.

This approach to problem-solving is called **stepwise refinement method** or **modular approach** or **divide and conquer approach** (Fig. 2.1).

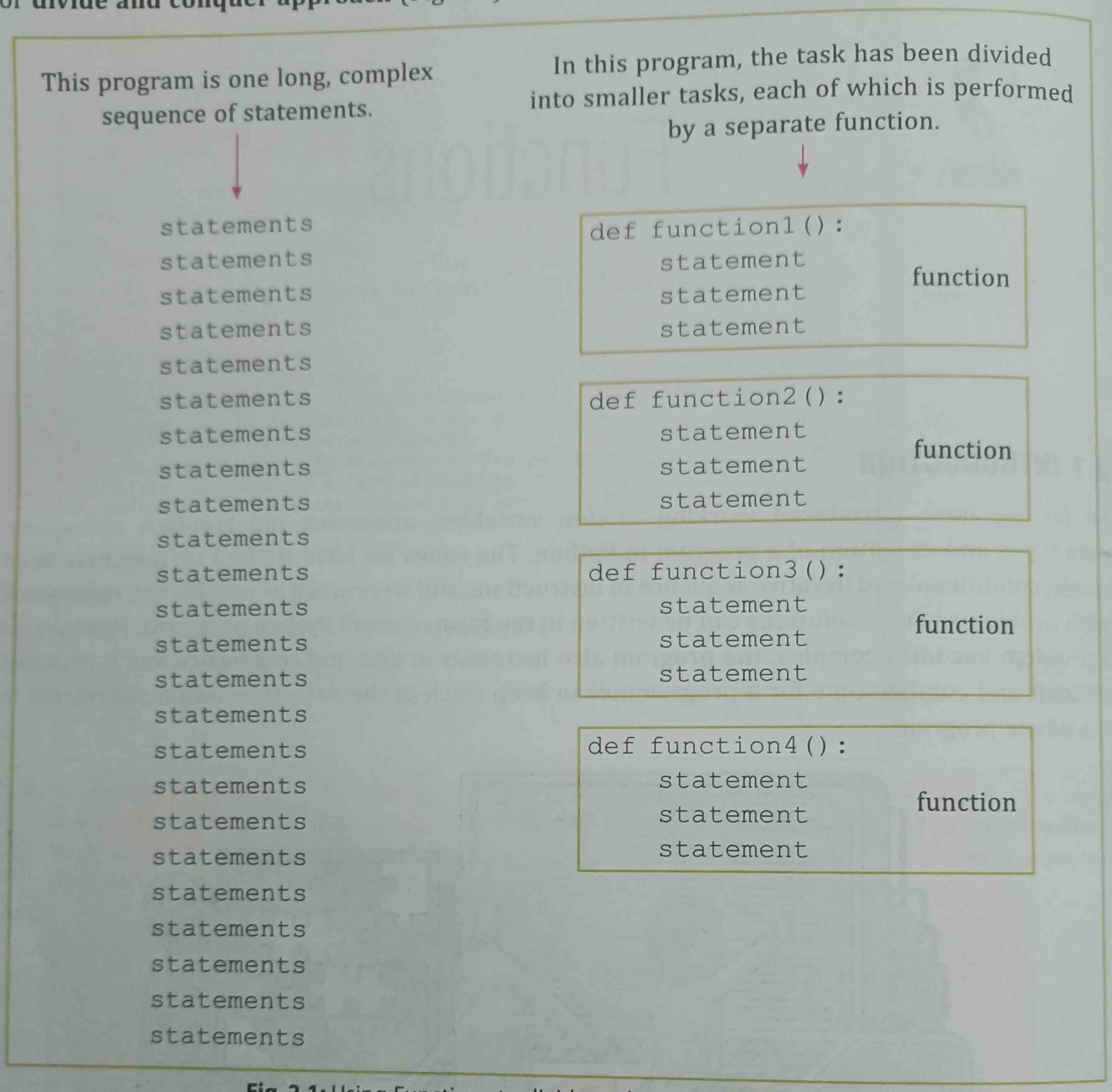


Fig. 2.1: Using Functions to divide and conquer a large task

2.2 FUNCTIONS IN PYTHON

A function is a group of statements that exists within a program for the purpose of performing a specific task. Instead of writing a large program as one long sequence of instructions, it can be written as several small functions, each performing a specific part of the task. They constitute line of code(s) that are executed sequentially from top to bottom by Python interpreter.

It is simply a group of statements under any name, *i.e.*, function name, and can be invoked (called) from another part of the program. Consider an example of School Management Software which constitutes various tasks like registering students, fee collection, issuing books from library, TC (transfer certificate) generation, result declaration, etc. In such cases, we have to create different functions for each task to manage software development.

This can be better explained using the example of a **dishwasher** from our day-to-day life. (Fig. 2.2).

A dishwasher works like a function. You put dirty dishes in the dishwasher and when you press the start button, it performs several tasks like adding detergent and water, washing dishes, and even drying them out.

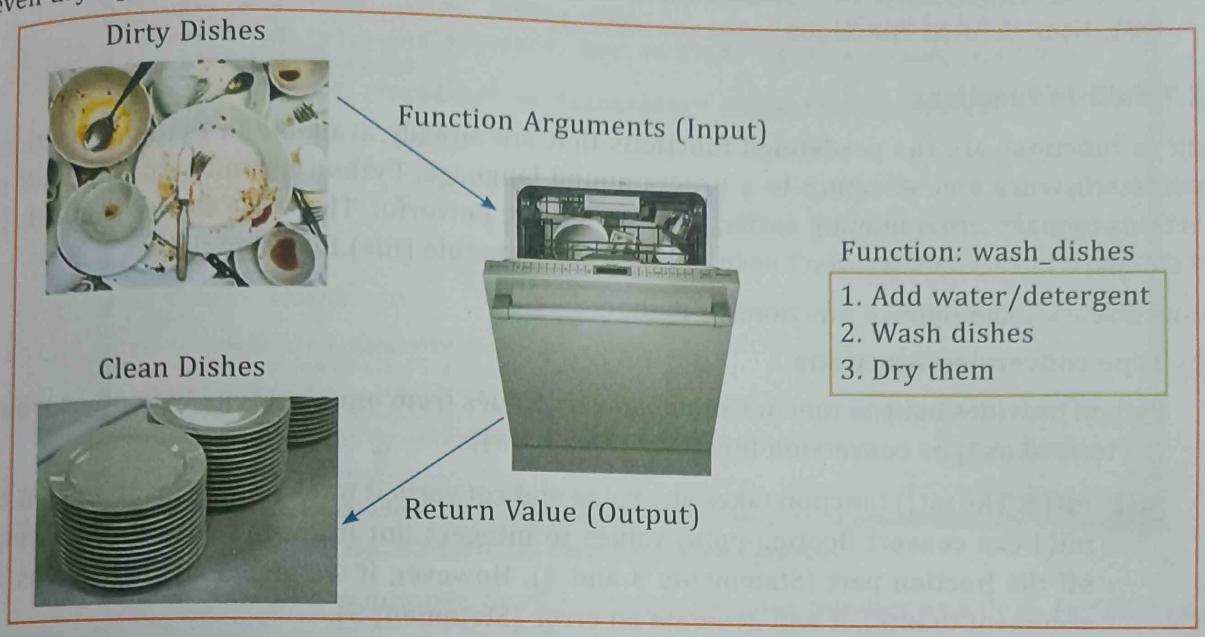


Fig. 2.2: Working of a Dishwasher similar to a Python Function

And, thus, you get clean dishes as a result (output).

Functions are also similar as they take some data as input and, depending upon the instructions/code given, perform the defined task and give some output in the end.

In Python, we have worked with the script mode in the Python revision chapter, which provides the capability of retaining our work for future usage. For working in script mode, we need to write a function in Python and save it in the file having .py extension.

A Python function is written once and is used/called as many times as required.

CTM: Functions are the most important building blocks for any application in Python and work on the divide and conquer approach.

- **Advantages of Functions:**

  Functions offer the following advantages over sequential approach of programming:

  1. **Program handling becomes easier:** Only a small part of the program is dealt with at a time.
  2. **Reduced lines of code:** While working with functions, a common set of code is written only once and can be called from any part of the program which reduces lines of code and programming overheads.
  3. **Easy updation:** In case a function is not used in a program, the same set of code which is required in multiple programs has to be written repeatedly. Hence, if we wish to make any change in a formula/expression, we have to make changes at every place, else it will result in erroneous or undesirable output. With function, however, it is required to make changes to only one location (which is the function itself).

Functions can be categorized into the following three types:

(i) Built-in Functions

(ii) Modules

(iii) User-defined Functions

2.2.1 Built-in Functions

Built-in functions are the predefined functions that are already available in Python. Functions provide efficiency and structure to a programming language. Python has many useful built-in functions to make programming easier, faster and more powerful. These are always available in the standard library and we don't have to import any module (file) for using them.

Let us discuss some built-in functions in Python.

A) Type conversion functions

Python provides built-in functions that convert values from one data type to another, which are termed as type conversion functions (Fig. 2.3(a)).

(i) **int():** The int() function takes any value and converts it into an integer (Statement 1). int() can convert floating-point values to integers but it doesn't round off; it chops off the fraction part (Statements 3 and 4). However, if we give a string value as an argument to int(), it will generate an error (Statement 5).

POINT TO REMEMBER

If no argument is passed, int() returns 0.

(ii) **str():** str() function converts its argument into a string (Statement 2).

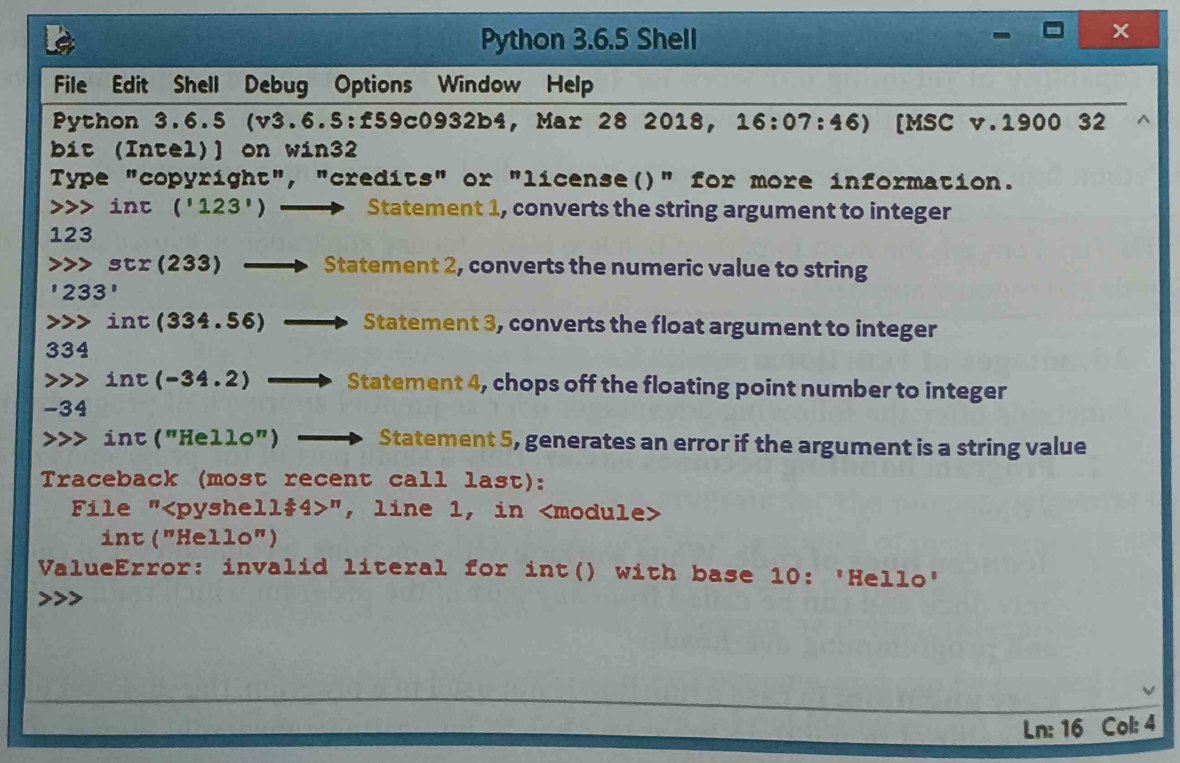


Fig. 2.3(a): Type Conversion Functions in Python

(iii) **float():** float converts integers and strings into floating-point numbers (Fig.2.3(b)). This function treats different types of parameters as explained below:

```
Python 3.6.5 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32
bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> float(26)
26.0
>>> float(21.76)
21.76
>>> float(33+25)
58.0
>>> float(12+3/4)
12.75
>>> float('3.14159')
3.14159
>>> float('40.63+'),float('86.7-'),float('35+4/7')
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    float('40.63+'),float('86.7-'),float('35+4/7')
ValueError: could not convert string to float: '40.63+'
>>>
                                                    Ln: 7  Col: 4
```

Fig. 2.3(b): float() Type Conversion Function

- If the argument is a number, float() returns the same number as a float. For example, float(26) returns 26.0, and float(21.76) returns 21.76.
- If the argument is an expression, the expression is evaluated and float() returns its value. For example, float(33+25) returns 58.0 and float(12+3/4) returns 12.75.
- If the argument is a string containing leading +/– sign and a floating point number in correct format, float() returns the float value represented by this string. For example, float ('3.14159') returns 3.14159.
- If the string argument contains any character other than leading +/– sign and floating point number in the correct format, then float() results in an error. For example, the following statements will result in errors:

  float('40.63+'), float('86.7–'), float('35+4/7')
- If the argument is a string containing leading – sign and a floating point number in correct format, float() returns the negative float value represented by this string. For example, float("–88.56") shall return –88.56 as shown in the screenshot.

```
File  Edit  Shell  Debug  Options  Window  Help
Python 3.7.0 (v3.7.0:1bf9cc5
1)] on win32
Type "copyright", "credits"
>>> float("-88.56")
-88.56
>>> float()
0.0
```

POINT TO REMEMBER

If no argument is passed, float() returns 0.0.

B) input() function

It enables us to accept an input in the form of string from the user without evaluating its value. It provides the most common way to gather input from the keyboard. The function input() continues to read input text from the user until it encounters a new line.

For example,

```
name = input('Enter a name:')
print('Welcome', name + 'Pleasure to meet you')
```

```
prog2.py - C:/python36/prog2.py (3.6.5)
File  Edit  Format  Run  Options  Window  Help
name =input ('Enter your name:')
print ('Welcome', name + 'Pleasure to meet you')
```

```
Python 3.6.5 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bi
t (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/python36/prog2.py ==========
====
Enter your name: Rinku
Welcome  RinkuPleasure to meet you
>>>
Ln: 7  Col: 4
```

The input() function allows to insert a value into a program. input() function returns a string value that can be converted into integer data type. For example,

➢ **To calculate the Selling price of an item.**

```
prog3.py - C:/python36/prog3.py (3.6.5)
File  Edit  Format  Run  Options  Window  Help
costPrice = int(input('Enter cost price: '))
profit = int(input('Enter profit: '))
sellingPrice = costPrice + profit
print('Selling Price: ', sellingPrice)
```

```
Python 3.6.5 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900
32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/python36/prog3.py ==========
=========
Enter cost price: 60
Enter profit: 20
Selling Price:  80
>>>
Ln: 8  Col: 4
```

Here, int() function is used to convert the inputted cost price and profit, which is of string type, into integer data type.

C) eval() function

This function is used to evaluate the value of a string. eval() takes a string as an argument, evaluates this string as a number, and returns the numeric result (int or float as the case may be). If the given argument is not a string, or if it cannot be evaluated as a number, then eval() result is an error.

For example,

```
Python 3.6.5 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:4
6) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more i
nformation.
>>> eval('45+10')
55
>>> eval('45.0+10')
55.0
>>> eval(43+23)
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    eval(43+23)
TypeError: eval() arg 1 must be a string, bytes or code object
>>>
                                                Ln: 6  Col: 4
```

D) max() and min() functions

These are used to find the maximum and minimum value respectively out of several values.

The **max()** function takes two or more arguments and returns the largest one.

For example,

```
Python 3.6.5 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.
1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information
.
>>> max(9, 12, 6, 15)
15
>>> max(-2, -7, -35, -4)
-2
>>> x = max(59, 80, 95.6, 95.2)
>>> print("Maximum Value", x)
Maximum Value 95.6
>>>
                                                Ln: 10  Col: 4
```

In case of String type argument, ASCII values of characters are compared for retrieving highest and lowest values using max() and min() functions.

```
>>> max("Delhi","Ujjain","Mumbai","Surat","Agartala")
'Ujjain'
>>>
```

The **min()** function takes two or more arguments and returns the smallest item.

For example,

```
Python 3.6.5 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1
900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> min(23, -109, 5, 2)
-109
>>> min(7, 26, 0, 4)
0
>>> y= min('hello', 'how', 'are', 'you', 'Sir')
>>> print("Minimum Value: ", y)
Minimum Value:  Sir
>>>
                                                    Ln: 10  Col: 4
```

Explanation:

The above lines of code shall print 'Sir' as the output. This is so because the ASCII value of 'S' is smallest, *i.e.*, 83, which is an uppercase letter and less than the rest of the words in lowercase—h ('hello'), a ('are') and y ('you') whose ASCII codes are 104, 97 and 121 respectively.

E) abs() function

The abs() function returns the absolute value of a single number. It takes an integer or float number as argument and always returns a positive value. It returns a floating or integer number depending upon an argument.

For example,

```
Python 3.6.5 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1
900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> abs(-50)
50
>>> abs(45)
45
>>> abs(-30.6)
30.6
>>>
                                                    Ln: 9  Col: 4
```

F) type() function

If you wish to determine the data type of a variable, *i.e.*, what type of value does it hold/point to, then type() function can be used.

```
Python 3.6.5 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> type (10)
<class 'int'>
>>> type (8.2)
<class 'float'>
>>> type(22/7)
<class 'float'>
>>> type('22/7')
<class 'str'>
>>> type (-18.6)
<class 'float'>
>>> type ("Hello Python")
<class 'str'>
>>> type (True)
<class 'bool'>
>>> type (False)
<class 'bool'>
>>> type (8>5)
<class 'bool'>
>>>
                                                                    Ln: 21  Col: 4
```

The Boolean argument given to type() function should have T (capital letter) for True and F (capital letter) for False; otherwise it will generate an error.

The above code can be alternatively typed as follows:

```
Python 3.6.5 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.190
0 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> x=10
>>> type (x)
<class 'int'>
>>> x=30.5
>>> print(x)
30.5
>>> type (x)
<class 'float'>
>>> x ="Hi"
>>> type (x)
<class 'str'>
>>>
                                                                    Ln: 14  Col: 4
```

G) len() function

The len() function returns the length (the number of items) of an object. The argument may be a sequence (string, tuple or list) or a mapping (dictionary).

For example,

```
Python 3.6.5 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1
900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> str= 'Hello World'
>>> len(str)
11
>>> x= [1,2,3,4]
>>> len (x)
4
>>> s = ('winter', 'spring', 'summer', 'fall')
>>> len(s)
4
>>> d={1:"CS",2:"IP",3:"IT",4:"Maths",5:"Physics"}
>>> len(d)
5
                                                Ln: 15  Col: 4
```

H) round() function

The round() function rounds a given number to the specified number of decimal places and returns the result. By default, round() function rounds a number to zero decimal places. The round() function takes two arguments.

Syntax: `round(n,p)`

Here, n is the number/expression to be rounded, and p is the number of digits up to which n is to be rounded. n is a mandatory argument and can be an integer or a floating point value while p is an optional argument and must be an integer.

The situations given below exhibit the behaviour of round() for different sets of values for n and p:

(i) p is not specified: n is rounded up to 0 digits of decimal and the result is an integer. If the first digit after the decimal value is 5 or >5, then the integer number is increased by 1.

For example, round(12.452) returns 12

Here, first digit after the decimal number is 4, so the number remains unchanged.

round(12.534) returns 13

Here, first digit after the decimal number is 5, so the number is increased by one.

(ii) p is 0: n is rounded up to 0 digits of decimal and the result is a float value.

For example, round(12.452,0) returns 12.0

round(12.534,0) returns 13.0

(iii) p is a positive integer: n is rounded up to p by checking $(p+1)^{th}$ digit; if it is 5 or >5, then p^{th} digit is increased by 1 and the result is a float.

For example, round(12.452,1) returns 12.5

round(12.534,2) returns 12.53

(iv) p is a negative integer: n is rounded up to p digits before the decimal by checking the pth digit; if it is 5 or >5, then the $(p-1)^{th}$ digit is increased by 1, all trailing p digits are converted to zero and the result is a float. The trailing p digits are rounded off to 0 in case of negative value of p.

For example, round(1234.56,-1) returns 1230.0

round(1258.4,-2) returns 1300.0

I) range() function

The range() function is used to define a series of numbers and is particularly useful in 'for loops'.

For example,

```
>>> range(5)
range(0, 5)
```

The expression range(5) above generates a series of integers from zero and ends with 4 (5-1).

- To show the list of numbers, we can use the command list(range(n)):

```
>>> list(range(5))
[0, 1, 2, 3, 4]
```

- We can explicitly define the starting and ending number by specifying the two arguments for beginning and ending numbers:

Syntax: `range(begin, end)`

For example,

```
>>> range(5, 9)
range(5, 9)
```

To show the list:

```
>>> list(range(5, 9))
[5, 6, 7, 8]
```

The above examples of *range()* demonstrated an increment of 1. We can change the way Python increments the number by introducing a third argument, the 'step'. It can be a negative or positive number, but never zero.

Syntax: `range(begin, end, stop)`

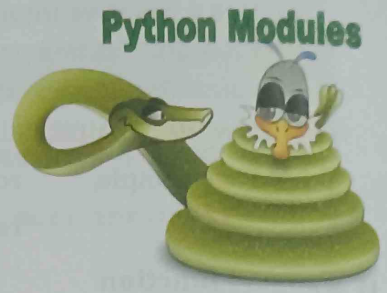
For example,

```
>>> range(10, 71, 5)
range(10, 71, 5)
```

Invoking the list, we'll see this sequence of numbers:

```
>>> list(range(10, 71, 5))
[10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70]
```

2.2.2 Modules



As the programs become more lengthy and complex, there arises a need for the tasks to be split into smaller segments called **modules**.

A module is a file containing functions and variables defined in separate files. A module is simply a file that contains Python code or series of instructions and is saved with .py extension.

In other words, the set of functions stored in a file is called MODULE and this approach is known as **modularization**, which makes a program easier to understand, test and maintain.

When we break a program into modules, each module should contain functions that perform related tasks. There are some commonly-used modules in Python that are used for certain predefined tasks and they are called **libraries**.

Modules also make it easier to reuse the same code in more than one program. If we have written a set of functions that is needed in several different programs, we can place those functions in a module. Then, we can import the module in each program that needs to call one of the functions. Once we import a module, we can refer to any of its functions or variables in our program.

CTM: A module in Python is a file that contains a collection of related functions.

We shall be discussing the most commonly-used Python module, *i.e.*, math module, in the upcoming topic.

Importing Modules in a Python Program

Python provides **import** keyword to import modules in a program. There are two methods to import modules:

(i) **Importing entire Module(s):** An entire module can be imported in the current program using import statement. The syntax to import the entire module is:

```
import<module name>
```

For example, `import math`

It gives access to all the functions of math module.

☞ To import multiple modules in a program, the syntax is:

```
import <module name1, [module2,.....]
```

For example, `import math, time, OS`

The above statement will import three modules in a program. We can access any function of the imported module by using dot notation. For example, `math.sqrt()`

(ii) **Importing selective objects from a module:** We can import all or the selected functions of a module in a program. The syntax to access all objects of a module is:

```
from <module name> import*
```

For example, `from math import*`

It gives access to all the functions of math module.

- To import a particular function of the module, the syntax is:

```
from <module name> import <object name>
```

For example,

```
from math import sqrt
```

This will import only sqrt() function from the math module.

- To import multiple functions, the syntax is:

```
from <module name> import <object1, object2, .... objectn>
```

For example,

```
from math import sqrt, pow, pi
```

This will import multiple functions, *i.e.*, sqrt, pow and pi functions of math module in a program.

```
>>> from math import sqrt,pow,pi
>>> print(sqrt(225))
15.0
>>> print(round(pi,2))
3.14
>>> print(pow(4,4))
256.0
```

As is evident from the screenshot, it is not required to use math module prefixed with the function name as it has already been imported using from statement.

However, `import` statement is the simplest and the most common way to use modules in our code.

For example, `import math`

On executing this statement, Python shall perform three operations:

(a) Search for the file, "**math.py**".

(b) Create space where modules definition and variable will be created.

(c) Execute the statements in the module.

Hence, the definitions of the module will become part of the code in which the module was imported.

`import` statement does not directly import the functions and constants in the program. To access/use any of the functions present in the imported module, we have to specify the name of the module followed by the name of the function, separated by a dot (also known as a period). This format is called **dot notation**.

For example,

```
import math
result = math.sqrt(64)    #Gives output as 8.0 which gets stored in
                           variable 'result'.
print(result)
```

sqrt() is one of the functions that belongs to math module. This function accepts an argument and returns the square root of the argument. Thus, the above statement calls the sqrt() passing 64 as an argument and returns the square root of the number passed as the argument, *i.e.*, 8.0, which is then assigned to the variable 'result'.

math module

Let us understand a few more functions from math module.

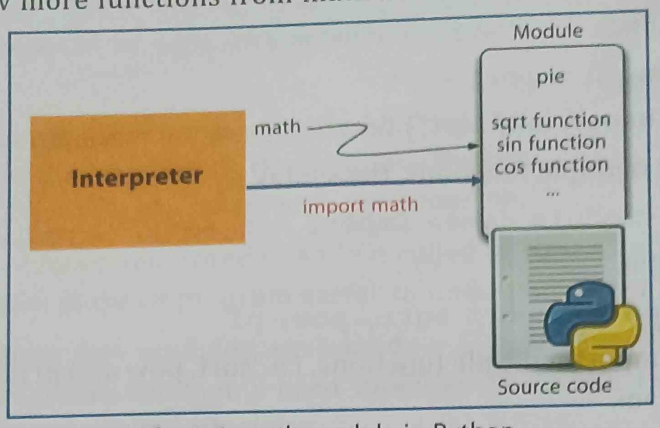


Fig. 2.4: math module in Python

```
import math   #the very first statement to be given
```

- **ceil(x):** Returns the smallest integer that is greater than or equal to x.

  For example,

```
print("math.ceil(3.4) :", math.ceil(3.4))
```

  Output: `math.ceil(3.4) : 4`

```
>>>math.ceil(-67.8)
-67
```

- **floor(x):** Returns the largest integer that is less than or equal to x.

```
>>> math.floor(-45.17)
-46.0
>>> math.floor(100.12)
100.0
>>> math.floor(100.72)
100.0
```

Note: If the argument is integer, then ceil() and floor() functions will return the same number.

- **pow(x, y):** It returns the value of x^y (x raised to the power of y), where x and y are numeric expressions, and returns the output in floating point number.

```
print("math.pow(3, 3) :", math.pow(3, 3))
```

  Output: `math.pow(3, 3) : 27.0`

```
>>> math.pow(100, -2)
0.0001
>>> math.pow(2, 4)
16.0
>>> math.pow(3, 0)
1.0
>>> print("The value of 3**4 is : ", math.pow(3,4))
The value of 3**4 is : 81.0
```

- **fabs():** Returns the absolute value (positive value) of the expression/number. It always returns a floating point number even if the argument is integer.

```
>>> import math #to import math module
>>> print('Absolute value=', math. fabs(-15))
Absolute value: 15.0
```

- **sqrt(x):** Returns the square root of x. This function accepts a number which is greater than zero, otherwise results in run-time error.

```
print("math.sqrt(65) :", math.sqrt(65))
```

Output: `math.sqrt(65) : 8.06225774829855`

```
>>> math.sqrt(100)
10.0
>>> math.sqrt(-64)
```

Results in run-time `ValueError`

```
>>> math.sqrt(7)
2.6457513110645907
```

Alternatively,

```
>>> print("math.sqrt(65) :", round(math.sqrt(65),2))
math.sqrt(65) : 8.06
```

Explanation:

The function sqrt() used for calculating the square root of 65 using statement math.sqrt(65) shall not give a perfect square. Instead, it will generate a higher precision value of 8.06225774.

This is to be rounded off to 2 decimal places using round() of math library in Python.

Ques. Write a Python module to display square root of a number using Math module.

To illustrate sqrt() function of math library

prog_sq_rootnew.py - C:/Users/preeti/AppData/Local/Programs/Python/Python3...

File Edit Format Run Options Window Help

```
#Python module to display square root of a number using
#Math module

import math
def cal_sqrt():
    number =float(input("Enter a number: "))
    sq_root =math.sqrt(number)
    #Display the square root
    print("The square root of",number, "is",sq_root)


cal_sqrt()
```

Ln: 15 Col: 0

```
>>>
 RESTART: C:/Users/preeti/AppData/
on/Python37-32/prog_sq_rootnew.py
Enter a number: 64
The square root of 64.0 is 8.0
>>>
```

For example,

```
>>> import math
>>> print(help(math.cos))
Help on built-in function cos in module math:

cos(x,/)
    Return the cosine of x (measured in radians).

None
```

String module

String module along with its commonly used functions has already been taken up and explained in Section 1.11 of Chapter 1.

random module (Generating Random Numbers)

The programming concepts we have learnt so far involved situations where we were aware of the definite output to be obtained when a particular task is to be executed, which is described as the deterministic approach. But there are certain situations/applications that involve games or simulations which work on non-deterministic approach. In these types of situations, random numbers are extensively used such as:

1) Pseudo-random numbers on Lottery scratch cards.
2) reCAPTCHA (like, in login forms) uses a random number generator to define which image is to be shown to the user.
3) Computer games involving throwing of a dice, picking a number, or flipping a coin.
4) Shuffling deck of playing cards, etc.

In Python, random numbers are not enabled implicitly; therefore, we need to invoke random module or random code library explicitly in order to generate random numbers. The first statement to be given in a program for generating random numbers is:

```
>>> import random
```

The various functions associated with the module are explained as follows:

(i) **randrange():** This method generates an integer between its lower and upper argument. By default, the lower argument is 0 and upper argument is range–1.

Syntax: `randrange(stop)` OR
`randrange(start,stop,step)`

Example 1(a): To generate random numbers from 0 to 29 (30 excluded).

```
>>> import random
>>> ran_number = random.randrange(30)
>>> print(ran_number)
1
>>>
```

Alternatively,

```
>>> import random
>>> random.randrange(30)
13
>>>
```

As is evident from the above two programs, the randrange(30) shall generate random numbers from 0 to 29. In the first code, the output is 1 and in the other code, the output is 13.

Example 1(b): To generate a random number from range 10 to 100 with the multiple of 5.

```
>>>import random
>>>val= random.randrange(10,100,5)
65
```

This code shall generate any random number between 10 and 100, which is a multiple of 5.

Example 1(c): To select a random subject from a list of subjects.

```
radn_demo.py - C:/Users/preeti/AppData/Local/Programs/Python/Python36-32/radn...
File Edit Format Run Options Window Help
#To demonstrate randrange()
#To select a random subject from a list of subjects

import random
subjects = ["Computer Science","IP","Physics","Accountancy"]
ran_index = random.randrange(2)
print(subjects[ran_index])
```

```
>>>
 RESTART: C:/Users/preeti/
rams/Python/Python36-32/ra
Computer Science
>>>
```

Explanation:

In the above program, randrange() method generates any number between 0 and 1 as 2 is excluded. So, in the above case, 0 gets selected as the random number which is stored as the index value in the variable 'ran_index'.

The element present at index 0 gets printed; as a result, the output obtained is "Computer Science".

(ii) **random()**

This function generates a random number from 0 to 1. This function can be used to generate random floating point values in the range [0.1, 1.0). It takes no parameters and returns values uniformly distributed between 0 and 1 (including 0, but excluding 1). Let us see some examples to better understand this function.

Example 2(a):

```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16
:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for m
ore information.
>>> from random import random
>>> random()
0.1353092847401708
>>> random()
0.9198861501740009
>>>
Ln: 8 Col: 4
```

As is evident from the above statements, each time random() generates a different number. The random() function can also be written with the following alternative approach:

Example 2(b):

```
>>> import random
>>> ran_number = random.random()
>>> print("The random number is :",ran_number)
The random number is : 0.29949466847152595
>>>
```

Ln: 12 Col: 4

Let us take another example that involves some simple mathematical processing.

Example 2(c): The code given below generates a random floating point number between 10 and 15:

```
>>> ran_number = random.random()*5 + 10
>>> print("The random number generated is :",ran_number)
The random number generated is : 14.407425210560241
>>>
```

Ln: 15 Col: 4

Example 2(d): Program to calculate the sum of the digits of a random three-digit number.

rad_sumofdigits.py - C:/Users/preeti/AppData/Local/Programs/Python/Python36-32/rad_sumo...

File Edit Format Run Options Window Help

```
#To calculate the sum of the digits of a random three-digit number

from random import random

n = random() *900 + 100
n = int(n)
print(n)

a = n//100
b = (n//10)%10
c = n % 10

print(a+b+c)
```

```
>>>
 RESTART:
Local/Prog
rad_sumofd
739
19
>>>
```

Ln: 14 Col: 0

Explanation:

The random() function generates a random fractional number from 0 to 1. When a random number generated using random() gets multiplied by 900, a random number is obtained from 0 to 899. When we add 100 to it, we get a number from 100 to 999.

Then, the fractional part gets discarded using int() and gets printed on the shell window. In the next statement, the first digit (the most significant) of the number is extracted by dividing it with 100 (a = n//100).

The digit at one's place is extracted by dividing the number by 10. The number obtained as the quotient further takes mod with 10 to extract its last digit which is the same as the digit placed in the middle of the original number. For extracting each digit of the number, division by 10 (c = n%10) is carried out.

In the last statement, the sum of the digits is calculated and displayed on the screen.

(iii) **randint()**

This function accepts two parameters, a and b, as the lowest and highest number; returns a number 'N' in the inclusive range(a, b); which signifies a<=N<=b, where the endpoints are included in the range. This function generates a random integer number between two given numbers. This function is best suited for guessing number applications. The syntax is:

```
random.randint(a, b)
```

Here, a is the lower bound and b is the upper bound. In case we input a number N, then the result generated will be a<=N<=b. Please ensure that the **upper limit is included** in randint() function.

Example 3(a): To generate a number between 0 and 9.

```
*imp_rand1.py - C:/Users/preeti/A...
File Edit Format Run Options Window Help
# import the random module

import random
print(random.randint(0,9))
```

```
>>>
 RESTART: C:/Users/preeti/AppData/Local/Prog
rams/Python/Python36-32/imp_rand1.py
7
>>>
Ln: 6 Col: 4
```

Example 3(b): To generate a number between 0 and 5.

```
>>> import random
>>> print(random.randint(0,5))
5
>>>
```

The above function can generate any of the integers between 0 and 5 as output, *i.e.*, either 1, 2, 3, 4 or 5.

Example 3(c): Write a program that fills a list with numbers. (Using randint())

```
*rand_list1.py - C:/Users/preeti/AppData/Local/Programs/Python/Python36-32/rand_list1.py (3.6.5)*
File Edit Format Run Options Window Help
# A function that fills a list with numbers

from random import randint


def fill_list(lst,limit_num,low,high):
    for i in range(limit_num):
        lst.append(randint(low,high))

minimum = int(input("Min : "))
maximum = int(input("Max : "))
n = int(input("Numbers limit :"))
a = []       #an empty list
fill_list(a,n,minimum,maximum)
print(a)    #Prints the newly created l
```

```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018,
16:07:46) [MSC v.1900 32 bit (Intel)] on win3
2
Type "copyright", "credits" or "license()" fo
r more information.
>>>
 RESTART: C:/Users/preeti/AppData/Local/Progr
ams/Python/Python36-32/rand_list1.py
Min : 10
Max : 50
Numbers limit :5
[33, 25, 10, 13, 20]
>>>
Ln: 9 Col: 4
```

Explanation:

The above program creates a list of integers using randint() function by accepting two parameters holding the lower limit and upper limit as the argument to this function, which are inputted by the user. Hence, the program shall display a list of numbers between the given range having lower limit as 10 and upper limit as 50 with total number of elements as 5.

Example 3(d): Write a program to perform binary search using randint().

```
bin_srch1.py - C:\Users\preeti\AppData\Local\Programs\Python\Python37-32\bin_srch1.py (3.7.0)
File Edit Format Run Options Window Help
#Binary Search using randint

from random import randint

def bin_search(lst,item):
    mid = len(lst)//2        #Integer division
    low = 0
    high = len(lst) - 1
    while lst[mid] != item and low<=high:
        if item > lst[mid]:
            low = mid + 1
        else:
            high = mid - 1
        mid = (low + high) // 2
    if low > high:
        return None
    else:
        return mid

a = []
for i in range(10):
    a.append(randint(1,20)) #list elements within the range gets automatically generated
a.sort()     #sort() used to arrange the list elements in ascending order
print(a)

value = int(input("Enter the number to be searched:"))
#index = bin_search(a,value))
print("Element found at the index : ",bin_search(a,value))
```

```
>>>
 RESTART: C:\Users\preeti\AppData\Local\Programs\Python\Python37-32\

[2, 4, 8, 10, 10, 12, 15, 15, 17, 18]
Enter the number to be searched:30
Element found at the index :  None
>>>
 RESTART: C:\Users\preeti\AppData\Local\Programs\Python\Python37-32\

[1, 2, 4, 6, 8, 10, 11, 12, 14, 15]
Enter the number to be searched:12
Element found at the index :  7
```

Explanation:

In the above program, binary search operation is performed for the numbers generated using randint() within the range of 1 to 20. This algorithm holds true in case of a sorted list only as searching begins on the basis of the middle element of the list elements.

(iv) **uniform()**

This method returns a random floating-point number in between two numbers. The syntax is:

```
random.uniform(x, y)
```

Here, x is the lower limit and y is the upper limit of random float. The returned random floating point number is greater than or equal to x and less than y.

Example 4:

```
>>> import random
>>> print("Uniform Lottery number (1,100) :",random.uniform
(1,100))
Uniform Lottery number (1,100) : 85.47678892268065
>>>
                                                    Ln: 21 Col: 4
```

(v) **choice()**

This method is used for making a random selection from a sequence like a list, tuple or string. The syntax is:

```
random.choice(sequence)
```

Example 5:

```
>>> import random
>>> direction_choice= random.choice(['1:East','2:West','3
:North','4:South'])
>>> print('My Direction is :',direction_choice)
My Direction is : 4:South
>>>
Ln: 25 Col: 4
```

(vi) **shuffle()**

This method is used to shuffle or swap the contents of a list (that is, generate a random permutation of a list in-place). The syntax is:

```
shuffle(list)
```

Here, list could be an array/list or tuple but returns reshuffled list.

Example 6:

```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
>>> import random
>>> fruits =['Apple','Orange','Banana','Pineapple','Apricot']
>>> random.shuffle(fruits)
>>> print(random.shuffle(fruits))
None
>>> import random
>>> fruits =['Apple','Orange','Banana','Pineapple','Apricot']
>>> random.shuffle(fruits)
>>> print("Reshuffled Fruits :",fruits)
Reshuffled Fruits : ['Apricot', 'Pineapple', 'Orange', 'Apple
', 'Banana']
>>> random.shuffle(fruits)
>>> print("Reshuffled Fruits :",fruits)
Reshuffled Fruits : ['Pineapple', 'Apricot', 'Orange', 'Apple
', 'Banana']
>>>
Ln: 38 Col: 4
```

As shown in the above example, each time we use the function shuffle(), the list elements shall be shuffled at random and the output generated will be different every time.

2.2.3 User-defined Functions

A function is a set of statements that performs a specific task; a common structuring element that allows you to use a piece of code repeatedly in different parts of a program.

The use of functions improves a program's clarity and readability and makes programming more efficient by reducing code duplication and breaking down complex tasks into more manageable pieces. Functions are also known as routines, sub-routines, methods, procedures or sub-programs.

☞ **How to define and call a function in Python()**

A user-defined Python function is created or defined by the **def** statement followed by the function name, parentheses () and colon (:) as shown in the given syntax:

Syntax:

```
def function_name(comma_separated_list_of_parameters):
        """docstring"""
Keyword
    Statement(s)
```

Function Definition

POINT TO REMEMBER

Statements below **def** begin with four spaces. This is called **indentation**. It is a requirement of Python that the code following a colon must be indented.

A function definition consists of the following components.

1. Keyword **def** marks the start of function header.
2. A function name to uniquely identify it. Function naming follows the same rules as rules of writing identifiers in Python.
3. Parameters (arguments) through which we pass values to a function. They are optional.
4. A colon (:) to mark the end of function header.
5. Optional documentation string (docstring) to describe what the function does. It is enclosed in three double or single quotes and spanned in multiple lines.
6. One or more valid Python statements that make up the function body. Statements must have same indentation level (usually 4 spaces).
7. An optional return statement to return a value from the function.
8. Function must be called/invoked to execute its code.

POINT TO REMEMBER

Function can alter only MUTABLE TYPE values.

Practical Implementation–1

Let us define a simple function (which does not return any value) by using the command "def func1():" and call the function. The output of the function will be "**I am learning Functions in Python**" Fig. 2.5(a).

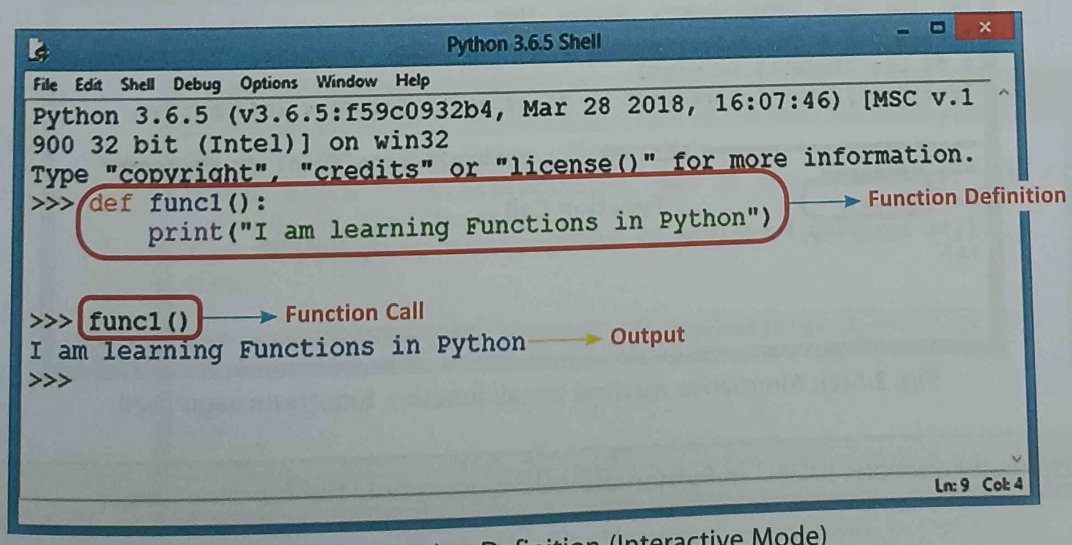


Fig. 2.5(a): Function Definition (Interactive Mode)

The above example displays a message on screen and does not contain a return statement. Such functions are called **void functions** Fig. 2.5(b).

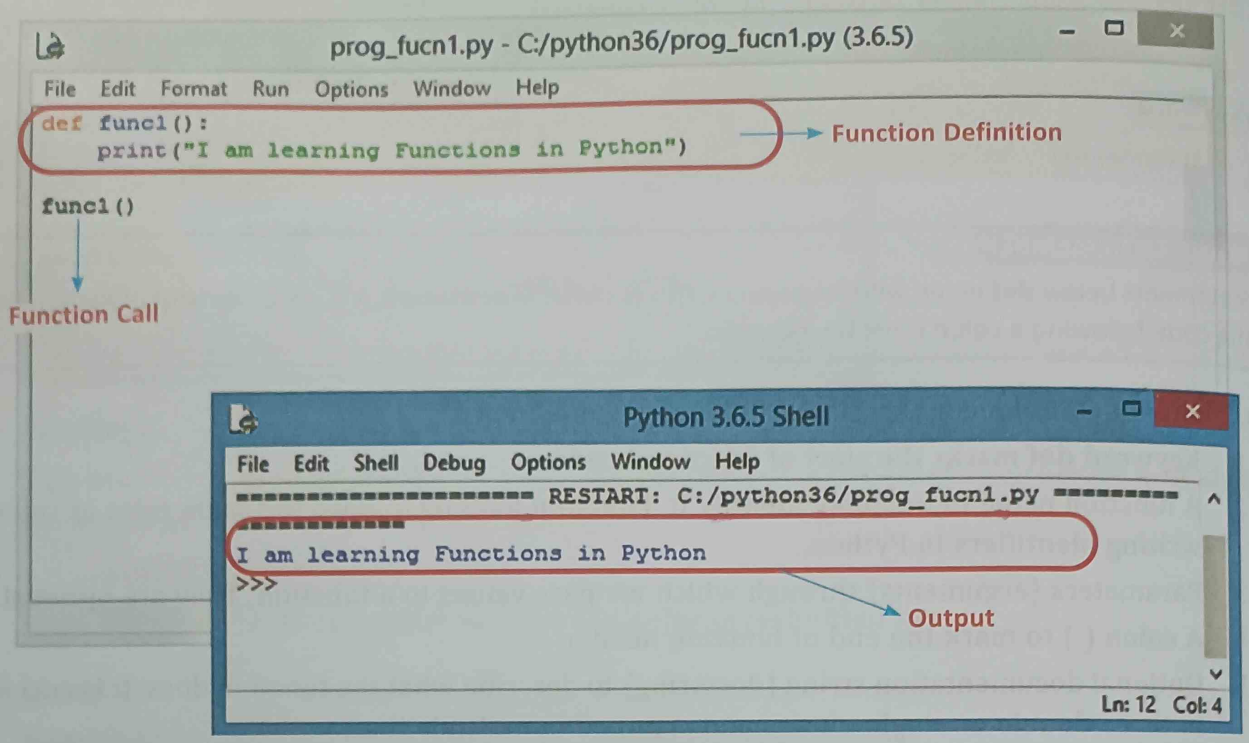


Fig. 2.5(b): Function Definition (Script Mode)

Void functions might display something on the screen or have some other effect, but they don't have a return value. If we try to assign the result of such a function to a variable, we get a special value called **None**.

Alternatively, we can make the function call directly from Python shell also as shown in Fig. 2.5(c).

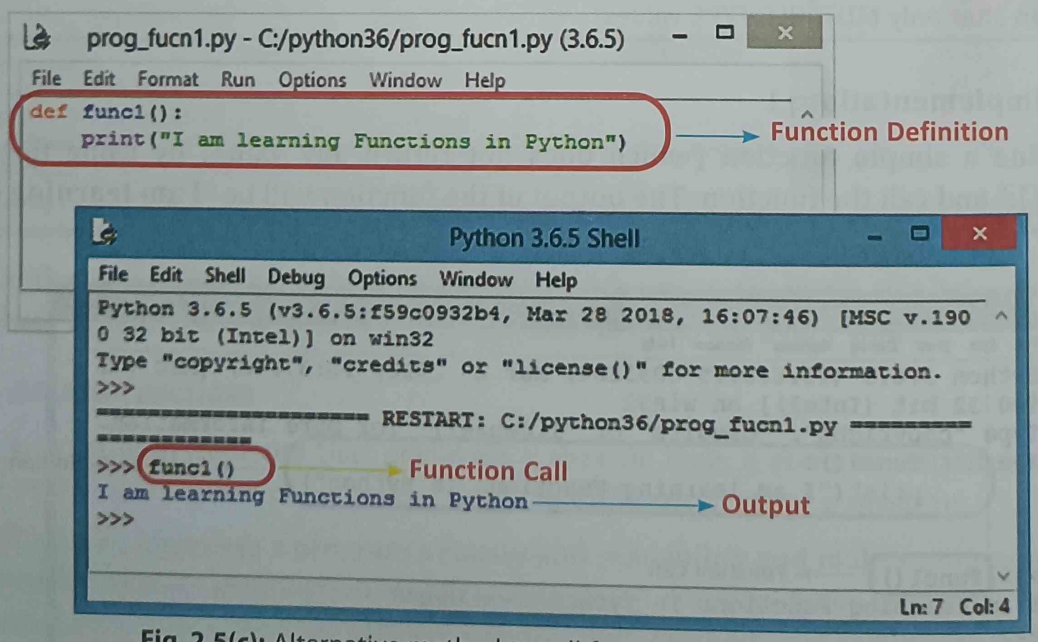


Fig. 2.5(c): Alternative method to call function func1() through shell

CTM: User-Defined Functions (UDFs) in Python are called using only the function name.

Apart from this, it is also necessary that while giving indentation, we **maintain the same indent for the rest of our code**. For example, when we call another statement "Still in function" in the same program and when it is not declared right below the first print statement, it will show a Syntax error "**unindent does not match any outer indentation level**" as shown in Fig. 2.6(c).

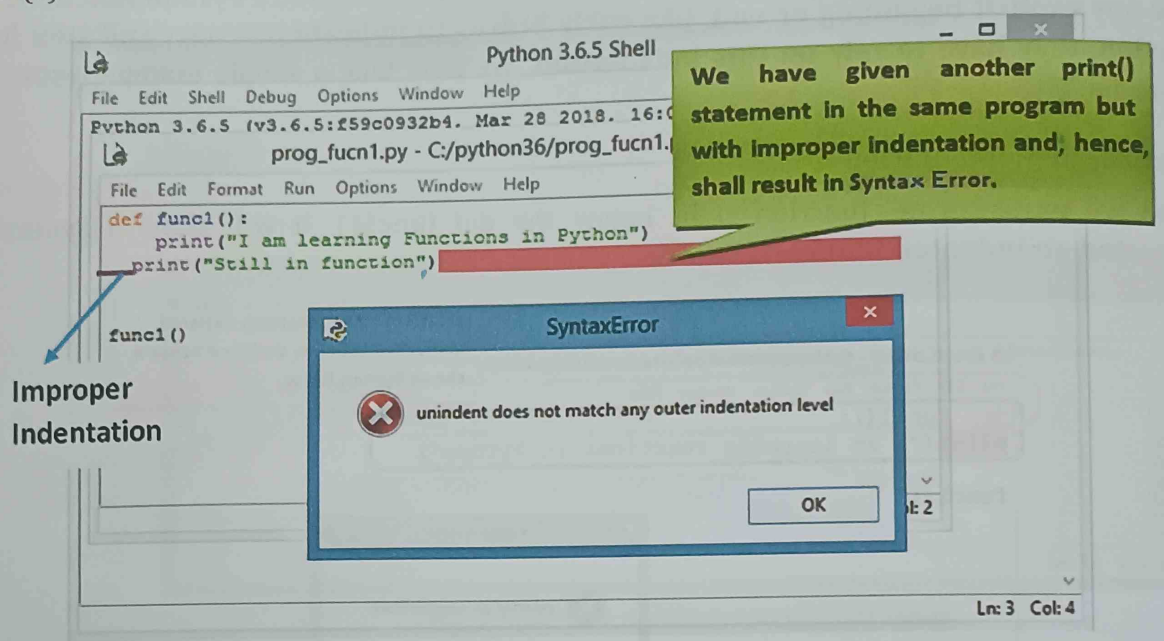


Fig. 2.6(c): Another example of improper indentation

Now, when we apply same indentation for both the statements and align them in the same line, it gives the expected output as shown in Fig. 2.6(d).

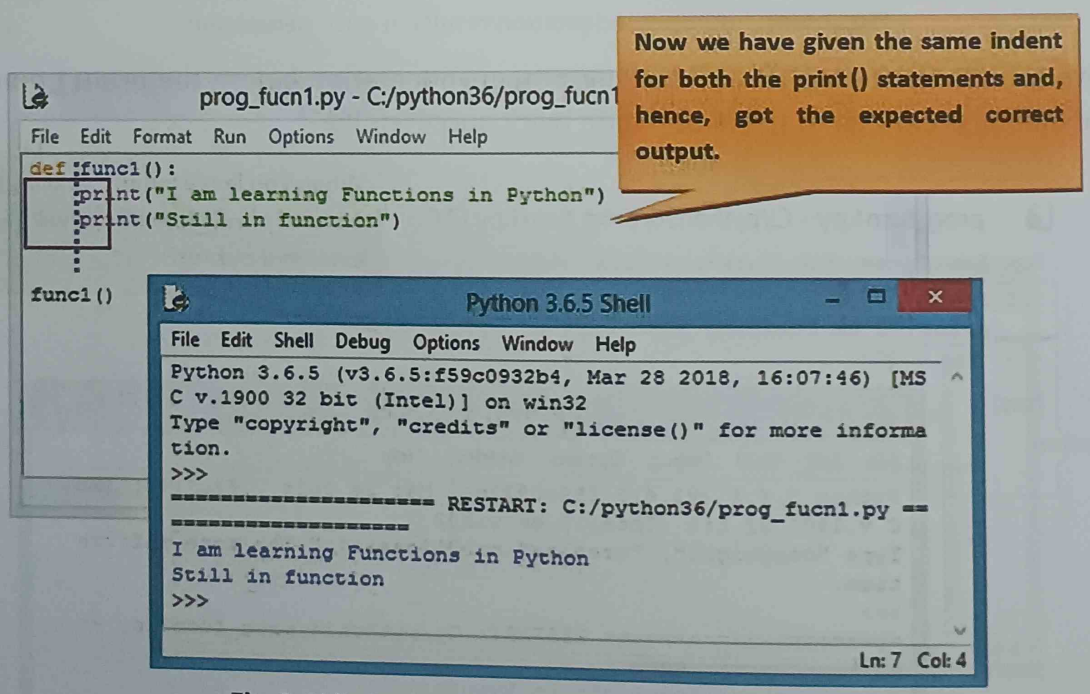


Fig. 2.6(d): Same indentation for both print() statements

Practical Implementation–3

To write a user-defined function to print a right-angled triangle.

```
def triangle():
    '''
    Objective: To print a right angled triangle
    Input Parameter: None
    Return Value: None
    '''
    '''
    Approach: To use a print statement for each line of output
    '''
    print('*')
    print('* *')
    print('* * *')
    print('* * * *')
```

```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900
32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/python36/prog_func2.py ==========
>>> triangle()
*
* *
* * *
* * * *
>>>
Ln: 10 Col: 4
```

Invoking the Function

2.3 HOW FUNCTION RETURNS A VALUE

Till now we have discussed simple user-defined functions where no value was returned to the calling function or Python Interpreter. Now, we shall be discussing the functions with some return values.

Usually, function definitions have the following basic structure:

Syntax:

```
def function_name(arguments):
    return <value>
```

return command in function is used to end the execution of the function call and also specifies what value is to be returned back to the calling function. Function can take input values as parameters, execute them and return output (if required) to the calling function with a return statement. The function returning value to the calling function is termed as fruitful function.

Practical Implementation–4

Write a Python function to compute area of a rectangle.

```
prog_func3.py - C:/python36/prog_func3.py (3.6.5)
File Edit Format Run Options Window Help
def areaRectangle(length, breadth):
    '''
    Objective: To compute the area of rectangle
    Input Parameters: length, breadth numeric value
    Return Value: area - numeric value
    '''
    area = length * breadth
    return area
```

```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v
.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more informatio
n.
>>>
========== RESTART: C:/python36/prog_func3.py =====
>>> areaRectangle(30, 20)
600
>>>
Ln: 7 Col: 4
```

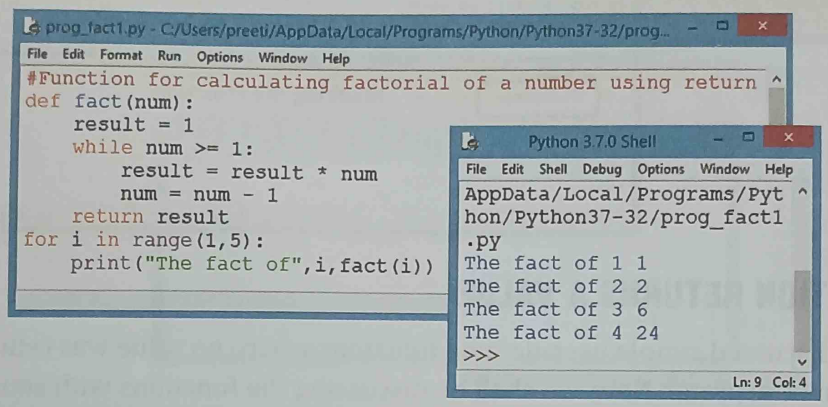
In the example, we have written a function areaRectangle() that accepts two arguments: length and breadth, and computes the area of a rectangle by multiplying length and breadth, then returns the computed value as an output using the return statement:

```
return area
```

The function areaRectangle() is called with arguments 30 and 20 which will return the computed area of a rectangle as 600.

Practical Implementation–5

Write a function to find factorial of a given number. The function should return the calculated factorial using return statement.



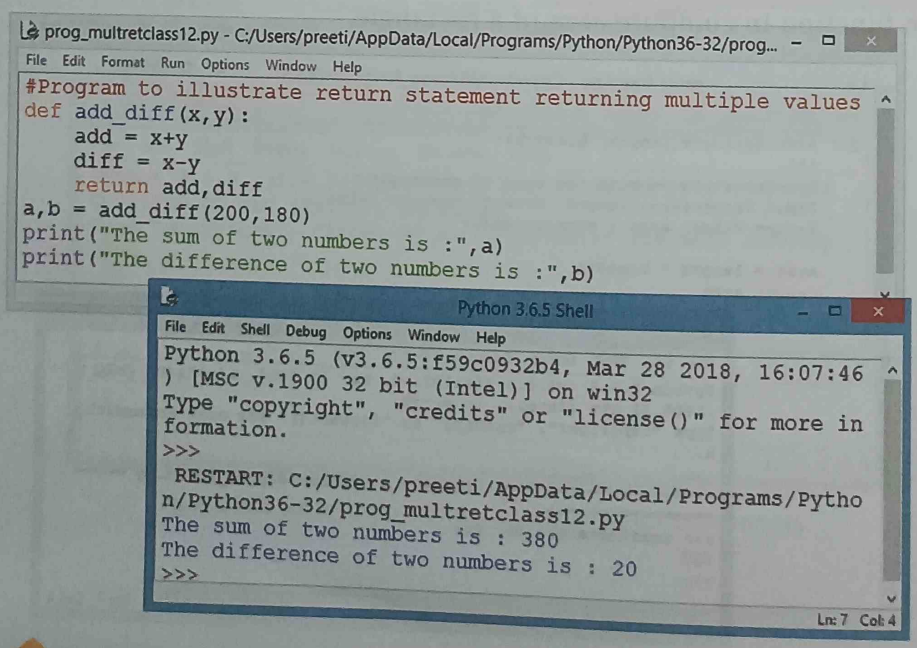
Function returning multiple values

Unlike other high-level languages like C, C++, Java, wherein a function can return at most one value, a function in Python can return multiple values. Thus, a return statement in Python function can return any number of values and the calling function should receive the values in one of the following ways:

(i) By specifying the same number of variables on the left-hand side of the assignment operator in a function call. This has been elaborated in the following examples.

Practical Implementation–6

Write a program to add and subtract two values and return the calculated result.



In the above program, we have used a single return statement to return the two values (add, diff) at the same time. We have specified two variables to receive two values from the called functions.

(ii) By receiving the returned values in the form of a tuple variable.

Practical Implementation–7

Modification of Program 6, to compute the basic calculations performed in a calculator.

```
prog_calcula_class12.py - C:/Users/preeti/AppData/Local/Programs/Python/Python36-32/prog_calcula_class12.py (3.6...
File  Edit  Format  Run  Options  Window  Help
#Program for calculator

def calc(a,b):
    add = a+b
    sub = a-b
    mul = a*b
    div = a/b
    return add,sub,mul,div      #returning multiple values using return statement
result = calc(500,40)
print(result)
print("The result can be displayed as:")
for i in result:                #displaying multiple values returned as the output
    print(i)
                                                                    Ln: 16  Col: 4
```

```
Python 3.6.5 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16
:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for m
ore information.
>>>
 RESTART: C:/Users/preeti/AppData/Local/Programs
/Python/Python36-32/prog_calcula_class12.py
(540, 460, 20000, 12.5)
The result can be displayed as:
540
460
20000
12.5
>>>
                                                    Ln: 10  Col: 4
```

Learning Tip:
A function may or may not return a value. In a nutshell, while creating functions, we can use two keywords:
1. def (mandatory)
2. return (optional)

Practical Implementation–8

Write a program to implement calculator functions using the concept of user-defined modules. (Alternative method to Program 7)

We will first create a module named 'calculate.py' as per the given code. Then, we shall call this module through Python shell prompt.

```
#Program to implement "calculate" module

def add(a,b):
    return a+b
def diff(a,b):
    return a-b
def mult(a,b):
    return a*b
def div(a,b):
    return a/b
def rem(a,b):
    return a//b
```

```
RESTART: C:/Users/preeti/AppData/Local/Programs/Python/Python36-32/calculate.py
>>> import calculate    #invoking calculate module
>>> calculate.add(5,20)
25
>>> calculate.diff(5,20)
-15
>>> calculate.mult(5,20)
100
>>> calculate.div(5,20)
0.25
>>> calculate.rem(5,20)
0
>>>
```

In the above implementation, we have written the 'calculate' module in the script mode and saved it with the name 'calculate.py'. Once this module is created and saved, it can be called anytime, along with passing the values for performing calculator operations, *i.e.*, addition, subtraction, multiplication, division and modulus of two values passed as an argument, after invoking the module (import calculate) first using the import <module_name> statement at Python shell, followed by the output displayed.

The output of the above program can be calculated by simply calling the function on IDLE, using function call statement as **name_of_the_module.function_name** (calculate.add,...). A module can contain executable statements as well as function definitions. These statements are intended to initialize the module. They are executed only once when the module name is encountered in an import statement. (They are executed even if the file is run as a script.)

Practical Implementation–9

Write a program to find the roots of a quadratic equation.

```
#Program to find the roots of a quadratic equation

def qdf(a,b,c):
    d = (b*b-4*a*c)**0.5
    return (-b+d)/(2*a),(-b-d)/(2*a)
s,t= qdf(2,-2,-2)
print(s,'\t',t)
```

```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:/Users/preeti/AppData/Local/Programs/Python/Python36-32/quad_eqtn_class12.py
1.618033988749895        -0.6180339887498949
>>>
```

As is evident from the given program, the return statement returns a value from a function and assigns it to multiple variables.

return without an expression argument is used to return from the middle of a function in which case the None value is returned.

For example,

```
>>> def func():
        pass
>>> a = func()
>>> print(a)
None
```

It is to be remembered that functions must be defined before they are called. Statements inside the definition are not executed until the function is called. The order in which statements are executed is called the flow of execution. Statements are executed one at a time, in order, until a function call is reached.

CTM: return statement returns the value of an expression following the return keyword. In the absence of the return statement, the default value None is returned.

2.4 PARAMETERS AND ARGUMENTS IN FUNCTIONS

Parameters are defined by the names (variable(s)) provided in the parentheses when we write function definition. These variables contain values required by the function to work. These parameters are called formal parameters.

If there is more than one value required by the function to work upon, then all of them will be listed in parameter list separated by comma as shown in Fig. 2.7(a).

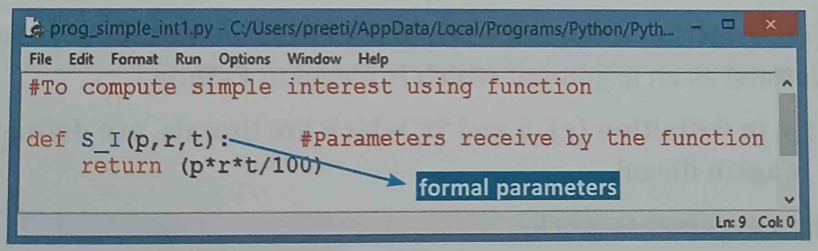


Fig. 2.7(a): Parameters in the function

In the above example, the function S_I (simple interest) is used to calculate simple interest by getting the values through IDLE, which are termed as arguments.

An **argument** is a value that is passed to the function when it is called. In other words, **arguments** are the value(s) provided in function call/invoke statement and passed to the variables of formal parameters of the function. These arguments are called actual parameters or actual arguments (Fig. 2.7(b)). List of arguments should be supplied in the same way as parameters are listed.

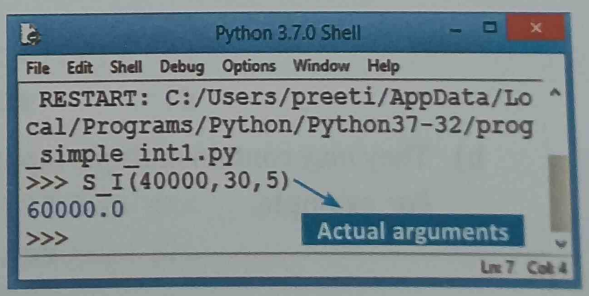


Fig. 2.7(b): Passing Arguments to the function

Bounding of parameters to arguments is done 1:1, so the number and type of arguments should be same as mentioned in parameter list. If the number of formal arguments and actual arguments differs, then Python will raise an error.

Regardless of the arguments (including the case of no arguments), a function call must end with parentheses.

Arguments in Python can be one of these value types—literals or variables or expressions—but the parameters have to be some names, *i.e.*, variables to hold incoming values; parameters cannot be literals or expressions.

CTM: The values used in function call are **actual parameter** and **actual argument**.
The variables used in function header are **formal parameter** and **formal argument**.

A few things to remember about parameter passing:

In Python functions, if you are passing values of immutable types, *i.e.*, numbers, strings, etc., to the called function, then the called function cannot alter their values. But in case parameters passed are mutable in nature, such as lists or dictionaries, then called function would be able to make changes to them.

Ques: Given below are some function definitions. Identify which of these are invalid giving reasons:

(a)
```
def product(a,b):
    print(a*b)
```

(b)
```
def product(a+1,b):
    print(a*b)
```

(c)
```
def product(5,'b')
    print(a*b)
```

Ans: (a) Function definition is valid as the parameters defined are valid variable names.

(b) and (c) Function definitions are invalid as in definition (b), expression (a+1) has been defined as an argument which is illegal in Python.

Similarly, in definition (c), 5 and 'b', which are literals, are defined as the parameter which is again illegal.

Let us see how Python arguments work.

a) Functions may be simple one-to-one mappings

For example,
```
>>> def f1(x):
...     return x*(x-1)
...
>>> f1(3) #Calling the function with the value of x as 3
6          #Output
```

b) They may contain multiple input and/or output variables as well as return multiple values.

For example,
```
>>> def f2(x,y):
...     return x+y, x-y
...
>>> f2(3,2)
(5,1)
```

c) Functions don't need to contain arguments at all.

For example,

```
>>> def f3():
...     print('Hello world')
...
>>> f3()
Hello world
```

d) The user can set arguments to default values in function definitions.

For example,

```
>>> def f4(x,a=1):
...     return a*x**2
...
```

- If this function is called with only one argument, the default value of 1 is assumed for the second argument:

```
>>> def f4(x,a=1):
...     return a*x**2
>>> f4(2)
4
```

- However, the user is free to change the second argument from its default value while invoking/calling the function:

```
>>> f4(2,a=2) #f4(2,2) would also work
8
```

2.4.1 Types of Arguments

Let us first understand the types of arguments provided by Python in detail. Consider the example given as under:

For example,

```
def f1(x,y):
    statement
    ------------------
f1(20,30)
```

Here, x, y are formal arguments whereas 20, 30 are actual arguments.

On the basis of examples we have discussed so far, only four types of actual arguments are allowed in Python. They are:

1. Positional arguments
2. Default arguments
3. Keyword arguments
4. Variable length arguments

1) **Positional arguments:** Positional arguments are arguments passed to the actual arguments of a function in correct positional order, *i.e.*, in the same order as in the function header.

```
position_arg1.py - C:/Users/preeti/AppData/L...
File Edit Format Run Options Window Help
#To demonstrate Positional arguments

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

```
RESTART: C:/Users/preeti/AppData/Local/Pr
ograms/Python/Python36-32/position_arg1.py
>>> subtract(100,200)
-100
>>> subtract(200,100)
100
>>>
Ln: 9 Col: 4
```

a gets 100 and b gets 200
Output: –100

a gets 200 and b gets 100
Output: 100

As shown in the given program, the number and position of arguments must be matched. If we change their order, then the result will be changed. If we change the number of arguments passed, then it shall result in an error.

2) **Default arguments:** A default argument is an argument that can assign a default value if a value is not provided in the function call for that argument. The default value is assigned at the time of function definition by using the assignment (=) operator in the format:

Argumentname = value

Example 7:

```
key_arg1.py - C:/Users/preeti/AppData/Local/Progra...
File Edit Format Run Options Window Help
def greet_msg(name = "Geetu"):
    print("Hello",name,"Good Morning")

greet_msg("Vinay")    #valid
greet_msg()           #valid
```

```
>>>
 RESTART: C:/Users/preeti/AppData/Local/Progr
ams/Python/Python36-32/key_arg1.py
Hello Vinay Good Morning
Hello Geetu Good Morning
>>>
Ln: 7 Col: 4
```

If we are not passing any value to name argument, then only default values will be considered, but if a value is provided, it will overwrite the default value. We must remember that if we are passing default arguments to a function, then all the arguments to its right must also have default values, *i.e.*, positional arguments must appear before the default arguments, otherwise it will result in SyntaxError as shown below:

```
key_arg1.py - C:/Users/preeti/AppData/Local/Programs/Python/Python36-32/key_...
File Edit Format Run Options Window Help
def greet_msg(name = "Geetu",msg="Good Morning"):
    print("Hello",name,"Good Morning")

def greet_msg(name,msg="Good Morning"):
    print(name,msg)    #valid

def greet_msg(name="abc",msg):          #invalid
    print(name,msg)
```

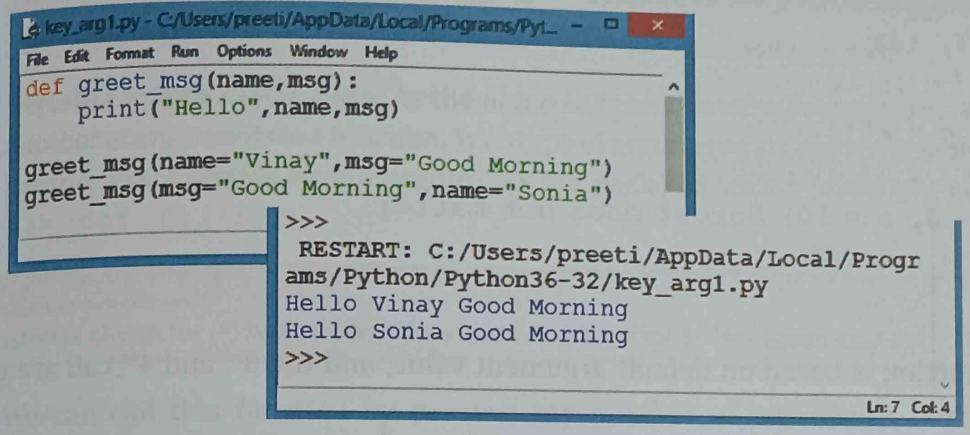
SyntaxError

non-default argument follows default argument

OK

3) **Keyword arguments:** To get control and flexibility over the values sent as arguments, Python offers **keyword arguments**. This allows to call function with arguments in any order using name of the arguments. Thus, using keyword arguments, we can pass argument values by keyword, *i.e.*, by parameter name.

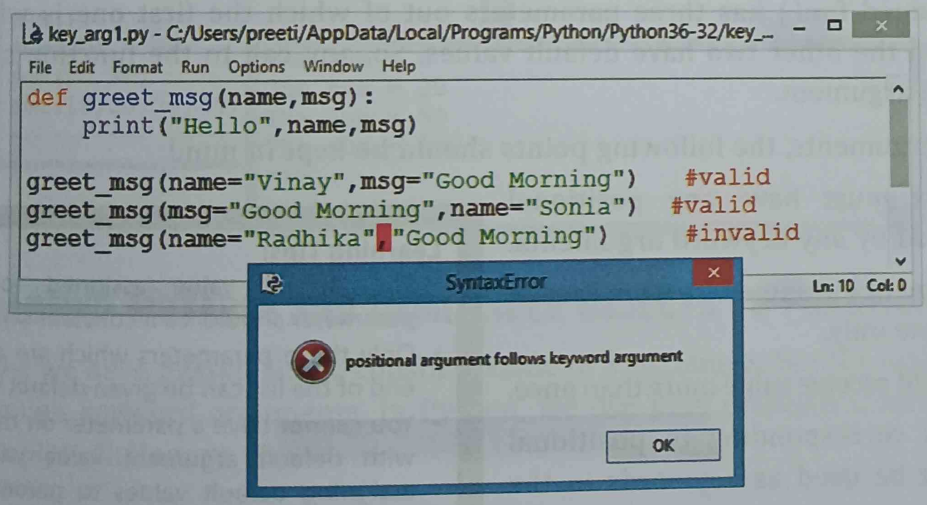
In a function call, we can specify values for some parameters using their **name** instead of the position (order). These are called **keyword arguments** or **named arguments**.



Here, the order of arguments is not important but the number of arguments must be matched.

CTM: Keyword arguments are the named arguments with assigned values being passed in the function call statement.

Note: We can use both positional and keyword arguments simultaneously. But first we have to specify positional argument and then the keyword argument, otherwise it will generate a syntax error as shown in the screenshot that follows.



Advantages of writing functions with keyword arguments are:

- Using the function with keyword arguments is easier as we do not need to remember the order of the arguments.
- We can specify values of only those parameters which we want to, as other parameters have default argument values.

Example 8: Consider the following function definition:

```
def fun(a, b=1, c=5):
    print("a is ", a, "b is ", b, "c is ", c)
```

The function fun() can be invoked in many ways:

1. `>>> fun(3)`
   `a is 3 b is 1 c is 5`
2. `>>> fun(3, 7, 10)`
   `a is 3 b is 7 c is 10`
3. `>>> fun(25, c = 20)`
   `a is 25 b is 1 c is 20`
4. `>>> fun(c = 20, a = 10) #order does not matter.`
   `a is 10 b is 1 c is 20`

Explanation:

1st and 2nd call to function is based on default argument value, and the 3rd and 4th call are using **keyword arguments.**

In the first usage, value 3 is passed to **a** while **b** and **c** work with default value. In the second call, all the three parameters get values in function call statement, and values of **b** and **c** will be overwritten by the default values. In the third usage, variable **a** gets the first value 25 due to the position of the argument, and parameter **c** gets the value 20 due to naming, *i.e.*, keyword arguments. Parameter **b** uses the default value. In the fourth usage, we use keyword argument **a** and **c** values as we have specified the value for **c** before **a**; although **a** is defined before **c** in the parameter list, the parameter **b** uses the default value.

Note: The function named fun() has three parameters out of which the first one is without default value and the other two have default values. So, any call to the function should have at least one argument.

While using keyword arguments, the following points should be kept in mind:

- An argument list must have any positional arguments followed by any keyword arguments.
- Keywords in argument list should be from the list of parameters name only.
- No parameter should receive value more than once.
- Parameter names corresponding to positional arguments cannot be used as keywords in the same calls.

Learning Tips:

- The default value assigned to the parameter should be a constant only.
- Only those parameters which are at the end of the list can be given default value.
- You cannot have a parameter on the left with default argument value without assigning default values to parameters lying on its right side.
- The default value is evaluated only once at the point of function definition.

On the basis of rules defined above for combining all three types of arguments, following are the examples of valid/invalid function call:

Consider the given user-defined function to calculate average of two numbers:

```
def Average(n1,n2,n3=1000):
    return (n1+n2+n3)/3
```

Function Call	Legal/Illegal	Reason
Average(n1=20,n2=40,n3=80)	LEGAL	Non-default values provided as named arguments
Average(n3=10,n2=7,n1=100)	LEGAL	Keyword argument can be in any order
Average(100,n2=10,n3=15)	LEGAL	Positional argument before the keyword arguments
Average(n3=70,n1=90,100)	ILLEGAL	Keyword argument before the positional arguments
Average(100,n1=23,n2=1)	ILLEGAL	Multiple values provided for n1
Average(20,num=9,n3=11)	ILLEGAL	Undefined argument num

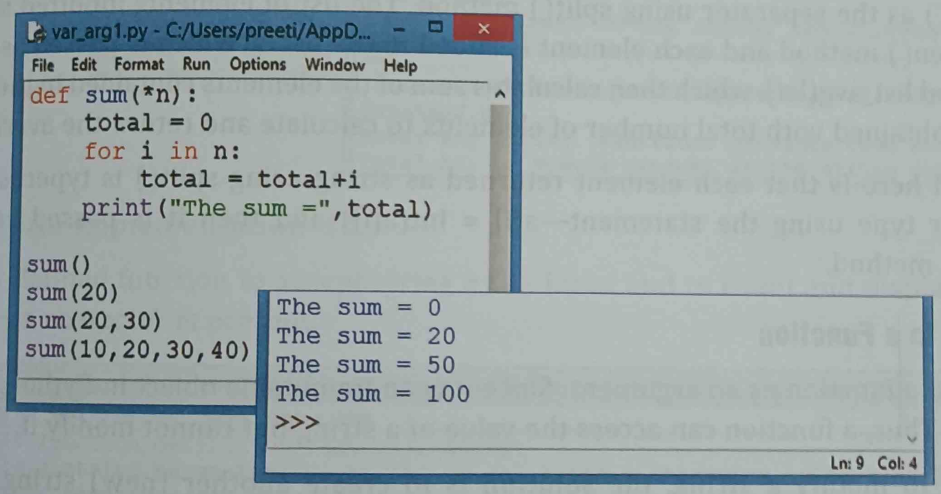
4) **Variable length arguments:** As the name suggests, in certain situations, we can pass variable number of arguments to a function. Such type of arguments are called variable length arguments/parameters. Variable length arguments are declared with * (asterisk) symbol in Python as:

```
>>> def f1(*n):
```

POINT TO REMEMBER

The asterisk character (*) has to precede a variable identifier in the parameter list.

We can call this function by passing any number of arguments, including zero number. Internally, all these values are represented in the form of a tuple.



2.5 PASSING MUTABLE AND IMMUTABLE OBJECTS TO FUNCTIONS

Till now we have passed single constant value to functions either through positional arguments or through keyword arguments. In Python, we can pass multiple values to the function using lists, tuples, dictionary, etc.

2.5.1 Passing Arrays/Lists to Functions

Arrays in basic Python are actually lists that can contain mixed data types. However, for proper implementation of arrays, we require Numpy library/interface, which is beyond the scope of this book. So, we shall be implementing lists as arrays using Python shell.

However, lists are better than arrays as they may hold elements of several data types, whereas in case of arrays, the elements should be of same data type only and, hence, lists are much more flexible and faster than arrays. A list can be passed as argument to a function similar to passing normal variables as arguments to a function.

```
#Arrays(list) passing to a function that calculates
#the arithmetic mean of list elements

def list_avg(lst):
    l = len(lst)
    sum = 0
    for i in lst:
        sum += i
    return sum/l

print("Input Integers :")
a = input()
a = a.split()
for i in range(len(a)):
    a[i] = int(a[i])


avrg = list_avg(a)

print("Average is :")
print(round(avrg,2))
```

```
>>>
 RESTART: C:/Users/preeti/AppData/Local/Programs/Py
thon/Python36-32/array1.py
Input Integers :
4 5 6 7 8 2
Average is :
5.33
>>>
```

Explanation:

In the above program, the input is fetched as a string and the elements are returned as a list of strings with space (" ") as the separator using split() method. The list of elements inputted shall be counted for using len() method and each element is added in the list 'a' which is passed as the argument to the method list_avg(lst), which then calculates sum of the elements contained in the list and dividing the sum obtained with total number of elements to calculate and return the average.

One point to be noted here is that each element returned as string using split() is typecast or converted into integer type using the statement—a[i] = int(a[i]) and then it is passed as an argument to list_avg() method.

2.5.2 Passing Strings to a Function

String can be passed to a function as an argument. Since it is an immutable object in Python, it is used as pass by value. Thus, a function can access the value of a string but cannot modify it.

However, if you wish to modify a string, the solution is to create another (new) string and concatenate the modified value of parameter string in the newly created string.

This can be explained with the following example:

Example 9:

```
# Python program to pass a string to the function

# function definition: it will accept
# a string parameter and print it
def printMsg(str1):
        # printing the parameter
        print(str1)

# Main code
# function calls
printMsg("Hello world!")
printMsg("Start using Functions")
```

```
>>>
 RESTART: C:/Users/preeti
passing1.py
Hello world!
Start using Functions
```

Explanation:

In the given program, two strings are passed as arguments to the user-defined function printMsg() by giving two function calls. These string messages are received as a parameter in the function definition and finally printed.

Practical Implementation–10

User-defined function to accept string as an input and to count and display the total number of vowels present in it.

prog_string_passing2.py - C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/p...

File Edit Format Run Options Window Help

```
# function definition that will accept
# a string parameter and return number of vowels
def countVowels(str1):
        count = 0
        for ch in str1:
                if ch in "aeiouAEIOU":
                        count +=1
        return count

# Main code
# function calls
str_input = input("Enter any String :")
count =countVowels(str_input)
print("Total no. of vowels present in the string are :",count)
```

```
>>>
 RESTART: C:/Users/preeti/AppData/Local/Programs/Python
passing2.py
Enter any String :Functions provides Code Reusability
Total no. of vowels present in the string are : 13
```

Practical Implementation–11

User-defined function to accept string as an input and to count and display the total number of times a character is present.

prog_count_char.py - C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/prog_count_char.py (3.7.0)

File Edit Format Run Options Window Help

```
# function definition that will accept
# a string parameter, count the no. of times a character is present inside it
def countchar(str,ch):
    count = 0
    for i in str:
        if i == ch:
            count +=1
    return count

# Main code
# function calls
str_input = input("Enter any String :")
ch1 =input("Enter the character to count :")
result =countchar(str_input,ch1)
if result == 0:
    print(ch1,"does not exist in the string")
else:
    print(ch1,"exist",result,"times in the string")
```

```
>>>
 RESTART: C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/prog_count
har.py
Enter any String :To count the number of times a character is present
Enter the character to count :e
e exist 6 times in the string
```

2.5.3 Passing Tuple to a Function

You can also pass a tuple as an argument to a function. Because of its immutable nature, a function can only access the value of a tuple but cannot modify it. It will become clearer from the following examples:

Example 10:

prog_tupinside_funct.py - C:/Users/preeti/AppData/Loc...

File Edit Format Run Options Window Help

```
#Passing immutable tuple to a function

def func(A):
    A=list(A) #tuple converted to a list
    A[0] = A[0]*2
    A[1]= A[1] + 10
    print("Value inside function:", A)
tup=(100,200)
print(tup)
func(tup)
print(tup)
```

```
>>>
 RESTART: C:/Users/preeti/AppData/Local
de_funct.py
(100, 200)
Value inside function: [200, 210]
(100, 200)
```

In the above program, tuple 'tup' is passed as an argument to the function func. In order to perform mathematical processing, it is required to be converted into a list using the method list(). The next two statements alter the value of list and, hence, the changes are reflected when these values are displayed.

But outside the function, the values of tuple are not modified and, hence, remain the same as 100 and 200 respectively.

Practical Implementation–12

Program to input elements in a tuple and to count and display number of even and odd numbers present in it using function.

prog_evenodd_tuple1.py - C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/prog_evenodd_tuple1.py...

File Edit Format Run Options Window Help

```
#To input elements in a tuple and pass it to a function to determine the total
#number of even and odd numbers in it.

def countEvenOdd(tup):
    counteven=0
    countodd =0
    for i in tup:
        if i % 2 ==0:
            counteven += 1
        else:
            countodd += 1
    return counteven,countodd

tup1=()
n=int(input("Enter the total elements in a tuple: "))
for i in range(n):
    num=int(input("Enter the number :"))
    tup1=tup1 + (num,)
count_stats=countEvenOdd(tup1)
print("Even numbers are:",count_stats[0])
print("Odd numbers are:",count_stats[1])
```

```
>>>
 RESTART: C:/Users/preeti/AppData/Local/Programs
_tuple1.py
Enter the total elements in a tuple: 10
Enter the number :1
Enter the number :2
Enter the number :3
Enter the number :4
Enter the number :5
Enter the number :6
Enter the number :7
Enter the number :8
Enter the number :9
Enter the number :10
Even numbers are: 5
Odd numbers are: 5
```

2.5.4 Passing Dictionary to a Function

Python also allows us to pass dictionary to a function. Since dictionaries are mutable in nature, function can alter the values of dictionary in place.

Let us look at some examples of how to pass dictionary to a function.

Practical Implementation–13

To pass student record as a dictionary to a function and update their marks.

```
prog_dict_funct.py - C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/prog_dic...
File Edit Format Run Options Window Help
#To pass student record as a dictionary and update their marks

def Marks_increase(stud,P):
    stud["Marks"] += P
    stud["Status"] ="Updated"

student1={"Rollno":1,"Name":"Radhika","Marks":89,"Status":"*"}
student2={"Rollno":4,"Name":"Shaurya","Marks":90,"Status":"*"}
Marks_increase(student1,50)
Marks_increase(student2,60)
print(student1)
print(student2)
```

```
>>>
 RESTART: C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/prog_dict
nct.py
{'Rollno': 1, 'Name': 'Radhika', 'Marks': 139, 'Status': 'Updated'}
{'Rollno': 4, 'Name': 'Shaurya', 'Marks': 150, 'Status': 'Updated'}
```

Practical Implementation–14

Program to pass a dictionary to a function with list of elements as keys and frequency of its occurrence as value and return as a dictionary.

```
prog_dict_func.py - C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/prog_dict_func.py (3.7.0)
File Edit Format Run Options Window Help
#Program to pass a dictionary to a function with list of elements as keys
#and frequency of its occurrence as value and return as a dictionary

def frequencyCount(list1,dict):
    for i in list1:
        if i not in dict:
            dict[i] =1
        else:
            dict[i] += 1
    return dict

list1=[4,5,6,5,10,6,5,5,40,30,2,4]
d ={}
frequencyCount(list1,d)
print(d)
```

```
>>>
 RESTART: C:/Users/preeti/AppData/Local/Programs/Python
nc.py
{4: 2, 5: 4, 6: 2, 10: 1, 40: 1, 30: 1, 2: 1}
```

Anything calculated inside a function but not specified as an output (either with return or global) will be deleted once the function stops running.

For example,

```
>>> def f5(x,y):
...     a = x+y
...     b = x-y
...     return a**2, b**2
...
>>> f5(3,2)
(25,1)
```

If we try to call a or b, we get an error message:

```
>>> a
Traceback (most recent call last):
  File "<stdin>", line 1, in?
NameError: name 'a' is not defined
```

This brings us to scoping issues, which will be addressed in the next section.

2.6 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. Scope of variables refers to the part of the program where it is visible, *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 scope of variables—global scope or local scope.

- Global (module)
  - Names assigned at the top level of a module, *i.e.*, outside of any function, or directly in the interpreter
  - Names declared with global keyword in a function
  - Can be accessed inside or outside of the function
- Local (function)
  - Names assigned inside a function definition or loop
  - Cannot be accessed outside the function

☞ We can access global variable in the absence of local variable with the same name.

For example,

```
>>> a = 2          # a is assigned in the interpreter, so it's global
>>> def f(x):      # x is in the function's argument list, so it's local
...     y = x+a    # value of a is not defined in function, so it can access
                   # value of global a and y is only assigned inside the
...     return y # function, so it's local
>>> p=f(5)
>>> print(p)
7
```

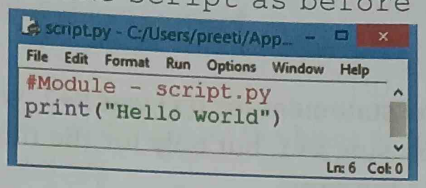
☞ We can use the same names in different scopes but the priority is given to the local variable. For example,

```
>>> a = 2
>>> def f5():
...     a=10   #this 'a' has no knowledge of the global 'a' and vice versa
...     print("a in function",a)
...
>>> a
2
>>>f5()
a in function 10
```

- The local 'a' is deleted as soon as the function stops running. So, the life time of a local variable gets over early in comparison to a global variable.

  For example,

```
>>> x = 5
>>> import script                    #same script as before
Hello world
>>> x
5
```



  Here, lifetime of a variable is the time for which a variable lives in memory. For global variables, lifetime is entire program run (*i.e.*, they live in memory as long as the program is running) and for local variables, lifetime is their function's run (*i.e.*, as long as their function is being executed).

- Value of global variable can be modified using `global` keyword.

  For example,

```
>>> a = 2
>>> def f5():
        global a            #accessing variable a using global keyword
        a=a+5               #a=2+5
        return a
>>> a
2
>>> f5()
7
>>> a                       #value of global a is changed from 2 to 7
7
```

- Changing a global name used in a function definition changes the function.

  For example,

```
>>> a = 2
>>> def f(x):
...     return x+a          #this function is, effectively, f(x) = x+2
...
>>> f(4)
6
>>> a = 1
>>> f(4)                    #since we set a=1, f(x) = x+1 now
5
```

- Unlike some other languages, Python function arguments are not modified by default:

```
>>> x = 4
>>> f(x)
5
>>> x
4
```

In the previous example, the statement for f(x) was x+1, which is carried forward in this example also. x=4 gets changed to 5 using x+1, but only for the function f(x). However, the original value of the argument 'x' remains the same.

2.7 FLOW OF EXECUTION OF PROGRAM USING MAIN() AS A FUNCTION

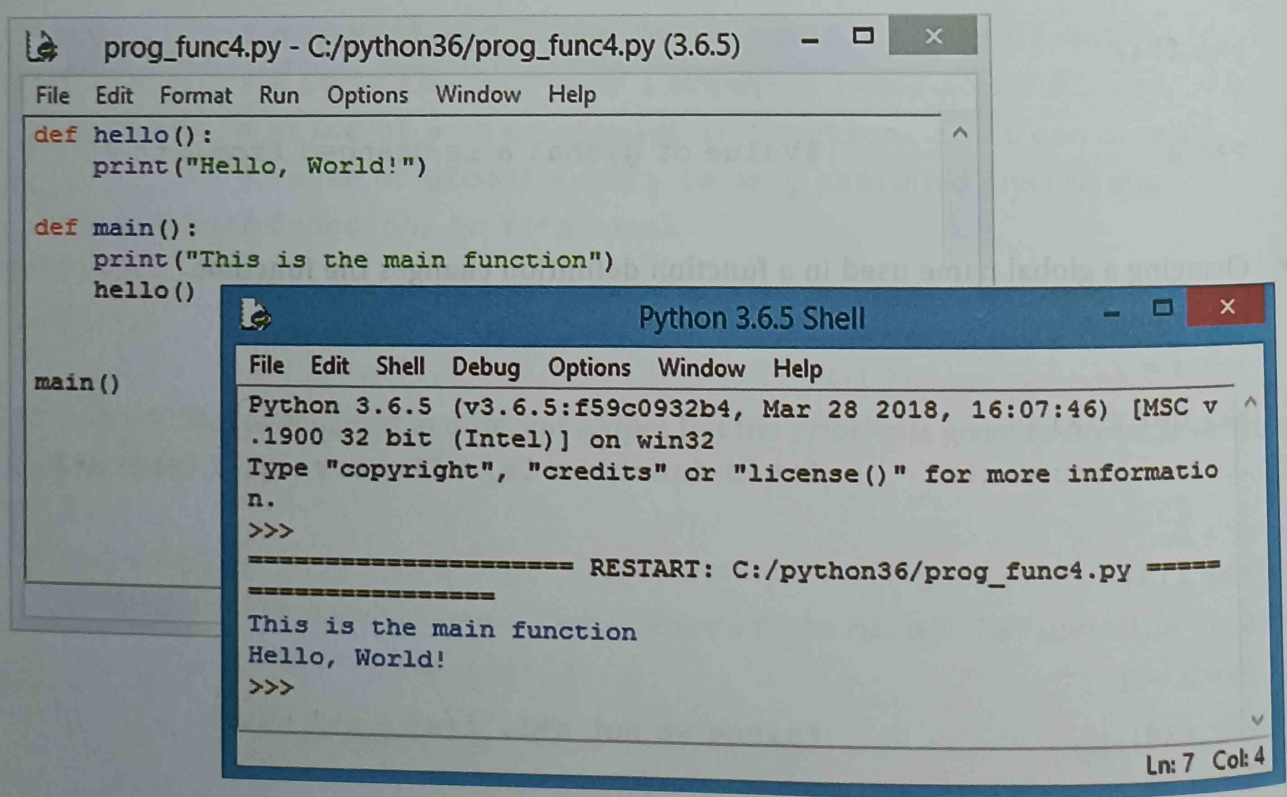
Including a main() function is not mandatory in Python. It can structure our Python programs in a logical way that puts the most important components of the program into one function. It can also make our programs easier for non-Python programmers to read.

Practical Implementation–15

We will start with adding a main() function to the 'prog_func4.py' program below. We will first define hello() function and then define a main() function.

After defining main() function, we will give print() statement so as to ensure the execution of main().

After defining the main() function, finally, at the bottom of the program, we will call the main() function as shown below:



In the given example, we have called the main() explicitly either from the shell or from within the script after giving the calling function for main() function in the end.

This limitation of calling the main() function can be resolved by including the script for main() within the program and then calling it using the syntax defined below:

```
if__name__=='__main__':
    main()
```

Let us take an example to implement the above concept.

Practical Implementation–16

Write a program to compute the area of a rectangle on the basis of length and breadth inputted by the user as the arguments to this function.

rect_func.py - C:/python36/rect_func.py (3.6.5)

File Edit Format Run Options Window Help

```
def areaRectangle(length, breadth=1):
    '''
    Objective: To compute the area of rectangle
    Input Parameters: length, breadth - numeric value
    Return Value: area - numeric value
    '''
    area = length * breadth
    return area
def main():
    '''
    Objective: To compute the area of rectangle based on user input
    Input Parameter: None
    Return Value: None
    '''
    print('Enter the following values for rectangle:')
    lengthRect = int(input('Length : integer value: '))
    breadthRect = int(input('Breadth : integer value: '))
    areaRect = areaRectangle(lengthRect, breadthRect)
    print('Area of rectangle is', areaRect)

if __name__ == '__main__':
    main()
```

Python 3.6.5 Shell

File Edit Shell Debug Options Window Help

```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bi
t (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/python36/rect_func.py ================
====
Enter the following values for rectangle:
Length : integer value: 50
Breadth : integer value: 30
Area of rectangle is 1500
>>>
```

Ln: 9 Col: 4

Let us try to understand the above code. Every Python module has a built-in variable called _name_ (underscores placed before and after 'name') containing the name of the module. When the module itself is being run as the script, this variable _name_ is assigned the string '__main__' designating it to be a __main__ module.

Python checks whether the name of the current module is __main__. This being true, the expression _name_=='__main__' evaluates to true and the function main() gets executed.

As the main() gets executed, the arguments for length and breadth are accepted as the input by the user and call to function areaRectangle() gets invoked. Thus, the area of the rectangle gets calculated inside this function and returns back to the calling main() function where it gets displayed.

It is to be remembered that a function does not execute until it is invoked, whether it is main() or any other user-defined function.

2.8 FLOW OF EXECUTION OF PROGRAM CONTAINING FUNCTION CALL

Execution always begins at the first statement of the program. Statements are executed one at a time, starting from the top to the bottom. Function definition does not alter the flow of execution of a program, as the statement inside the function is not executed until the function is called.

On a function call, instead of going to the next statement of the program, the control jumps to the body of the function, executes all statements of the function, starting from the top to the bottom and then comes back to the point where it left off. This remains simple till a function does not call another function. Similarly, in the middle of a function, the program might have to execute statements of the other function and so on.

However, Python is good at keeping track of the execution so that each time a function completes, the program picks up from the place where it left off, until it gets to the end of the program where it terminates. For example,

```
1.  def add(a,b):
2.  c = a+b
3.  return  c
4.  def sub(a,b):
5.  if (a > b):
6.  c = a-b
7.  else:
8.  c = b – a
9.  return c
10.
11.
12. X = 20
13. Y = 10
14. Z = add(X,Y)
15. print(Z)
16. i = sub(X,Y)
17. print(i)
```

Upon execution of the above code, the flow of execution will be:

$1 \rightarrow 4 \rightarrow 12 \rightarrow 13 \rightarrow 14 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 14 \rightarrow 15 \rightarrow 16 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 16 \rightarrow 17$

However, the flow of execution in case of both the functions shall be:

- go to line $1 \rightarrow 2 \rightarrow 3 \rightarrow 14 \rightarrow 15$: for add(a,b)
- go to line $5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 10 \rightarrow 16 \rightarrow 17$: for sub(a,b)

Explanation:

1. Execution begins from Statement 1. Here, interpreter encounters a function definition, which will not be executed till the function is called.
2. Then the interpreter's control reaches Statement 4, where the second function definition is encountered which gets skipped.
3. After this, the control reaches Statements 12 and 13 and gets executed.
4. Then, in succession, Statement 14 is executed where the interpreter encounters a function call and jumps to the function header and starts executing Statement 1, 2 and 3. Statement 3 is the return statement which shall return the value back to the function call statement.
5. Thus, the returned value is received by variable Z in the Statement 14 (from where it was called) and gets printed upon the execution of Statement 15.
6. Now, the interpreter executes Statement 16 in succession which is again a function call. The control then jumps to Statement 4 and executes Statements 5, 6, 7, 8, and 9 respectively. The control goes back to Statement 16 with returning value and finally executes Statement 17 and terminates the program.

2.9 RECURSION

Recursion is one of the most powerful tools in a programming language. It is a function calling itself again and again. Recursion is defined as defining anything in terms of itself. In other words, it is a method in which a function calls itself one or more times in its body.

Conditions for Implementing Recursion

1. There must be a terminating condition for the problem which we want to solve with recursion. This condition is called the base case for that problem.
2. There must be an if condition in the recursive routine. This if clause specifies the terminating condition of the recursion.
3. Reversal in the order of execution is the characteristic of every recursive problem, *i.e.*, when a recursive program is taken for execution, the recursive function calls are not executed immediately. They are pushed onto stack as long as the terminating condition is encountered. As soon as the recursive condition is encountered, the recursive calls which were pushed onto stack are removed in reverse order and executed one by one. It means that the last call made will execute first, then the second-last call will execute and so on until the stack is empty.
4. Each time a new recursive call is made, a new memory space is allocated to each local variable used by the recursive routine. In other words, during each recursive call to recursive function, the local variable of the recursive function gets a different set of values, but with the same name.
5. The duplicated values of the local variables of a recursive call are pushed onto the stack with its respective call, and all these values are available to the respective function call when it is popped off from the stack.

Disadvantages of using Recursion

1. It consumes more storage space because the recursive calls along with local variables are stored on the stack.
2. The computer may run out of memory if the recursive calls are not checked.
3. It is less efficient in terms of speed and execution time.

How Recursion Works

Recursion is implemented by dividing the recursive problem into two parts. A useful recursive function usually consists of the following parts:

- **Terminating part** (for the smallest possible problem) when the function returns a value directly.
- **Recursive part** which contains one of the more recursive calls on smaller parts of the problem.

Let us write a recursive function to implement the power() function. Before writing the recursive function, let us understand the arithmetical approach to solve this problem.

X^n = X*X*X*X.....*X n times

Or

X^n = X*(X*X*X.....X*X) n–1 times

If we know the value of X^{n-1}, we can easily calculate X^n

Or

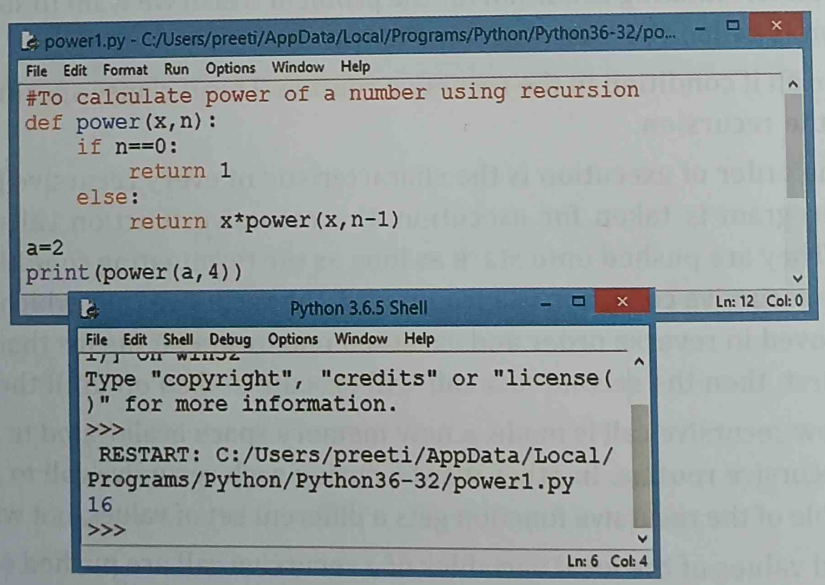
X^n = X* (X*(X*X....X*X) n–2 times

If we know the value of X^{n-2}, we can calculate X^{n-1} and, thus, calculate X^n

X^n=X(X(X.....n–2))

And this process will continue until the innermost expression becomes X^1. Now, it is easy to write the recursive function for X^n.

Example 11: To calculate power of a number using recursion.



Explanation:

In each call, the power() function passes the actual parameters to the version being called. The value of x is same for each power function call, but the value of n is decremented by 1 for each call until n becomes zero. When n becomes 0, the function power() starts returning a value x. This value of power() function is passed back to the previous version that made this call and this process of returning values will continue until the value of the power can be passed to the original call. When we define a function recursively, then there must exist such condition that is responsible for its termination.

Let us see some more examples on Recursion.

Practical Implementation–17

Write a function to calculate the sum of first 'n' natural numbers using Recursion.

```
sum_recur.py - C:/Users/preeti/AppData/Local/Programs/Python/Python36-32/sum_recur.py (3.6.5)
File Edit Format Run Options Window Help
#Function to find the sum of first 'n' Natural numbers using Recursion
def recur_sum(n):
    if n <= 1:
        return n
    else:
        return n+recur_sum(n-1)
num = int(input("Enter a number :"))

if num < 0:
    print("Enter a positive number")
else:
    print("The sum is :",recur_sum(num))
```

```
>>>
 RESTART: C:/Users/preeti/AppData/Local/
Programs/Python/Python36-32/sum_recur.py
Enter a number :6
The sum is : 21
>>>
Ln: 7 Col: 4
```

Practical Implementation–18

Write a function to display Fibonacci Series using Recursion.

```
*recur_fibo.py - C:/Users/preeti/AppData/Local/Programs/Python/Python36-32/recur_fi...
File Edit Format Run Options Window Help
#Function to display Fibonacci Series using Recursion

def recur_fibo(n):
    if n <= 1:
        return n
    else:
        return(recur_fibo(n-1) + recur_fibo(n-2))

nterms = int(input("How many terms required in the series: "))
print("Fibonacci Sequence generated is :")
for i in range(nterms):
    print(recur_fibo(i))
```

```
>>>
 RESTART: C:/Users/preeti/AppData/Local/Progra
ms/Python/Python36-32/recur_fibo.py
How many terms required in the series: 10
Fibonacci Sequence generated is :
0
1
1
2
3
5
8
13
21
34
>>>
Ln: 17 Col: 4
```

Explanation:

A Fibonacci Sequence is a sequence of integers in which the first two terms are 0 and 1, and all other terms of the sequence are obtained by adding their preceding two numbers. The above program takes the number of terms and determines the Fibonacci Series using Recursion up to that term.

This is achieved by first taking the number of terms from the user and storing it in a variable. This number is passed as an argument to the recursive functions recur_fibo().

The base condition on the basis of which the loop shall execute is defined as the number less than or equal to 1. Otherwise, the function call is made recursively with the argument as the number minus 1 added to the function called recursively with the argument as the number minus 2.

Finally, the returned value which is the Fibonacci Series is printed using a for loop.

Practical Implementation–19

Write a Recursive function to calculate the Factorial of a number.

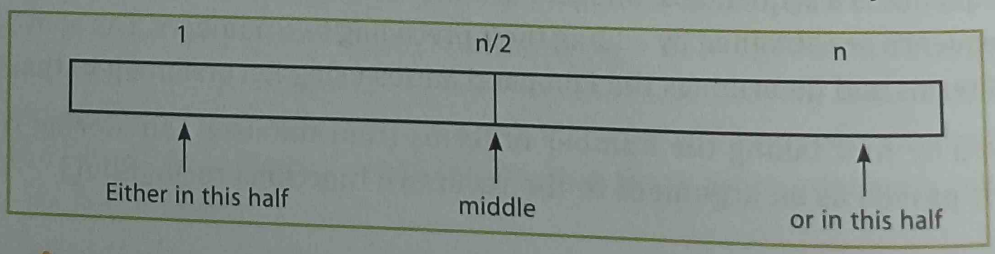
```
#Function to find the Factorial of a Number using Recursion

def recur_factorial(n):
    if n == 1:
        return n
    else:
        return n*recur_factorial(n-1)

num = int(input("Enter a number :"))

if num < 0:
    print("Sorry, factorial for a negative number does not exist")
elif num == 0:
    print("The factorial of 0 is 1")

else:
    print("The factorial of",num,"is",recur_factorial(num))
```

```
Python 3.6.5 Shell
in32
Type "copyright", "credits" or "license()" f
or more information.
>>>
 RESTART: C:/Users/preeti/AppData/Local/Prog
rams/Python/Python36-32/recur_fact.py
Enter a number :5
The factorial of 5 is 120
>>>
 RESTART: C:/Users/preeti/AppData/Local/Prog
rams/Python/Python36-32/recur_fact.py
Enter a number :10
The factorial of 10 is 3628800
>>>
```

Practical Implementation–20 (Binary Search in lists/arrays)

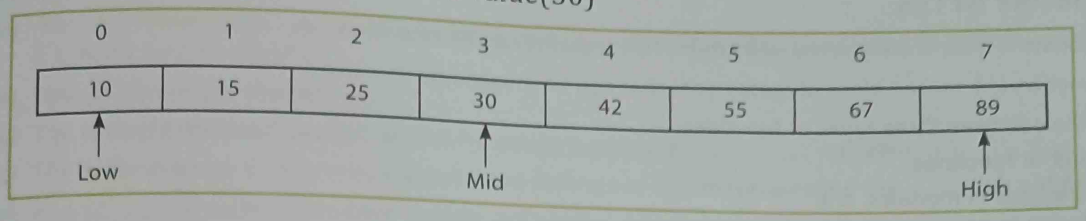
Write a Recursive program to implement binary search in arrays/lists.

In binary search technique, the entire sequence of numbers (list/array) is divided into two parts and the middle element of list/array is extracted. This technique is also known as **divide and conquer technique**. To search an element, say 'val' in a sorted array (suppose in ascending order), the val is compared with the middle element first. If the searched value is equal to the middle value, then the index number and position number of this element is returned. If the val is greater than the middle element, the second half of the array becomes the new segment to be traversed; if the val is less than the middle element, the first half becomes the new segment to be scanned. The same process is repeated until val is found or the segment is reduced to the single element and still val is not found (in case of an unsuccessful search).

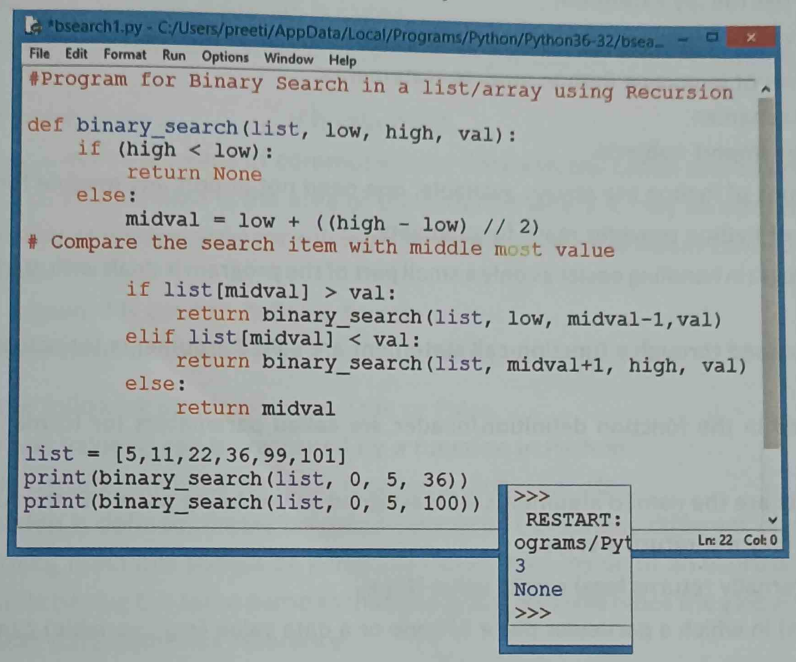


For example,

- If searched value is 25, then 25<mid value(30)



- If searched value is 42, then 42>mid value(30)
- If searched value is 30, then 30=mid value(30)



2.9.1 Recursion vs Iteration

Recursion and iteration are inter-related concepts. However, even for problems that can be handled equally well using both the techniques, there is a subtle difference in the manner loops and method calls are handled by a programming language compiler.

When a loop is executed repeatedly, it uses the same memory locations for variables and repeats the same unit of code. On the other hand, recursion, instead of repeating the same unit of code and using the same memory locations for variables, allocates fresh memory space for each recursive call.

Therefore, we can say that any problem that can be solved through iteration can be solved using recursion as well and vice versa. But the limitation in case of recursion is that it involves an additional cost in terms of the space used in RAM by each recursive call to a function and in time used by the function call.

Also, because of extra memory space occupied by stack manipulation, recursive functions often run slower and use more memory than their iterative counterparts. At the same time, in certain situations, recursion sometimes makes the code easier to understand.

Therefore, selection between the two approaches (*i.e.*, recursion and iteration) should be made on the basis of the problem under consideration.

MEMORY BYTES

- A module is a separately saved unit whose functionality can be reused at will.
- A function is a named block of statements that can be invoked by its name.
- Python can have three types of functions:
  - Built-in functions,
  - Functions in modules, and
  - User-defined functions.
- A Python module can contain objects like docstrings, variables, constants, classes, objects, statements, functions.
- A Python module has the .py extension.
- A Python module can be imported in a program using import statement.
- There are two forms of importing Python module statements:
  (i) import <modulename>
  (ii) from <module> import <object>
- The built-in functions of Python are always available; one need not import any module for them.
- The math module of Python provides math functionality.
- Functions make program handling easier as only a small part of the program is dealt with at a time, thereby avoiding ambiguity.
- The values being passed through a function-call statement are called arguments (or actual parameters or actual arguments).
- The values received in the function definition/header are called parameters (or formal parameters or formal arguments).
- Keyword arguments are the named arguments with assigned values being passed in the function-call statement.
- A function may or may not return a value.
- A void function internally returns legal empty value None.
- The program part(s) in which a particular piece of code or a data value (*e.g.*, variable) can be accessed is known as Variable Scope.
- In Python, broadly, scopes can either be global scope or local scope.
- A local variable having the same name as that of a global variable hides the global variable in its function.
- A file that contains a collection of related functions grouped together and other definitions is called module.
- A search path is the list of directories that the interpreter searches before importing a module.
- A library is just a module that contains some useful definitions.
- The random() function generates a floating point random value from 0 to <1.
- A function is said to be recursive if it calls itself.
- There are two cases in each recursive function—the recursive case and the base case.
- An infinite recursion is when a recursive function calls itself endlessly.
- If there is no base case or if the base case is never executed, infinite recursion occurs.
- Iteration uses the same memory space for each pass contrary to recursion where fresh memory is allocated for each successive call.
- Recursive functions are relatively slower than their iterative counterparts.
- Some commonly used recursive algorithms are factorial, gcd, fibonacci series printing, binary search, etc.
- String can be passed to a function as argument but it is used as pass by value.
- Tuple value cannot be modified in a function.
- In Python, everything is an object, so the dictionary can be passed as an argument to a function like other variables are passed.

OBJECTIVE TYPE QUESTIONS

1. Fill in the blanks.
   (a) A set of instructions/operations which is a part of a program and can be executed independently to do a specific task is called a ..... .
   (b) Python passes parameters by ..... .
   (c) The variable declared outside all the functions is called a ..... variable.
   (d) The order in which statements are executed during a program run is called the ..... of execution.
   (e) A ..... is a file containing Python definitions and statements intended for use in other Python programs.
   (f) Functions that do not explicitly return a value return the special object ..... .
   (g) The first line of a function definition is called ..... .
   (h) The function which is written by the programmer as per his/her requirements is known as ..... function.
   (i) A function is said to be ..... if it calls itself.
   (j) ..... act as a means of communication between the called and calling function.
   (k) The ..... of a variable is the area of the program where it may be referenced.
   (l) The terminating condition used for solving a problem using recursion is termed as the ..... for that problem.
   (m) ..... keyword is used to define a function.
   (n) Function name must be followed by ..... and ..... .

2. State whether the following statements are True or False.
   (a) More than one value(s) can be returned by a function in Python.
   (b) The variable declared inside a function is called a global variable.
   (c) Once a function is defined, it may be called only once from many different places in a program.
   (d) Value returning functions should be generally called from inside of an expression.
   (e) A local variable having the same name as that of a global variable hides the global variable in its function.
   (f) Python passes parameters by reference.
   (g) Parameters specified within a pair of parentheses in the function definition are the actual parameters or non-formal parameters.
   (h) A function in Python is used by invoking it via a function call.
   (i) Built-in functions are created by users and are not a part of the Python library.
   (j) The first line of a function header begins with def keyword and eventually ends with a colon (:).
   (k) Recursive functions are faster than their iterative counterparts.
   (l) Recursion is defined as defining anything in terms of itself.
   (m) Function can alter only mutable data types.

3. Multiple Choice Questions (MCQs)
   (a) A function in Python begins with which keyword?
      (i) void (ii) return (iii) int (iv) def
   (b) Name the statement that sends back a value from a function.
      (i) print (ii) input (iii) return (iv) None
   (c) Functions that do not return any value are known as:
      (i) fruitful functions (ii) void functions
      (iii) library functions (iv) user-defined functions
   (d) A variable created or defined within a function body is classified as:
      (i) local (ii) global (iii) built-in (iv) instance
   (e) Which of the following arguments works with implicit values that are used if no value is provided?
      (i) keyword (ii) required (iii) variable-length (iv) default

(f) Which values are used by the functions to communicate information back to the caller?

(i) local (ii) global (iii) return (iv) random

(g) What is the output of the program given below?

```
x = 50
def func(x):
    x = 2
func(x)
print('x is now', x)
```

(i) x is now 50 (ii) x is now 2 (iii) x is now 100 (iv) Error

(h) Which is the most appropriate definition for recursion?

(i) A function that calls itself

(ii) A function execution instance that calls another execution instance of the same function

(iii) A class method that calls another class method

(iv) An in-built method that is automatically called

(i) Fill in the line of code for calculating the factorial of a number:

```
def fact(num):
    if num == 0:
    return 1
else:
    return _____________________
```

(i) num*fact(num–1) (ii) (num–1)*(num–2)

(iii) num*(num–1) (iv) fact(num)*fact(num–1)

(j) Which of the following statements is false about recursion?

(i) Every recursive function must have a base case.

(ii) Infinite recursion can occur if the base case isn't properly mentioned.

(iii) A recursive function makes the code easier to understand.

(iv) Every recursive function must have a return value.

(k) What is the output of the following snippet?

```
def fun(n):
    if (n > 100):
        return n - 5
    return fun(fun(n+11))
print(fun(45))
```

(i) 50 (ii) 100 (iii) 74 (iv) Infinite loop

(l) What happens if the base condition isn't defined in recursive programs?

(i) Program gets into an infinite loop

(ii) Program runs once

(iii) Program runs n number of times, where n is the argument given to the function

(iv) An exception is thrown

SOLVED QUESTIONS

1. What is a function? How is it useful?

Ans. A function in Python is a named block of statements within a program to perform a specific task. For example, the following program has a function within it, namely greet_msg ( ).

```
# prog1.py
def greet_msg():
    print("Hello there!!")
    name = input("Your name :")
    print(name, greet_msg())
```

Functions are useful as they can be reused anywhere through their function call statements. So, for the same functionality, one need not to rewrite the code every time it is needed, rather through functions it can be used again and again without rewriting the code.

2. Python has certain functions that you can readily use without having to write any special code. What type of functions are these?

Ans. The predefined functions that are always available for use are known as Python's built-in functions. Examples of some built-in functions are: len(), type(), int(), input(), etc.

3. What is an argument? Give an example.

Ans. An argument is data passed to a function through function call statement. It is also called actual argument or actual parameter. For example, in the statement print(math.sqrt(25)) the integer 25 is an actual argument.

4. Find the error:

```
def minus(total, decrement)
output = total - decrement
return output
```

Ans. The function's header has a colon missing at the end and the next statement must be indented. So, we need to add colon (:) at the end of the header line and the next line will be indented.

Thus, the corrected code will be:

```
def minus(total, decrement):
    output = total - decrement
    return output
```

Indented statements

5. What is the utility of Python standard library's math module and random module?

Ans. (i) math module: The math module is used for math-related functions that work with all number types except for complex numbers. For example, pow(), sqrt(), etc.

(ii) random module: The random module is used for different random number generator functions. For example, randint(), random(), etc.

6. What is Python module? What is its significance?

Ans. A "module" is a chunk of Python code that exists in its own (.py) file and is intended to be used by Python code outside itself.

Modules allow one to bundle together code in a form in which it can easily be used later. The Modules can be "imported" in other programs so the functions and other definitions in imported modules become available to code that imports them.

7. What is the utility of the built-in function help()?

Ans. Python's built-in function help() is very useful. When it is provided with a program name or a module-name or a function-name as an argument, it displays the documentation of the argument as help. It also displays the docstring within its passed-argument's definition. For example,

```
>>> help(math)
```

will display the documentation related to module math.

It can even take function name as argument. For example,

```
>>> help(math.sqrt())
```

The above code will list the documentation of math.sqrt() function only.

8. Write the output of the following commands:

(i) int('462.3')
(ii) int("IEEE")
(iii) float(8.0−6/3)
(iv) float("398.678)
(v) float('8.0+'), float('2−'), float('3')
(vi) z=int("82")+ float("90")−float("30")
(vii) max(−8, −10, 2, −9, 3)
(viii) min (8, 6, 3, −2, 1)
(ix) eval('32/4 *2')
(x) abs(−80.6)
(xi) len({10:23, 20: 34, 30:100, 40: 200})
(xii) round(1926.486,1)
(xiii) round(32609.432, −3)
(xiv) round(892.6329, 2)
(xv) round(362.96)

Ans. (i) 462
(ii) Error
(iii) 6.0
(iv) 398.678
(v) Error
(vi) 142.0
(vii) 3
(viii) −2
(ix) 16.0
(x) 80.6
(xi) 4
(xii) 1926.5
(xiii) 33000.0
(xiv) 892.63
(xv) 363

9. Write a Python program to demonstrate the concept of variable length argument to calculate sum and product of the first 10 numbers.

Ans.
```
def sum10(*n):
    total=0
    for i in n:
        total=total + i
    print("Sum of first 10 Numbers:",total)

def product10(*n):
    pr=1
    for i in n:
        pr=pr * i
    print("Product of first 10 Numbers:",pr)
sum10(1,2,3,4,5,6,7,8,9,10)
product10(1,2,3,4,5,6,7,8,9,10)
```

10. Write a function compute_volume() that:

(i) Asks the user to input a diameter of a sphere (centimetres, inches, etc.)

(ii) Sets a variable called radius to one-half of that number.

(iii) Calculates the volume of a sphere using this radius and the formula:
Volume = 4/3 pi*r**3

(iv) Prints a statement estimating the volume of the sphere; include the appropriate unit information in litres or quarts.
(Note that there are 1000 cubic cm in a litre and 57.75 cubic inches in a quart.)

(v) Returns this same amount as the output of the function.

Ans.
```
def compute_volume():
    import math
    diameter =float(input("Please enter the diameter in inches:"))
    radius = diameter/2
    volume_in_cubic_inches =((4/3) * math.pi) * (radius ** 3)
    volume_in_quarts = volume_in_cubic_inches/57.75
    print("The sphere contains", volume_in_quarts, 'quarts')
    return(volume_in_quarts)
```

11. What is the output of the following Python code?

```
x=60
def f1():
    global x
    print('x is: ", x)
    x=10
    print("Changed global x is: ", x)
    x+=5
f1()
print('Value of x is: ', x)
```

Ans. x is: 60
Changed global x is: 10
Value of x is: 15

12. Find and write the output of the following Python code:

```
def Change(P,Q=30):
    P=P+Q
    Q=P-Q
    print(P,"#",Q)
    return P
R=150
S=100
R=Change(R,S)
print(R,"#",S)
S=Change(S)
```

Ans.
```
250 # 150
250 # 100
130 # 100
```

13. What possible output(s) is/are expected to be displayed on screen at the time of execution of the given code?

```
import random
POINTS = [30,50,20,40,45]
BEGIN = random.randint(0,3)
LAST = random.randint(2,3)
for C in range(BEGIN,LAST+1):
    print(POINTS[C],"#", end ="")
```

Also specify the maximum value that can be assigned to each of the variables Last and Begin.

(i) 30#50#20 (ii) 50#20#40
(iii) 20#40#45 (iv) 20#40#30

Ans. (i), (ii) and (iv)
Begin(min value)=0 Last(max value)=3

14. What possible output(s) is/are expected to be displayed on screen at the time of execution of the program from the following code? Also specify the maximum and minimum value that can be assigned to pos variable.

```
import random
languages=['Python', 'Java', 'PHP', 'HTML', 'C++']
pos =random.randint(1,4)
for c in range(0, pos+1):
        print (languages[c],end="&")
```

(i) Python&Java&PHP&HTML&C++& (ii) Java&PHP&HTML&C++&
(iii) Python&Java& (iv) Python&Java&PHP&

Ans. (i), (iii) & (iv)
Maximum Value = 4
Minimum Value = 1

15. Identify the Local and Global variable in the following code.

```
x=100
def myfunc(a):
    k=a
    print(k,a)
p=(0,1,2,3,4)
myfunc(p)
print(x)
```

Ans.
```
x= 100                  # Global var x
def myfunc(a):          # Local var a
    k=a                 # Local var k
    print(k,a)
p=(0,1,2,3,4)           # Global var p
myfunc(p)
print(x)
```

16. What is the significance of having functions in a program?

Ans. Creating functions in a program is very useful. It offers the following advantages:

(i) **The program is easier to understand.**
Main block of the program becomes compact as the code of functions is not a part of it and, thus, it is easier to read and understand.

(ii) **Redundant code is at one place, so making changes is easier.**
Instead of writing code again when we need to use it more than once, we can write the code in the form of a function and call it more than once. If we later need to change the code, we will need to change it at one place only. Thus, it saves our time also.

(iii) **Reusable functions can be put in a library in modules.**
We can store the reusable functions in the form of a module. These modules can be imported and used when needed in other programs.

17. From the program code given below, identify the parts mentioned below:

```
def processNumber(x):
    x=72
    return x + 3
y = 54
res = processNumber(y)
```

Identify these parts: function header, function call, actual arguments, formal parameters, function body, main program

Ans.

Function header	: def processNumber(x):
Function call	: processNumber(y)
Actual arguments	: y
Formal Parameters	: x
Function body	: x = 72 return x + 3
Main program	: y = 54 res = processNumber(y)

18. Find errors, if any, in the following code:

(i)
```
import math
math.power(4,2)
```

(ii)
```
import random
random. randint(4,16,4)
```

(iii)
```
import random
random. random(2,10)
```

(iv)
```
import math
math.sqrt(-256)
```

Ans. (i) **Error:** function name is pow() instead of power

(ii) **Error:** randint() function does not take step value as an argument.

(iii) **Error:** random() function does not take any argument.

(iv) **Error:** sqrt() function does not take negative number as an argument.

19. Name the built-in mathematical function/method that is used to return greatest common divisor of x and y.

Ans. gcd (x, y) which is a part of math module in Python.

20. Trace the flow of execution for the following programs:

(a)
```
1. def power (b, p):
2.     r = b ** P
3.     return r
4.
5. def calcSquare(a):
6.     a = power(a, 2)
7.     return a
8.
9. n = 5
10. result = calcSquare(n)
11. print(result)
```

(b)
```
1. def increment(x):
2.     x= x + 1
3.
4. # main() program
5. x = 3
6. print(x)
7. increment (x)
8. print(x)
```

(c)
```
1. def increment(x):
2.     z = 45
3.     x = x + 1
4.     return x
5.
6. # main()
7. Y = 3
8. print(y)
9. y = increment(y)
10. print(y)
11. q = 77
12. print (q)
13. increment (q)
14. print(q)
15. print(x)
16. print(z)
```

Ans. (a) 1->5->9->10->5->6->1->2->3->6->7->10->11

(b) 1->5->6->7->1->2->8

(c) 1->7->8->9->1->2->3->4->9->10->11->12->13->1->2->3->4->14->15->16

21. Trace the flow of execution for the following code and predict the output produced by it.

```
1. def power (b, p):
2.     y = b ** p
3.     return y
4.
5. def calcSquare (x) :
6.     a = power (x, 2)
7.     return a
8.
9. n = 5
10. result = calcSquare (n) + power (3, 3)
11. print(result)
```

Ans. Flow of execution for the above code will be:

1->5->9->10->5->6->1->2->3->6->7->10->1->2->3->10->11

The output produced by the above code will be:

52

22. What is the difference between local variable and global variable? Also, give a suitable Python code to illustrate both.

Ans. The differences between a local variable and global variable have been given below:

S.No.	Local Variable	Global Variable
1.	It is a variable which is declared within a function or within a block.	It is a variable which is declared outside all the functions or in a global space.
2.	It cannot be accessed outside the function but only within a function/block of a program.	It is accessible throughout the program in which it is declared.

For example, in the following code x, xCubed are global variables and n and cn are local variables.

```
                                #n and cn are local variables
def cube(n):
    cn = n * n * n
    return cn
                                # x is a global variable
x = 10
                                # xCubed is a global variable
xCubed = cube(x)
print(x, "cubed is", xCubed)
```

23. What is the output of the following code?

```
a = 1
def f():
    a = 10
print(a)
```

Ans. The code will print global variable 1 on the console (Python shell) as it is accessible from anywhere and a=10 will print only when the function f() is invoked as it is a local variable.

24. What will be the output of the following code?

```
def interest(prnc, time=2, rate = 0.10) :
    return(prnc * time * rate)
print(interest(6100, 1))
print(interest(5000, rate =0.05))
print(interest(5000, 3, 0.12))
print(interest(time = 4, prnc = 5000))
```

Ans.

```
610.0
500.0
1800.0
2000.0
```

25. Rewrite the following code in Python after removing all syntax errors. Underline each correction done in the code:

```
Def func(a):
S,m,n=0
for i in (0,a):
if i%2 =0:
s=s+i
else if i%5= =0
    m=m+i
    else:
n=n+i
print(s,m,n)
    func(15)
```

Ans.

```
def func(a): #def
    s=m=n=0 #local variable
    for i in range (0,a): #indentation and range() function missing
        if i%2==0: #indentation and equality operator
            s=s+i
        elif i%5==0: #elif and colon
            m=m+i
        else:
            n=n+i
    print(s,m,n) #indentation
func(15)         #indentation
```

26. Which line in the given code(s) will not work and why?

```
def Interest(P,R,T=7):
    I = (P*R*T)
    print(I)
print(Interest(20000, .08, 15))          #Line 1
print(Interest(T=10, 20000, 0.75))       #Line 2
print(Interest(50000, .07))              #Line 3
print(Interest(P=10000, R=.06, Time=8))  #Line 4
print(Interest(80000, T=10))             #Line 5
```

Ans. Line 2: Positional argument must not be followed by keyword argument, *i.e.*, positional argument must appear before a keyword argument.

Line 4: There is no keyword argument with the name 'Time'

Line 5: Missing value for positional argument, R

27. Is return statement optional? Compare and comment on the following two return statements:

```
return
return val
```

Ans. The return statement is optional only when the function is void, or we can say that when the function does not return a value. A function that returns a value must have at least one return statement.

From the given two return statements, the statement:

```
return
```

is not returning any value. Rather, it returns the control to caller along with empty value None. And the statement:

```
return val
```

is returning the control to caller along with the value contained in variable val.

28. Divyansh, a Python programmer, is working on a project which requires him to define a function with name CalculateInterest(). He defines it as:

```
def CalculateInterest(Principal,Rate=.06,Time): # code
```

But this code is not working. Can you help Divyansh to identify the error in the above function and with the solution?

Ans. Yes, here non-default argument is followed by default argument which is wrong as per Python's syntax.

Solution:

(a) The first way is by putting Rate as the last argument:
def CalculateInterest(Principal,Time, Rate=.06):

(b) Or, give any default value to Time also as:
def CalculateInterest(Principal,Rate=.06,Time=12):

29. Write a function listchange (Arr) in Python, which accepts a list Arr of numbers, the function will replace the even number by value 10 and multiply odd number by 5.

Sample Input Data of the list is:

```
a=[10,20,23,45]
listchange(a,4)
```

Output: [10, 10, 115, 225]

Ans.

```
def listchange(arr,n):
    l=len(arr)
    for a in range(l):
        if(arr[a]%2==0):
            arr[a]=10
        else:
            arr[a]=arr[a]*5
a=[10,20,23,45]
listchange(a)
print(a)
```

30. What will be the output of the following code?

```
def Alter(M,N=50):
    M = M + N
    N = M - N
    print(M,"@",N)
    return M
A=200
B=100
A = Alter(A,B)
print(A,"#",B)
B = Alter(B)
print(A,"@",B)
```

Ans. 300 @ 200
300 # 100
150 @ 100
300 @ 150

31. What will be the output of the following code?

```
def drawline(char='$',time=5):
    print(char*time)
drawline()
drawline('@',10)
drawline(65)
drawline(chr(65))
```

Ans. $$$$$
@@@@@@@@@@
325
AAAAA

32. What will be the output of the following code?

```
def Fun1(mylist):
    for i in range(len(mylist)):
        if mylist[i]%2==0:
            mylist[i]/=2
        else:
            mylist[i]*=2
list1 =[21,20,6,7,9,18,100,50,13]
Fun1(list1)
print(list1)
```

Ans. [42, 10.0, 3.0, 14, 18, 9.0, 50.0, 25.0, 26]

33. Write a Python function to find the maximum of three numbers using recursion:

Ans.

```
def max_of_two(x, y):
    if x > y:
        return x
    return y
def max_of_three(x, y, z):
    return max_of_two(x, max_of_two(y, z))
print(max_of_three(8, -4, 10))
```

34. Write a Python function to sum all the numbers in a list.
Sample List: [4, 6, 3, 5, 6]
Expected Output : 24

Ans.

```
def sum(numbers):
    total = 0
    for x in numbers:
        total += x
    return total
print(sum([4, 6, 3, 5, 6]))
```

35. Write a Python program to reverse a string.
Sample String : "python123"
Expected Output : "321nohtyp"

Ans.

```
def string_reverse(str1):
    rstr1 = ''
    index = len(str1)
    while index > 0:
        rstr1 += str1[index - 1]
        index = index - 1
    return rstr1
print(string_reverse('python123'))
```

36. Write a Python function that accepts a string and calculates the number of uppercase letters and lowercase letters.

Sample String : PythonProgramminG

Expected Output :

```
Original String : Python ProgramminG
No. of Uppercase characters : 3
No. of Lowercase characters : 14
```

Ans.
```
def string_test(s):
    d={"UPPER_CASE":0, "LOWER_CASE":0}
    for c in s:
        if c.isupper():
            d["UPPER_CASE"]+=1
        elif c.islower():
            d["LOWER_CASE"]+=1
        else:
            pass
print("Original String : ", s)
print("No. of Uppercase characters : ", d["UPPER_CASE"])
print("No. of Lowercase characters : ", d["LOWER_CASE"])
string_test('PythonProgramminG')
```

37. Write a Python program to print the even numbers from a given list.

Sample List : [1, 2, 3, 4, 5, 6, 7, 8, 9]

Expected Result : [2, 4, 6, 8]

Ans.
```
def is_even_num(l):
    enum = []
    for n in l:
        if n % 2 == 0:
            enum.append(n)
    return enum
print(is_even_num([1, 2, 3, 4, 5, 6, 7, 8, 9]))
```

38. Write a Python function to check whether a number is perfect or not. According to Wikipedia, in number theory, a perfect number is a positive integer that is equal to the sum of its proper positive divisors, that is, the sum of its positive divisors excluding the number itself (also known as its aliquot sum). Equivalently, a perfect number is a number that is half the sum of all of its positive divisors (including itself).

Example: The first perfect number is 6, because 1, 2, and 3 are its proper positive divisors, and 1 + 2 + 3 = 6. Equivalently, the number 6 is equal to half the sum of all its positive divisors: (1 + 2 + 3 + 6) / 2 = 6. The next perfect number is 28 = 1 + 2 + 4 + 7 + 14. This is followed by the perfect numbers 496 and 8128.

Ans.
```
def perfect_number(n):
    sum = 0
    for x in range(1, n):
        if n % x == 0:
            sum += x
    return sum == n
print(perfect_number(6))
```

39. Write a Python program to make a chain of function decorators (bold, italic, underline, etc.) in Python.

Note: Function decorators are a very useful tool in Python as they allow us to wrap a function into another function in order to extend the behaviour of wrapped function. Here, functions are taken as argument into another function and called inside the wrapper function. '@' sign before the function name is used to call the function.

Ans.
```
def make_bold(fn):
    def wrapped():
        return "<b>" + fn() + "</b>"
    return wrapped
def make_italic(fn):
    def wrapped():
        return "<i>" + fn() + "</i>"
    return wrapped
def make_underline(fn):
    def wrapped():
        return "<u>" + fn() + "</u>"
    return wrapped
@make_bold
@make_italic
@make_underline
def hello():
    return "hello world"
print(hello()) ## returns "<b><i><u>hello world</u></i></b>"
```

40. Write the output of the following code:

```
def test(a):
    def add(b):
        nonlocal a
        a += 1
        return a+b
    return add
func = test(5)
print(func(5))
```

Ans. 11

Note: Python 3 introduced the non-local keyword that allows you to assign value to variables in an outer, but non-global, scope. The usage of non-local is very similar to that of global, except that the former is used for variables in outer function scopes whereas the latter is used for variable in the global scope.

41. What is recursion?

Ans. In a program, if a function calls itself (whether directly or indirectly), it is known as recursion. And the function calling itself is called recursive function. Following are two examples of recursion:

(i)
```
def A():
    A()
```

(ii)
```
def B():
    C()
def C():
    B()
```

42. What are the base case and recursive case? What is their role in a recursive program?

Ans. In a recursive solution, the base cases are predetermined solutions for the simplest version of the problem: if the given problem is a base case, no further computation is necessary to get the result.

The recursive case is the one that calls the function again with a new set of values. The recursive step is a set of rules that eventually reduces all versions of the problem to one of the base cases when applied repeatedly.

43. Which of the following is correct Skeleton for a recursive function?

(a)
```
def solution (N) :
    if base_case_condition:
        return something easily computed or directly available.
    else:
        Divide problem into pieces
        return something calculated using solution (some number)
```

(b) def solution (N):
    if base_case_condition:
        return something easily computed or directly available
    else:
        Divide problem into pieces
        return something calculated using solution(N)

(c) def solution (N):
    Divide problem into pieces
    return something calculated using solution(N)

(d) def solution (N):
    if base_case_condition:
        return something easily computed or directly available
    else:
        Divide problem into pieces
        return something calculated using solution (some number other than N).

Ans. (d)

44. Why is base case so important in a recursive function?

Ans. The base case, in a recursive function, represents a pre-known solution. This case is very important because upon reaching the base case, the termination of recursive function occurs as base case does not invoke the function again, rather it returns a pre-known result. In the absence of base case, the recursive function executes endlessly. Therefore, the execution of base case is necessary for the termination of the recursive function.

45. When does infinite recursion occur?

Ans. Infinite recursion is when a recursive function executes itself again and again endlessly. This happens when either the base case is missing or never executed.

46. Differentiate between iteration and recursion.

Ans. In iteration, the code is executed repeatedly using the same memory space for variables. That is, the memory space allocated once is used for each pass of the loop.

On the other hand, in recursion, since it involves function call at each step, fresh memory is allocated for every variable in each recursive call. For this reason, *i.e.*, because of function call overheads and extra memory space occupied by stack manipulation, the recursive function runs slower than its iterative counterpart.

47. State one advantage and one disadvantage of using recursion over iteration.

Ans. *Advantage:* Recursion makes the code short and simple while iteration makes the code comparatively longer.

Disadvantage: Recursion is slower than iteration due to overhead of multiple function calls and maintaining a stack for it.

48. Consider the following function that takes two positive integer parameters a and b. Answer the following questions based on the code below.

```
def func1(a,b):
    if a > 1:
        if a % b == 0:
            print(b, end =' ')
            func1(int(a/b), b)
        else:
            func1(a, b + 1)
```

(a) What will be printed by the function call func1 (24, 2)?
(b) What will be printed by the function call func1 (84, 2)?
(c) State in one line what func1() is trying to calculate.

Ans. (a) 2 2 2 3
(b) 2 2 3 7
(c) Finding factors of a which are greater than and equal to b

49. Write a Recursive function **recurfactorial(n)** in Python to calculate and return the factorial of number n passed to the parameter.

Ans.
```
def recurfactorial(n):
    if n == 1:
        return n
    else:
        return n*recurfactorial(n-1)
num = int(input("Enter a number:"))
if num < 0:
    print("Sorry, factorial does not exist for negative numbers")
elif num == 0:
    print("The factorial of 0 is 1")
else:
    print("The factorial of",num,"is",recurfactorial(num))
```

50. Figure out the error in the following recursive code of factorial:

```
def factorial(n):
    if n == 0:
        return 1
    else:
        n * factorial(n-1)
# __ main __
print(factorial(4))
```

Ans. The error in above code is that the recursive case is calling function factorial() with a changed value but not returning the result of the call.

The corrected version will be:

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n*factorial(n-1)
# __ main __
print(factorial(4))
```

51. Study the following program and select the possible output(s) from options (i) to (iv) following it. Also, write the maximum and the minimum values that can be assigned to variable Y.

```
import random
X= random.random()
Y= random.randint(0,4)
print(int(X),":",Y+int(X))
```

(i) 0 : 0 (ii) 1 : 6

(iii) 2 : 4 (iv) 0 : 3

Ans. (i) and (iv) are the possible outputs. Minimum value that can be assigned to Y is 0. Maximum value that can be assigned to Y is 3.

52. Write your own version of code so that the problem in the question 50 gets solved.

Ans.
```
def out_upto(n):
    if n == 0:
        return 1
    else:
        return n* out_upto(n-1)
        print(n)
#__main__
out_upto(4)
```

53. Write recursive code to compute and print sum of squares of n numbers. Value of n is passed as parameter.

Ans.
```
def sqsum(n):
    if n == 1:
        return 1
    else:
        return n * n + sqsum(n-1)
#__main__
n = int(input("Enter value of n:"))
print(sqsum(n))
```

54. Write recursive code to compute the greatest common divisor of two numbers.

Ans.
```
def gcd(a,b):
    if (b == 0):
        return a
    else:
        return gcd(b, a % b)
N1 = int(input("Enter first number:"))
N2 = int(input("Enter second number:"))
D = gcd(N1, N2)
print("GCD of", N1, "and", N2, "is:", D)
```

55. What will be the output of the following code?

```
def Calculate(A, B, C):
    return A*2, B*2, C*2
val = Calculate(10, 12, 14)
print(type(val))
print(val)
```

Ans.
```
<class 'tuple'>
(20, 24, 28)
```

56. What is the output of the below program?

```
def say(message, times = 1):
    print(message * times)
say('Hello')
say('World', 5)
```

Ans.
```
Hello
WorldWorldWorldWorldWorld
```

57. Write a user-defined function to generate odd numbers between a and b (including b).

Note: a and b are received as an argument by the function.

Ans.
```
def generateodd(a, b):
    for i in range(a, b+1):
        if(i%2 != 0):
            print(i)
```

58. Write a user-defined function findname(name) where name is an argument in Python to delete phone number from a dictionary phonebook on the basis of the name, where name is the key.

Ans.
```
def findname(name):
    if phonebook.has_key():
        del phonebook[name]
    else:
        print("Name not found")
    print("Phonebook Information")
    print("Name", "\t", "Phone Number")
    for i in phonebook.keys():
        print(i, '\t', phonebook[i])
```

UNSOLVED QUESTIONS

1. A program having multiple functions is considered better designed than a program without any functions. Why?
2. Write a function called calculate_area() that takes base and height as input arguments and returns area of a triangle as an output. The formula used is:

   Triangle Area = ½*base*height
3. Modify the above function to take a third parameter called shape type. Shape type should be either triangle or rectangle. Based on the shape, it should calculate the area.

   Formula used: Rectangle Area = length*width
4. Write a function called print_pattern() that takes integer number as argument and prints the following pattern if the input number is 3.

   *
   **
   ***

   If input is 4, then it should print:

   *
   **
   ***
   ****
5. What is the utility of:

   (i) Default arguments (ii) Keyword arguments
6. Describe the different types of functions in Python using appropriate examples.
7. What is scope? What is the scope-resolving rule of Python?
8. What is the difference between local and global variables?
9. Write the term suitable for the following descriptions:

   (a) A name inside the parentheses of a function header that can receive a value.
   (b) An argument passed to a specific parameter using the parameter name.
   (c) A value passed to a function parameter.
   (d) A value assigned to a parameter name in the function call.
   (e) A name defined outside all function definitions.
   (f) A variable created inside a function body.
10. Consider the following code and write the flow of execution for this. Line numbers have been given for our reference.

```
1. def power(b, p):
2. y = b ** p
3. return y
4.
5. def calcSquare(x):
6. a = power(x, 2)
7. return a
8.
9. n = 5
10. result = calcSquare(n)
11. print(result)
```

11. What will the following function return?

```
def addEm(x, y, z):
    print(x + y + z)
x=y=z=10
```

12. What will be the output displayed when addEM() is called/executed?

```
def addEM(x, y, z):
    return x + y + z
x=y=z=20
```

13. What will be the output of the following programs?

(i)
```
num = 1
def myfunc():
    return num
print(num)
print(myfunc())
print(num)
```

(ii)
```
num = 1
def myfunc():
    num = 10
    return num
print(num)
print(myfunc())
print(num)
```

(iii)
```
num = 1
def myfunc():
    global num
    num = 10
print(num)
print(myfunc())
print(num)
```

(iv)
```
def display():
    print("Hello",)
display()
print("bye!")
```

14. Predict the output of the following code:

```
a = 10
y = 5
def myfunc(a):
    y = a
    a = 2
    print("y=", y, "a=", a)
    print("a+y", a + y)
    return a + y
print("y=", y, "a=", a)
print(myfunc(a))
print("y=", y, "a=", a)
```

15. What is wrong with the following function definition?

```
def addEm(x, y, z):
    return x + y + z
    print("the answer is :", x + y + z)
```

16. Find the errors in the code given below:

(a)
```
def minus(total, decrement)
    output = total - decrement
    print(output)
    return (output)
```

(b)
```
define check()
    N = input ('Enter N:'_
    I = 3
    Answer = 1 + i ** 4/N
    Return answer
```

(c)
```
Def alpha (n, string = 'xyz', K=10):
    return beta(string)
    return n
def beta (string)
    Return string == str(n)
print(alpha("Valentine's Day"))
print(beta (string = 'true'))
print(alpha (n = 5, "Good-bye"):))
```

17. Define flow of execution. What does it do with functions?

18. Write a function that takes amount-in-dollars and dollar-to-rupee conversion price; it then returns the amount converted to rupees. Create the function in both void and non-void forms.

19. Write a function to calculate volume of a box with appropriate default values for its parameters. Your function should have the following input parameters:
    (a) Length of box
    (b) Width of box
    (c) Height of box

    Test it by writing a complete program to invoke it.

20. Write a program to display first four multiples of a number using recursion.

21. What do you understand by recursion? State the advantages and disadvantages of using recursion.

22. Write a recursive function to add the first 'n' terms of the series:
    1+1/2–1/3+1/4–1/5.....

23. Write a program to find the greatest common divisor between two numbers.

24. Write a Python function to multiply all the numbers in a list.
    Sample List: (8, 2, 3, –1, 7)
    Expected Output : –336

25. Write a Python function to calculate the factorial of a number (a non-negative integer). The function accepts the number whose factorial is to be calculated as the argument.

26. Write a Python function that takes a number as a parameter and checks whether the number is prime or not.

27. Write a Python function that checks whether a passed string is a palindrome or not.
    Note: A palindrome is a word, phrase, or sequence that reads the same backward as forward, *e.g.*, madam or nurses run.

28. Write a Python program that accepts a hyphen-separated sequence of words as input and prints the words in a hyphen-separated sequence after sorting them alphabetically.
    Sample Items: green-red-yellow-black-white
    Expected Result: black-green-red-white-yellow

29. What is a recursive function? Write one advantage of recursive function.

30. What are the two cases required in a recursive function?

31. What is base case?

32. What is recursive case?

33. Is it necessary to have a base case in a recursive function? Why/Why not?

34. Identify the base case(s) in the following recursive function:

```
def function1(n):
    if n == 0:
        return 5
    elif n == 1:
        return 8
    elif n > 8 :
        return function1(n-1) + function1(n-2)
    else:
        return -1
```

35. Why are recursive functions considered slower than their iterative counterparts?

36. Compare and contrast the use of iteration and recursion in terms of memory space and speed.

37. Predict the output of the following codes.

(a)
```
def code(n):
    if n == 0:
        print('Finally')
    else:
        print(n)
code(3)
code(15)
```

(b)
```
def code(n):
    if n==0:
        print('Finally')
    else:
        print(n)
code(2-2)
code(15)
```

(c)
```
def code(n):
    if n == 0:
        print('Finally')
    else:
        print(n)
code(10)
```

(d)
```
def code(n):
    if n==0:
        print('Finally')
    else:
        print(n)
        print(n-3)
code(10)
```

38. Predict the output of the following code:

```
def express(x, n):
    if n == 0:
        return 1
    elif n % 2 == 0:
        return express(x*x, n/2))
    else:
        return x * express(x, n-1)
express(2,5)
```

39. Consider the following Python function that uses recursion.

```
def check(n):
    if n <=1:
        return True
    elif n % 2 == 0:
        return check(n/2)
    else:
        return check(n/1)
```

What is the value returned by `check(8)`?

40. Can you find an error or problem with the above code? Think about what happens if we evaluate check(3). What output would be produced by Python? Explain the reason(s) behind the output.

41. Consider the following Python function Fn(), that follows:

```
def Fn(n):
    print(n, end=" ")
    if n < 3:
        return n
    else:
        return Fn(n//2) - Fn(n//3)
```

What will be the result produced by the following function calls?

(a) `Fn(12)`

(b) `Fn(10)`

(c) `Fn(7)`

42. Figure out the problem with the following code that may occur when it is run.

```
def recur(p):
    if p == 0:
        print("##")
    else:
        recur(p)
        p = p - 1
```

43. Write a method in Python to find and display the prime numbers from 2 to N.
The value of N should be passed as an argument to the method.

44. Write definition of a method EvenSum(NUMBERS) to add those values in the tuple of NUMBERS which are even.

45. Write definition of a method COUNTNOW(PLACES) to find and display those place names in which there are more than 5 characters after storing the name of places in a dictionary. For example,
If the dictionary PLACES contains:
{'1': "DELHI", '2': "LONDON", '3': "PARIS", '4': "NEW YORK", '5': "DUBAI"}
The following output should be displayed:
LONDON
NEW YORK

46. Write a program using user defined-function to calculate and display average of all the elements in a user-defined tuple containing numbers.

47. Name the built-in mathematical function/method:

(a) Used to return an absolute value of a number.

(b) Used to return the value of x^y, where x and y are numeric expressions.

(c) Used to return a positive value of the expression in float.

48. Name the built-in String function:
   (a) Used to remove the space(s) from the left of the string.
   (b) Used to check if the string is uppercase or not.
   (c) Used to check if the string contains only whitespace characters or not.

49. Write a function LeftShift(lst,x) in Python which accepts numbers in a list (lst) and all the elements of the list should be shifted to left according to the value of x.
   Sample Input Data of the list lst= [20,40,60,30,10,50,90,80,45,29] where x=3
   Output lst = [30,10,50,90,80,45,29,20,40,60]

CASE-BASED/SOURCE-BASED INTEGRATED QUESTIONS

1. Traffic accidents occur due to various reasons. While problems with roads or inadequate safety facilities lead to some accidents, majority of the accidents are caused by drivers' carelessness and their failure to abide by traffic rules.

   ITS Roadwork is a company that deals with manufacturing and installation of traffic lights so as to minimize the risk of accidents. Keeping in view the requirements, traffic simulation is to be done. Write a program in Python that simulates a traffic light. The program should perform the following:
   (a) A user-defined function trafficLight() that accepts input from the user, displays an error message if the user enters anything other than RED, YELLOW and GREEN. Function light() is called and the following is displayed depending upon return value from light():
      (i) "STOP, Life is more important than speed" if the value returned by light() is 0.
      (ii) "PLEASE GO SLOW." if the value returned by light() is 1.
      (iii) "You may go now." if the value returned by light() is 2.
   (b) A user-defined function light() that accepts a string as input and returns 0 when the input is RED, 1 when the input is YELLOW and 2 when the input is GREEN. The input should be passed as an argument.
   (c) Display "BETTER LATE THAN NEVER" after the function trafficLight() is executed.

Ans.

File Edit Format Run Options Window Help

```
#Program to simulate a traffic light comprising of
#two user defined functions trafficLight() and light().

def trafficLight():
    signal = input("Enter the colour of the traffic light: ")
    if (signal not in ("RED","YELLOW","GREEN")):
        print("Please enter a valid Traffic Light colour in CAPITALS")
    else:
        value = light(signal) #function call to light()
        if (value == 0):
            print("STOP, Life is more important than speed")
        elif (value == 1):
            print("PLEASE GO SLOW.")
        else:
            print("You may go now.")

def light(colour):
    if (colour == "RED"):
        return(0)
    elif (colour == "YELLOW"):
        return(1)
    else:
        return(2)
#function ends here

trafficLight()
print("BETTER LATE THAN NEVER")
```

```
= RESTART: C:/Users/Ashish Aggarwal/AppData/Local/Programs/Python/Python38/prog
_traffic_light.py
Enter the colour of the traffic light: RED
STOP, Life is more important than speed
BETTER LATE THAN NEVER
>>>
= RESTART: C:/Users/Ashish Aggarwal/AppData/Local/Programs/Python/Python38/prog
_traffic_light.py
Enter the colour of the traffic light: YELLOW
PLEASE GO SLOW.
BETTER LATE THAN NEVER
>>>
= RESTART: C:/Users/Ashish Aggarwal/AppData/Local/Programs/Python/Python38/prog
_traffic_light.py
Enter the colour of the traffic light: GREEN
You may go now.
BETTER LATE THAN NEVER
```

2. Kids Elementary is a playway school that focuses on 'Play and learn' strategy that helps toddlers understand concepts in a fun way. Being a senior programmer, you have taken responsibility to develop a program using user-defined functions to help children differentiate between upper case and lower case letters/English alphabet in a given sentence. Make sure that you perform a careful analysis of the type of alphabets and sentences that can be included as per age and curriculum.

Write a Python program that accepts a string and calculates the number of upper case letters and lower case letters.

Ans.

File Edit Format Run Options Window Help

```
#Write a user-defined function that accepts a string
#and calculates the number of upper case letters and lower case letters

def string_test(s):
    d = {"UPPER_CASE":0, "LOWER_CASE":0}
    for c in s:
        if c.isupper():
            d["UPPER_CASE"]+=1
        elif c.islower():
            d["LOWER_CASE"]+=1
        else:
            pass
    print("Original String : ", s)
    print("No. of Upper case characters : ", d["UPPER_CASE"])
    print("No. of Lower case characters : ", d["LOWER_CASE"])

string_test("Play Learn and Grow")
```

```
>>>
= RESTART: C:/Users/Ashish Aggarwal/AppData/Local/Programs/Python/Python38/prog
_letters_playway.py
Original String :  Play Learn and Grow
No. of Upper case characters :  3
No. of Lower case characters :  13
>>>
```