UNIT - IV

4 Python Functions, Modules and Packages

4.1 Use of Python Built in Functions

What is function ?

- Function is a block of instructions that performs a specific and well defined task. This block can be repeatedly called by other programming instruction.
- Every function is specified by a name.
- **Following are some reasons why do we need to use functions in python.**
  1. All the logically related statements can be grouped together in one entity. This makes the program easy to read, understand and debug.
  2. The repetition of frequently required code can be avoided by using function.
  3. Dividing a long program into functions allows us to debug the parts one at a time and then assemble them into a working whole.
  4. Once the code of the function is written and tested, we can reuse this code. Reusability is an important use of function.

Review Question

1. *What is function ? Explain the need for writing a function in python.*

4.2 User Defined Functions

4.2.1 Function Definition

The function can be defined using the **def**.

The **syntax** of function definition is as follows -

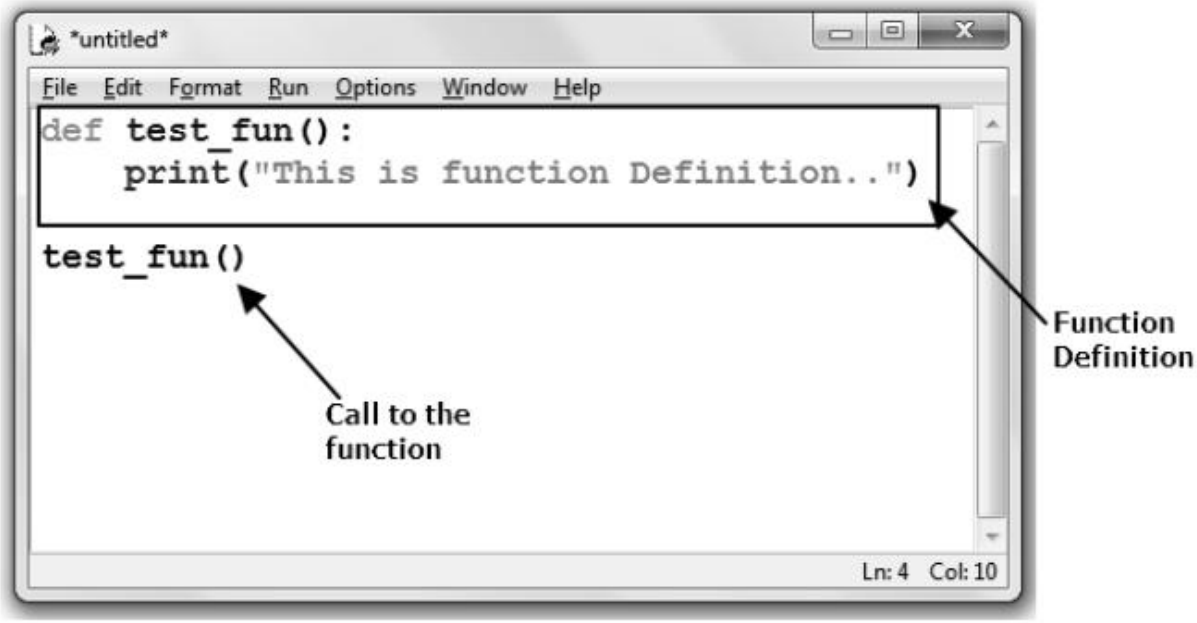
```
def function_name(parameters):
    statements
```

This colon is must

Example of creation of function

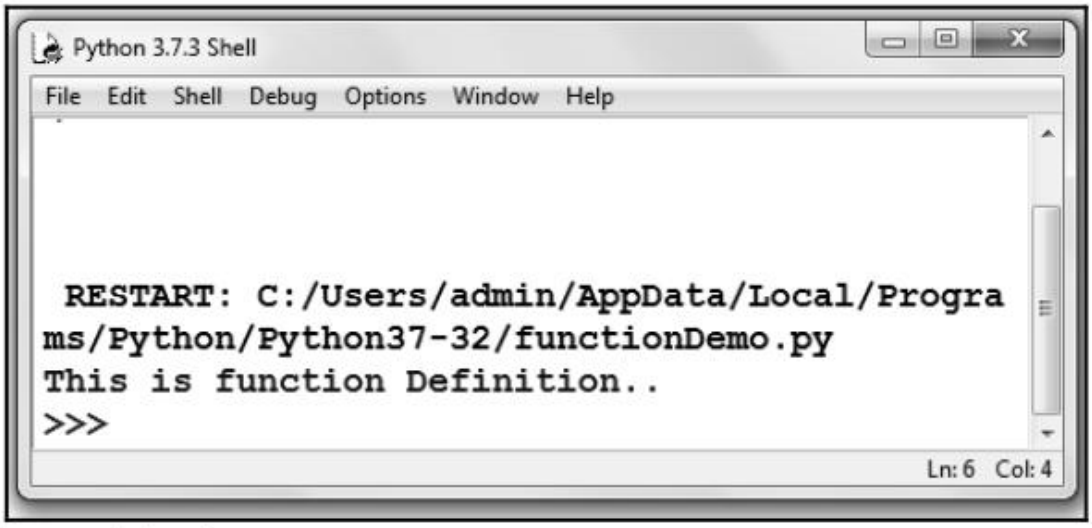
Step 1 : Create a function definition in a file. Open **File->New File** and type following code.

```
def test_fun():
  print("This is function Definition..")

test_fun()
```



Step 2 : Now save the above file using some suitable file name. For that click on **File->Save.** I have given the name **functionDemo.py.**

Step 3 : Press F5 button and run the above program. You will get the following output –



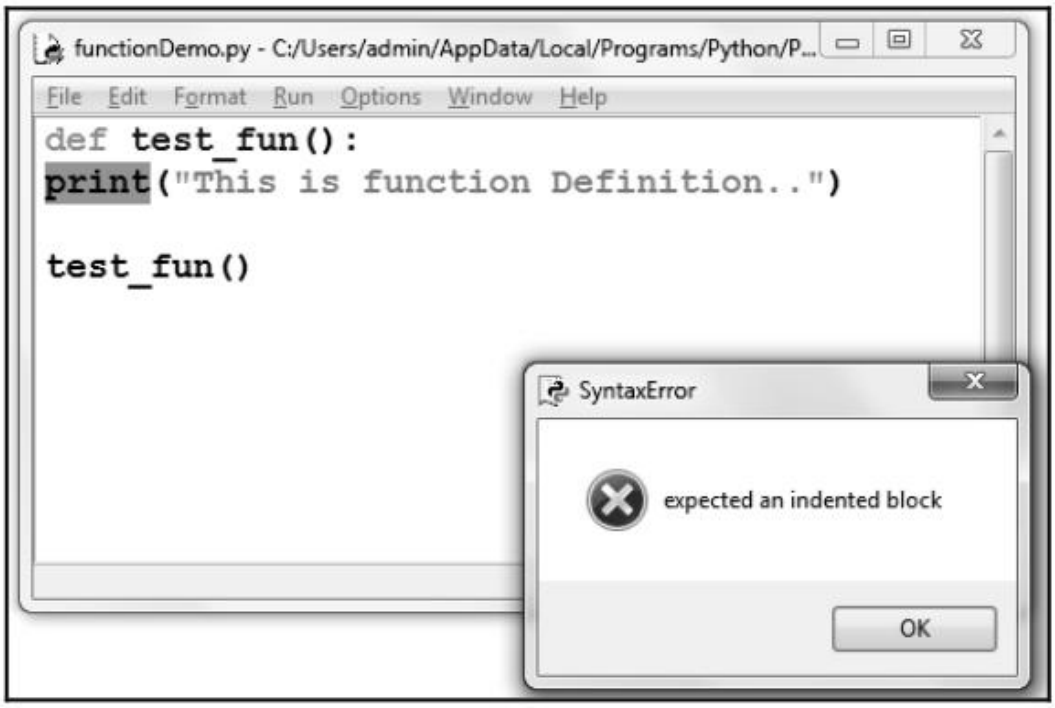
Here is a sample function.

```
def test_fun():
    print("The function is defined here...")
    print("We are now learning functions in Python")
    print("Python is easy to learn")
    print("Good Bye")
```

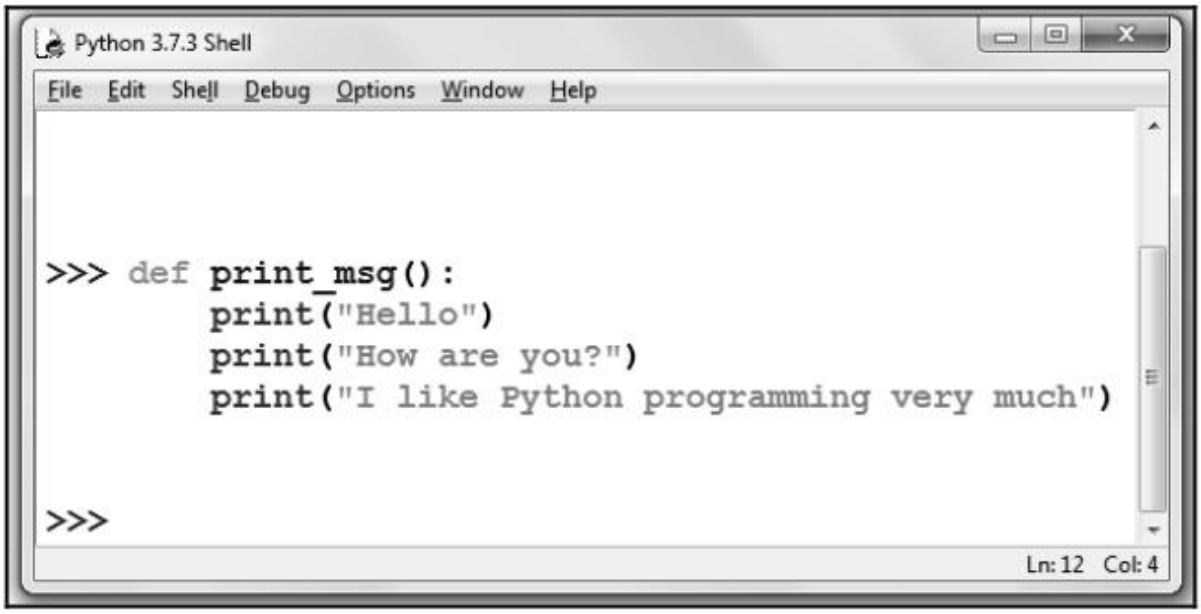
Function header

Function Body

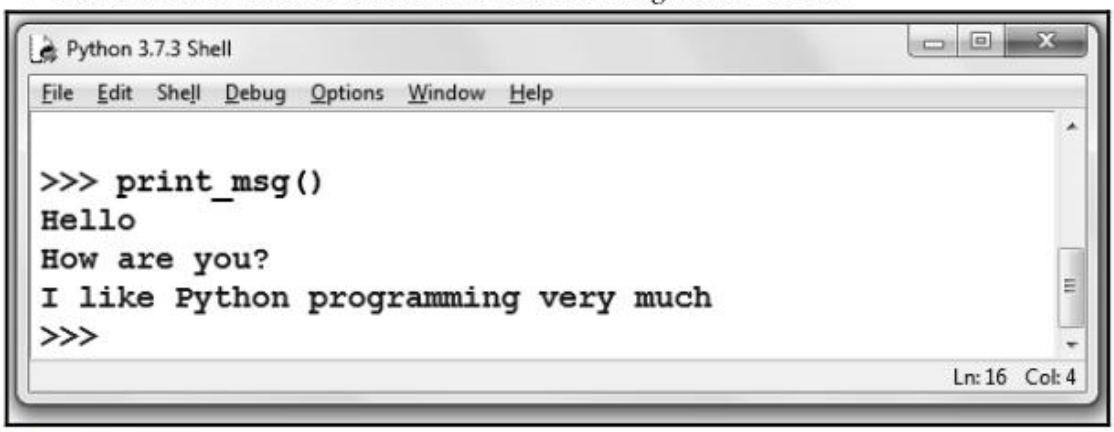
- The first line of function definition is called function **header** and rest is called **body.**
- Note that - The code within every function **starts with a colon (:)** and **should be indented (space)**.
- Python functions don't have any explicit begin or end like curly braces to indicate the start and stop for the function, they have to rely on this indentation.
- **For example :** If we write the above function without indentation, then error will occur. Following screenshot illustrates this -



- If you type a function definition in interactive mode, the interpreter prints dots (...) to let you know that the definition is not complete. For example :



- Then the call to the function in interactive mode is given as follows –



4.2.2 Function Calling

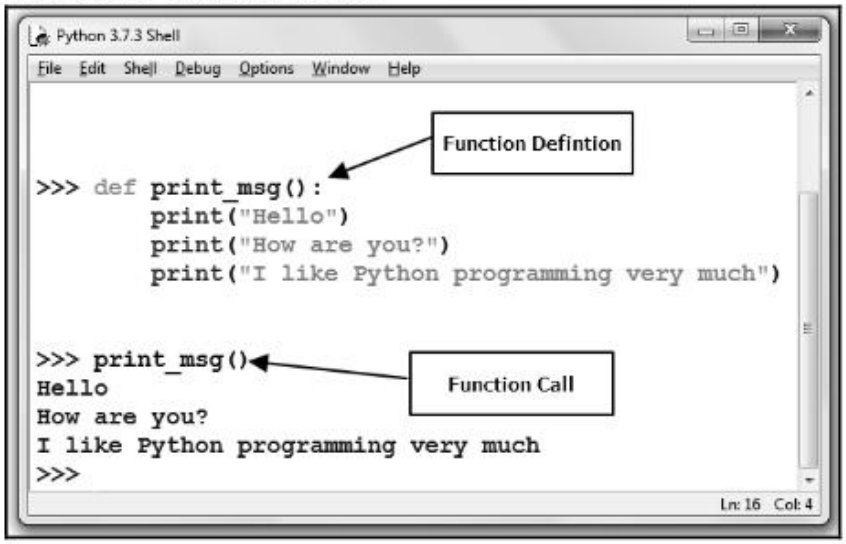
- The functional call is a statement that invokes the function.
- When a function is called the program control jumps to the definition of the function and executes the statements present in the function body.
- Once all the statements in function body get executed, the control goes back to the calling function.
- **Syntax**

```
Function_name()
```

- **Example**

```
>>> def print_msg():
        print("Hello")
        print("How are you?")
        print("I like Python programming very much")
#call to the function
>>> print_msg()
Hello
How are you?
I like Python programming very much
>>>
```

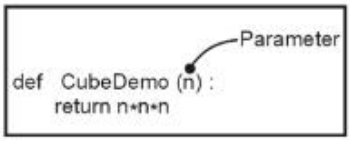
The following is a screenshot of the above code.



4.2.3 Function Argument and Parameter Passing

- The argument is a value that is passed to the function when it is called.
- **Syntax**

  Function_name(variable1, variable2,...)
- On the calling side, it is an **argument** and on the function side it is a **parameter.**
- For example - Consider following function in which there is single parameter passed to the function **CubeDemo.**

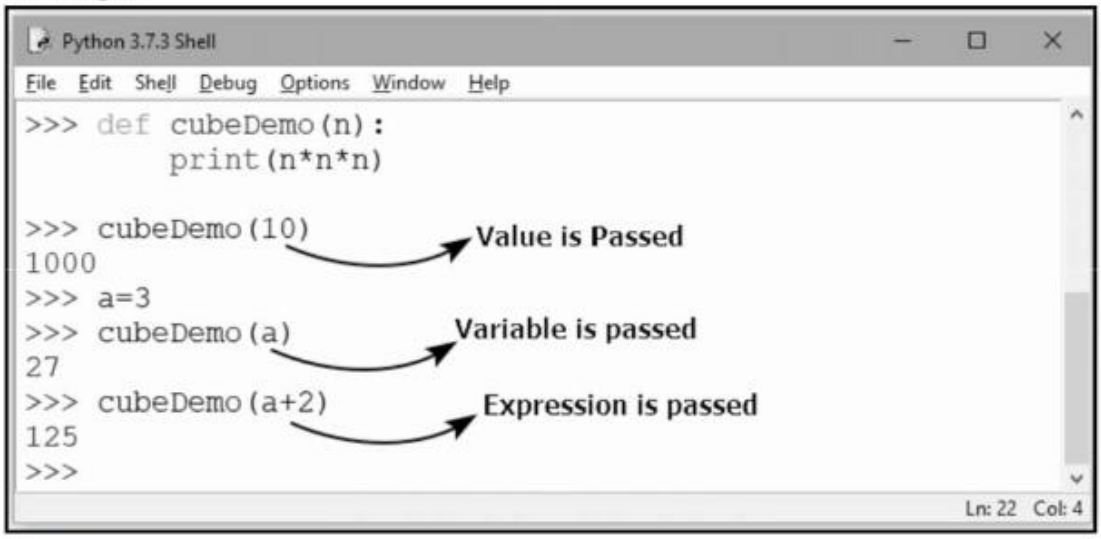


- We can pass

  1) Directly a value as a parameter or

  2) A variable or

  3) An expression.
- **For example :**



4.2.4 Return Statement

- The value can be returned from a function using the keyword **return**. For example
- **Syntax**

  return [expression_list]

Uses of Return Statement

1) The return statement is used to return a value from a function to the caller.
2) The return statement is sometimes used to return a control back to the caller.

Example 4.2.1 : Write a Python program to compute area of circle using function and a return statement.
Solution :

```
def areaCircle(r):
PI = 3.14
result=PI*r*r
print("The result of circle: ")
return result
```

Output

```
>>> areaCircle(10)
The result of circle:
314.0
>>>
```

4.2.5 Scope of Variables

- **Scope** of the variable means - the part of the program in which a variable is accessible.
- **Lifetime** of variable means - duration for which the variable exists.

Local and Global Variable

- **Definition of Global Variable :** A variable which is defined in the main body of a file is called a **global variable.** It will be visible throughout the file, and also inside any file which imports that file.
- Only objects which are intended to be used globally, like functions and classes, should be put in the **global namespace**.
- **Definition of Local Variable :** A variable which is defined inside a function is local to that function. It is accessible from the point at which it is defined until the end of the function, and exists for as long as the function is executing.
- The parameter names in the function definition behave like local variables, but they contain the values that we pass into the function when we call it.
- When we use the assignment operator (=) inside a function, its default behaviour is to create a new local variable.

For example -

```
a=10 # a is Global Variable
def My_Function(b):# b is Local variable
c=30 # c is Local variable
print("b = ",b)
print("c = ",c)

My_Function(20)
print("a = ",a)#a still exists
print("b = ",b)#being Local variable, it does not exist outside function
print("c = ",c)#being Local variable, it does not exist outside function
#Error: for b and c variable
```

Output

```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
b =  20
c =  30
a =  10
Traceback (most recent call last):
  File "C:/Users/Puntambekar/AppData/Local/Programs/Python/Python37-
32/test.py", line 9, in <module>
    print("b = ",b)#being Local variable, it does not exist outside
function
NameError: name 'b' is not defined
>>>
Ln: 12  Col: 4
```

4 70%

Difference between Local and Global Variable

Sr. No.	Local Variable	Global Variable
1.	A variable which is declared inside a function is called as local variable.	A variable that is declared outside the function is called global variable.
2.	Accessibility of variable is only within the function in which it is declared.	Accessibility of variable is throughout the program. All functions in the program can access the global variable.
3.	The local variable is created when the function(in which it is defined) starts executing and it is destroyed when the execution is complete.	The global remains throughout the entire program execution.
4.	It is preferred as local variables are more reliable. Because the values cannot be changed outside the function.	It is generally not preferred as any function can change the values of global variable.

4.2.6 Python Programs based on Functions

Example 4.2.2 : Write a Python program to find the maximum of three numbers using function.

Solution :

```
def largest_among_three(a,b,c):
   if(a>b and a>c):
      print("a is largest");
   elif(b>a and b>c):
      print("b is largest")
   else:
      print("c is largest")

print("\t\tProgram for finding largest of three numbers")
print("Enter value of a: ")
a=int(input())
print("Enter value of b: ")
b=int(input())
print("Enter value of c: ")
c=int(input())
largest_among_three(a,b,c)
```

Output(Run1)

```
      Program for finding largest of three numbers
Enter value of a:
10
Enter value of b:
20
Enter value of c:
30
c is largest
>>>
```

Output(Run2)

```
    Program for finding largest of three numbers
Enter value of a:
20
Enter value of b:
10
Enter value of c:
5
a is largest
>>>
```

Output(Run3)

```
    Program for finding largest of three numbers
Enter value of a:
10
Enter value of b:
30
Enter value of c:
20
b is largest
>>>
```

Example 4.2.3 : Write a Python program to swap two numbers using function.

Solution :

```
def swap(a,b):
  temp=a
  a=b
  b=temp
  print("After swapping")
  print("a: ",a)
  print("b: ",b)


print("\t\t Program for swapping two numbers")
print("Enter value of a: ")
a=int(input())
print("Enter value of b: ")
b=int(input())
print("Before swapping")
print("a: ",a)
print("b: ",b)

swap(a,b)
```

Output

```
Program for swapping two numbers
Enter value of a:
10
Enter value of b:
20
Before swapping
a:  10
```

```
b:  20
After swapping
a:  20
b:  10
>>>
```

Example 4.2.4 : Write a Python program to calculate simple interest using function.

Solution :

```
def interest_calculation(p,n,r):
   interest=0.0
   interest=float(p*n*r)/float(100)
   return interest

print("\t\t Program for calculating simple interest")
print("Enter value of p: ")
p=float(input())
print("Enter value of n: ")
n=int(input())
print("Enter value of r: ")
r=float(input())

print("Simple Interest = ",interest_calculation(p,n,r))
```

Output

```
      Program for calculating simple interest
Enter value of p:
10000
Enter value of n:
5
Enter value of r:
6.5
Simple Interest =  3250.0
>>>
```

Example 4.2.5 : Write a Python program to reverse the given number using function.

Solution :

```
def Reverse(n):
   rev=0
   while(n>0):
      remainder=n%10
      rev=(rev*10)+remainder
      n=n//10
   return rev

print("\t\t Program for Reversing a number")
print("Enter value of n: ")
n=int(input())
print("The reverse of ",n," is ",Reverse(n))
```

Output

```
   Program for Reversing a number
Enter value of n:
1234
The reverse of  1234  is  4321
>>>
```

34 70%

Example 4.2.6 : Write a python program to find out the factorial of a given number

Solution :

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

print("\t\t Program for Factorial a number")
print("Enter value of n: ")
n=int(input())
if(n<0):
   print("Can not find factorial of a negative number")
elif (n==0):
   print("Factorial of 0 is 1")
else:
   print("The reverse of ",n," is ",factorial(n))
```

Output

```
      Program for Factorial a number
Enter value of n:
5
The reverse of  5  is  120
>>>
```

Example 4.2.7 : Write a Python program using function to check if the entered year is leap year or not

Solution :

```
def leap(year):
   if((year % 4 == 0) and (year % 100 != 0)):
      print(year, "is a Leap Year")
   elif (year % 400 == 0):
      print(year, "is a Leap Year")
   elif (year % 100 == 0):
      print(year, "is not a Leap Year")
   else:
      print(year, "is not a Leap Year")

print("\t\t Program for Check whether the year is leap year or not")
print("Enter the value of year: ")
year=int(input())
leap(year)
```

Output(Run 1)

```
      Program for Check whether the year is leap year or not
Enter the value of year:
2001
2001 is not a Leap Year
>>>
```

84 70%

Output(Run 2)

```
 Program for Check whether the year is leap year or not
Enter the value of year:
2000
2000 is a Leap Year
>>>
```

Example 4.2.8 : Write a Python program check the number is prime or not.

Solution :

```
def isprime(num):
   if num > 1:
      for i in range(2, num):
         if(num%i)==0:
            print(num," is not a prime number")
            print(i," times",num//i," is ",num)
            break
      else:
         print(num," is a prime number")
   else:
      print(num," is not a prime number")


print("\t\t Program for Check whether the number is prime or not")
print("Enter the number: ")
num=int(input())
isprime(num)
```

Output(Run 1)

```
Program for Check whether the number is prime or not
Enter the number:
13
13  is a prime number
>>>
```

Output(Run2)

```
Program for Check whether the number is prime or not
Enter the number:
12
12  is not a prime number
2  times 6  is  12
>>>
```

Example 4.2.9 : Write a Python program to find the list of prime numbers from given range.

Solution :

```
def prime(lower,upper):
   for num in range(lower,upper + 1):
      if num > 1:
         for i in range(2,num):
            if (num % i) == 0:
```

```
            break
        else:
            print(num)
print("\t\t Program to Find is prime numbers from a range")
print("Enter the range[lower-upper]: ")
lower=int(input())
upper=int(input())
print(" The prime numbers are...")
prime(lower,upper)
```

Output

```
    Program to Find is prime numbers from a range
Enter the range[lower-upper]:
1
100
 The prime numbers are...
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
71
73
79
83
89
97
>>>
```

Example 4.2.10 : Write a Python program using function to check if the number is perfect or not.

Solution :

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.

For example – Consider number 28 =1 + 2 + 4 + 7 + 14.

Equivalently, a perfect number is a number that is half the sum of all of its positive divisors (including itself).

```
def perfect_num_chk(n):
   sum = 0
   for i in range(1,n):
      if n%i == 0:
         sum=sum+i
   if sum==n:
      print("It is a perfect number")
   else:
      print("It is a not a perfect number")
print("\t\t Program to check if the number is perfect or not")
print("Enter some number: ")
n=int(input())
perfect_num_chk(n)
```

Output

```
      Program to check if the number is perfect or not
Enter some number:
28
It is a perfect number
>>>
```

Example 4.2.11 : Write a Python program using function and print a list where values are squares of numbers between 1 and 30(both included)

Solution :

```
def print_squares():
   mylist=list()
   for i in range(1,30):
      mylist.append(i**2)
   print(mylist)

print("\t\t Program to print a list of squares")
print(" The squares between 1 to 30 are ...")
print_squares()
```

Output

```
      Program to print a list of squares
 The squares between 1 to 30 are ...
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400, 441, 484, 529, 576, 625, 676, 729,
784, 841]
>>>
```

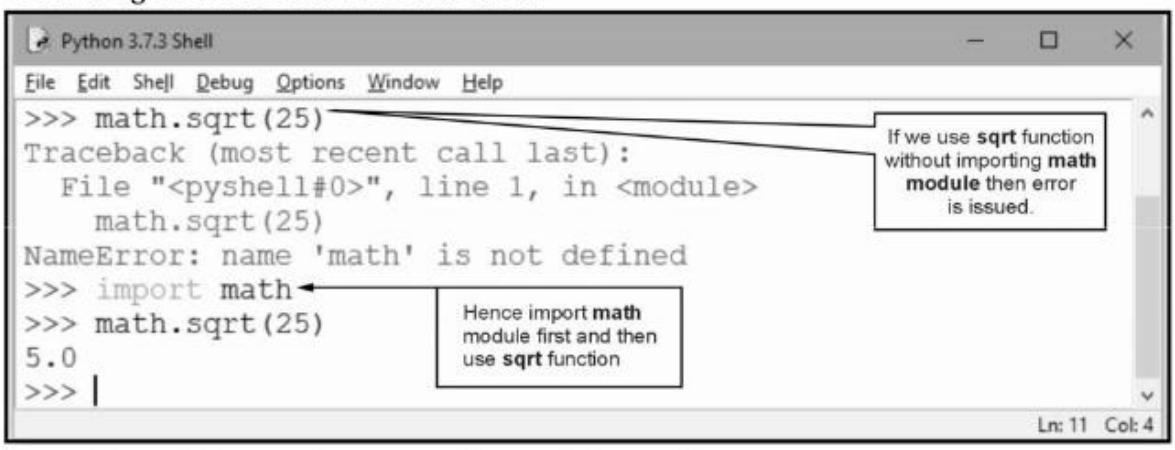
Review Questions

1. *What is function ?*
2. *How will you define and call a function in Python ? Give illustrative example.*
3. *What is scope of the variable ?*
4. *Explain return statement in function handling in Python.*

4.3 Modules

4.3.1 Introduction to Modules

- There are some standard functionalities in python that are written in particular module. For using those functionalities, it is essential to import the corresponding module first.
- For example : There are various functionalities available under the module math. For instance, if you want to find out the square root of some number then you need to import module math first and then use the sqrt function.
- **Following screenshot illustrates this idea :**



- Basically modules in python are .py files in which set of functions are written.
- Modules are imported using **import** command.

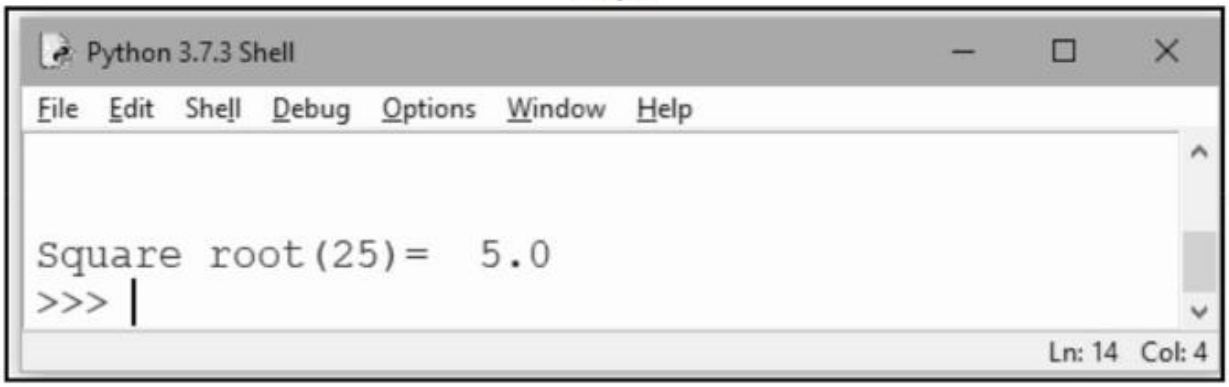
4.3.2 Importing Objects from Modules

- There are many variables and functions present in the module. We can use them by using the import statement.
- When we simply use **import** statement, then we can use any variable or function present within that module.
- When we use **from...import** statement, then we can use only selected variables or functions present within that module.

For example

```
from math import sqrt
print("Square root(25)= ",sqrt(25))
```

Output



- If we want to use some different name for the standard function present in the module, then we can use **as** keyword. Following code illustrates this –

moduleProg1.py

```
from math import sqrt as my_sq_root
print("Square root(25)= ",my_sq_root(25))
```

Output

```
Square root(25)=  5.0
>>>
```

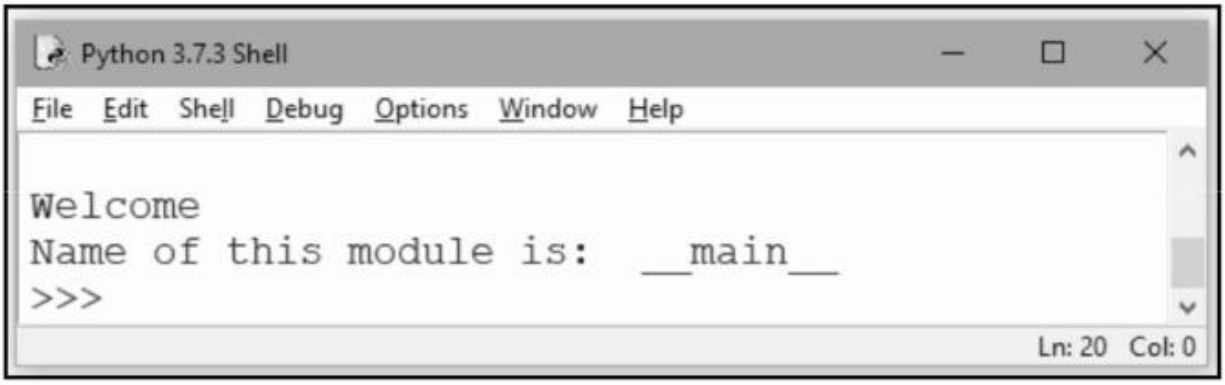
4.3.3 Name of the Module

- Every module has a name. One can use the name of the module using __name__ attribute of the module. For example

moduleProg3.py

```
print("Welcome")
print("Name of this module is: ",__name__)
```

Output



The __name__ is a built-in variable that is set when the program starts. If the program is running as a script, __name__ has the value '__main__'

For every standalone program written by the user the module is always __main__. Hence is the output.

4.3.4 Writing Modules

- We can create our own module. Every Python program is a module. That means every file that we save using .py extension is a module.
- Following are the steps that illustrates how to create a module –

Step 1 : Create a file having extension .py. Here we have created a file PrintMsg.py. In this file, the function fun is defined.

PrintMsg.py

```
def fun(usr):
  print("Welcome ",usr)
```

Step 2 : Now open the python shell and import the above module and then call the functionality present in that module.

Output

```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>> import PrintMsg
>>> PrintMsg.fun("Parth")
Welcome  Parth
>>>
Ln: 26 Col: 4
```

Step 3 : We can also call the above created **PrintMsg** module in some another file. For that purpose, open some another file and write the code in it as follows –

Test.py

```
import PrintMsg  #importing the user defined module
PrintMsg.fun("Rupali")  #calling the function present in that module
```

Step 4 : Now run the **test.py** program and you will get the output as follows -

```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
 RESTART: C:/Users/Puntambekar/AppData/Local/P
rograms/Python/Python37-32/test.py
Welcome  Rupali
>>>
Ln: 29 Col: 4
```

Example 4.3.1 : *Write a module for displaying the Fibonacci series.*

Solution :

Step 1 :

FibSeries.py

```
def fib(n):    # write Fibonacci series up to n
    a, b = 0, 1
    while b < n:
        print(b, end=' ')
        a, b = b, a+b
    print()
```

Step 2 : Open the python shell and import the above module and access the functionality within it. The output will be as follows :

```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>> import FibSeries
>>> FibSeries.fib(10)
1 1 2 3 5 8
>>>
Ln: 34 Col: 4
```

84 − + 70%

Example 4.3.2 : *Write a Python program to define a module for swapping the two values. Then write a driver program that will call the function defined in your swap module.*

Solution :

Step 1 : Create a module in a file and write a function for swapping the two values. The code is as follows –

swapProg.py

```
def Swap(a,b):
  temp=a
  a=b
  b=temp
  print("After Swapping")
  print(a)
  print(b)
```

Save and Run the above code.

Step 2 : Now create a driver program that will invoke the swap function created in above module.

Test.py

```
import swapProg
print("Swap(10,20)")
swapProg.Swap(10,20)
print("Swap('a','b')")
swapProg.Swap('a','b')
```

Step 3 : Save and run the above code. The output will be as follows –

4.3.5 Python Built in Modules

1. Numeric and Mathematical Modules

- The **numbers** module defines a hierarchy of numeric abstract base classes which progressively define more operations. None of the types defined in this module can be instantiated.
- The **fractions** module is used to return the instance with the value=numerator/denominator. For example

  21/49 = 3/7

- Following Python code makes use of fractions module.

```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help

>>> from fractions import Fraction
>>> print(Fraction(21,49))
3/7
>>> print(Fraction(10,18))
5/9
>>> print(Fraction())
0
>>>
Ln: 18 Col: 4
```

- The **math** module used in Python is for various mathematical functionalities such as finding log, computing sine or cosine values, calculating square root and so on.

Some of the functions supported by math module are enlisted as follows –

Function	Purpose
ceil(x)	Returns the smallest integer greater than or equal to x.
factorial(x)	Returns the factorial of x
floor(x)	Returns the largest integer less than or equal to x
isinf(x)	Returns True if x is a positive or negative infinity
exp(x)	Returns e**x
log(x[, base])	Returns the logarithm of x to the base (defaults to e)
pow(x, y)	Returns x raised to the power y
sqrt(x)	Returns the square root of x
cos(x)	Returns the cosine of x
sin(x)	Returns the sine of x
tan(x)	Returns the tangent of x
pi	Mathematical constant, the ratio of circumference of a circle to it's diameter (3.14159...)
e	mathematical constant e (2.71828...)

For example

i) Finding square root of a number.

```
>>> import math
>>> print(math.sqrt(25))
5.0
>>>
```

(ii) Finding gcd(greatest common divisor) of two numbers.

```
>>> import math
>>> print(math.gcd(12,15))
3
>>>
```

(iii) Finding Factorial of a given number.

```
>>> import math
>>> print(math.factorial(5))
120
>>>
```

(iv) Finding power i.e. x^y.

```
>>> import math
>>> print(math.pow(2,3))
8.0
>>>
```

(v) Finding remainder.

```
>>> import math
>>> print(math.remainder(10,3))
1.0
>>>
```

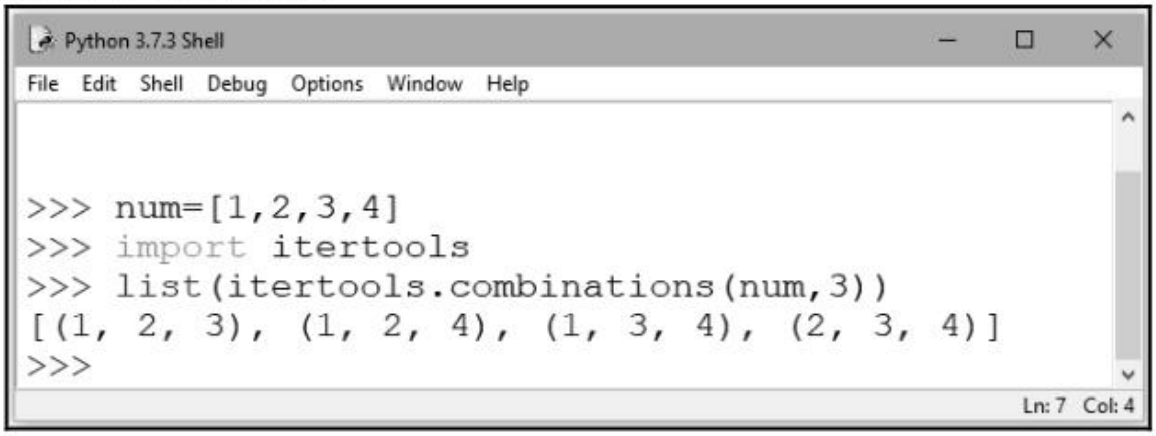
(vi) Displaying the pi constant.

```
>>> import math
>>> print(math.pi)
3.141592653589793
>>>
```

2. Functional Programming Module

The functional programming module provides functions and classes that support functional programming style.

For creating iteration or repetition the **itertools** module is used. There are various functionalities that can be used from itertools module.



Similarly we can have cartesian product, equivalent to a nested for-loop.

```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help

>>> import itertools
>>> list(itertools.product(num,repeat=2))
[(1, 1), (1, 2), (1, 3), (1, 4), (2, 1), (2, 2),
(2, 3), (2, 4), (3, 1), (3, 2), (3, 3), (3, 4),
(4, 1), (4, 2), (4, 3), (4, 4)]
>>> |
Ln: 10 Col: 4
```

We can repeat the numbers using the itertools modules. Following screenshot illustrates this -

```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help

>>> import itertools
>>> list(itertools.repeat(10,3))
[10, 10, 10]
>>> |
Ln: 19 Col: 4
```

Use of map function

The map( ) function applies the values to every member of the list and returns the result.

Syntax : The map function takes two arguments -

```
result=map(function, sequence)
```

For example

```
>>> def increment_by_three(x):
return x+3
>>> new_list=list(map(increment_by_three,range(10)))
>>> new_list
[3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
>>>
```

Another Example :

There is a built in function for calculating length of the string and i.e. len( ). We can pass this function to map and find the length of every element present in the list as a list.

```
>>> names=['Sandip','Lucky','Mahesh','Sachin']
>>> lengths=list(map(len,names))
>>> lengths
[6, 5, 6, 6]
>>>
```

4.3.6 Namespace and Scope

- Namespace is basically a collection of different names. Different namespaces can co-exist at a given time but are completely isolated.
- If two names(variables) are same and are present in the same scope then it will cause name clash. To avoid such situation we use the keyword namespace.
- In Python each module has its own namespace. This namespace includes all the names of its function and variables.
- If we use two functions having same name belonging to two different modules at a time in some other module then that might create name clash. For instance –

Step 1 : Create first module for addition functionality.

FirstModule.py

```
def display(a,b):
    return a+b
```

Step 2 : Create second module for multiplication functionality.

SecondModule.py

```
def display(a,b):
    return a*b
```

Step 3 : If we call above modules in our driver program for performing both addition and multiplication, then we get error because of name clash for the function.

Test.py

```
import FirstModule
import SecondModule
print(display(10,20))
print(display(5,4))
```

Step 4 : To resolve this ambiguity we should call these functionalities using their module names. It is illustrated as follows –

```
import FirstModule
import SecondModule
print(FirstModule.display(10,20))
print(SecondModule.display(5,4))
```

Output

```
30
20
```

4.3.6.1 Global, Local and Built-in Namespace

- There are three commonly used categories of namespaces – **Global namespace**, **local namespace**, and **built-in namespace**.
- The global namespace contains the module which is currently executing. The local namespace is a namespace for defining the names in a local function. The built-in namespace is a namespace in which the built in functionality can be invoked.
- Following Python code represents all these namespaces.

```
import math
def even_number(number):  #global namespace
   num=number #local namespace
   if(num%2==0):
      return num
   else:
      return 0

print("Enter some number")
num=int(input())
if(even_number(num)!=0):
  print("The square root of ",num,"is ",math.sqrt(num)) #built-in namespace
```

Review Questions

1. *What is module in Python ?*
2. *How will you import objects from modules?*
3. *Explain any two built in modules in Python.*
4. *Write a short note on – Namespace and scoping.*
5. *Write a Python program to demonstrate global, local and built in namespace.*

4.4 Python Packages

4.4.1 Introduction to Packages

- Packages are namespaces which contain multiple packages and modules. They are simply directories.
- Along with packages and modules the package contains a file named _ _init_ _.py. In fact **to be a package, there must be a file called _ _init_ _.py in the folder.**
- Packages can be nested to any depth, provided that the corresponding directories contain their own __init__.py file.
- The arrangement of packages is as shown in Fig. 4.4.1.

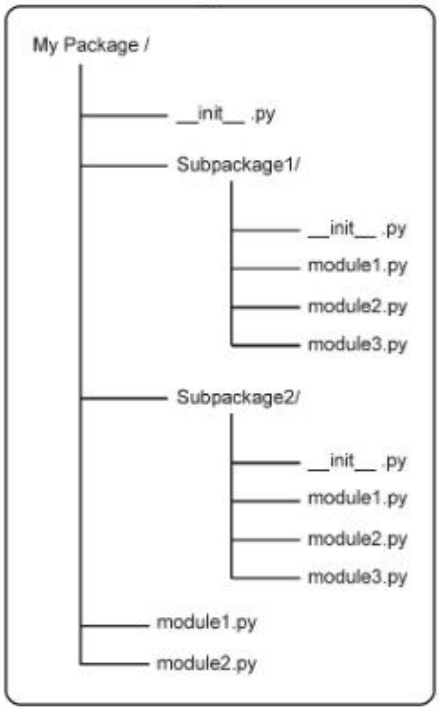


Fig. 4.4.1

For example :

We can access the package and various subpackages and modules in it as follows :

```
import My_Package #loads My_Package/__init__.py
import My_Package.module1 #loads My_Package/module1.py
from My_Package import module2
import My_Package.SubPackage1
```

- When we import Main package then there is no need to import subpackage. For example

```
import My_Package
My_Package.SubPackage1.my_function1()
```

- Thus we need not have to import SubPackage1.
- The primary use of __init__.py is to initialize Python packages. Simply putting the subdirectories and modules inside a directory does not make a directory a package, what it needs is a __init__.py inside it. Then only Python treats that directory as package.

4.4.2 Writing Python Packages

- To understand how to create a package, let us take an example. We will perform arithmetic operations addition, multiplication, division and subtraction operations by creating a package. Just follow the given steps to create a package.

Step 1 : Create a folder named **MyMaths** in your working drive. I have created it on D: drive.

Step 2 : Inside MyMaths directory create **__init__.py** file. Just create an empty file.

Step 3 : Create a folder named **Add** inside **MyMaths.** Inside **Add** folder create file named **addition.py.** The **addition.py** will be as follows

addition.py

```
def add_fun(a,b):
  return a+b
```

Step 4 : Similarly the folder **Sub** insider MyMaths is created. Inside which create a file **subtraction.py.** The code for this file is

subtraction.py

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

Step 5 : Similarly the folder **Mul** insider MyMaths is created. Inside which create a file **multiplication.py.** The code for this file is

multiplication.py

```
def mul_fun(a,b):
  return(a*b)
```

Step 6 : Similarly the folder **Div** insider MyMaths is created. Inside which create a file **division.py.** The code for this file is

division.py

```
def div_fun(a,b):
  return (a/b)
```

Step 7 : The overall directory Structure will be as given below. Note that _pycache_ is automatically generated directory.

Step 8 : Now on the D: drive create a driver program which will invoke all the above functionalities present in the MyMaths package. I have named this driver program as **testing_arithmetic.py**. It is as follows

```
import MyMaths.Add.addition
import MyMaths.Mul.multiplication
import MyMaths.Div.division
import MyMaths.Sub.subtraction
print("The addition of 10 and 20 is: ",MyMaths.Add.addition.add_fun(10,20))
print("The multiplication of 10 and 20 is: ",MyMaths.Mul.multiplication.mul_fun(10,20))
print("The division of 10 and 5 is: ",MyMaths.Div.division.div_fun(10,5))
print("The subtraction of 20 and 10 is: ",MyMaths.Sub.subtraction.sub_fun(20,10))
```

We need to import the modules in this way, to use the respective functionalities

Step 9 : Now just run this testing program and you will get following output

Output

```
The addition of 10 and 20 is:30
The multiplication of 10 and 20 is:200
The division of 10 and 5 is:2.0
The subtraction of 20 and 10 is:10
>>>
```

4.4.3 Using Standard Packages

In this section we will discuss how to install the standard packages of Python and use them

(1) Installation of numpy

Step 1 : Open command prompt and type the command

C:\> python -m pip install numpy

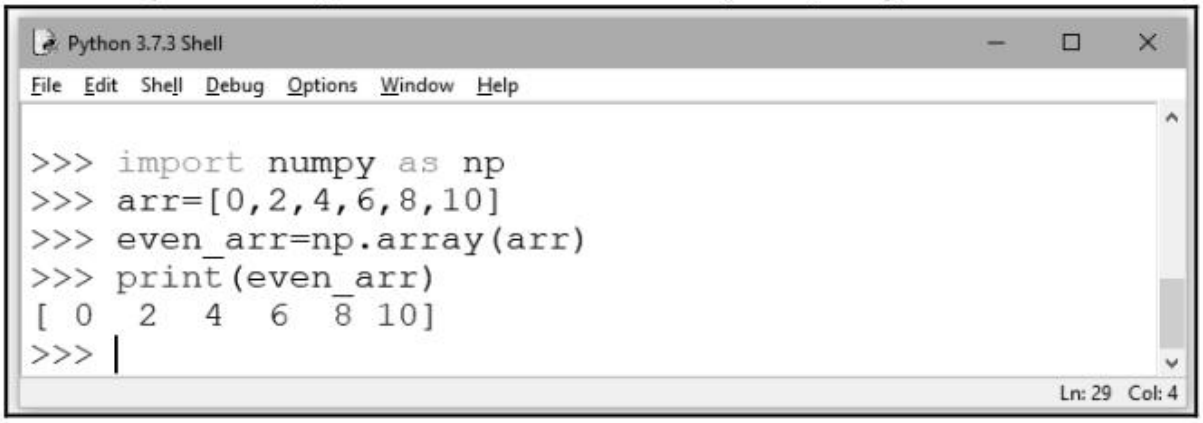
Step 2 : The package will be installed on your PC. On successful installation you will get the following message on your command prompt window.

Successfully installed numpy-1.17.4

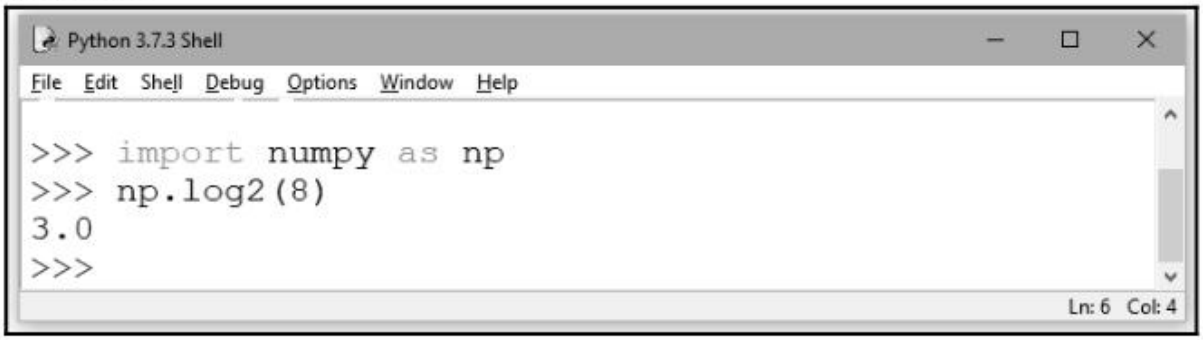
You are using pip version 19.0.3, however version 19.3.1 is available.

What is numpy Package ?

- Numpy is the core library for **scientific computing** in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.
- Advantages of using Numpy with Python :
  - Array oriented computing.
  - Efficiently implemented multi-dimensional arrays.
  - Designed for scientific computation.
- For example – Following screenshot show how to use arrays using numpy.



Some other scientific function such as finding log can be used using the numpy package. Here is an illustration -



Or we can also use the function from numpy as follows

```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>> from numpy import *
>>> log2(8)
3.0
>>>
Ln: 9 Col: 4
```

(2) Installation of scipy

Similar to numpy installation procedure, issue the following command at command prompt

Python –m pip install scipy

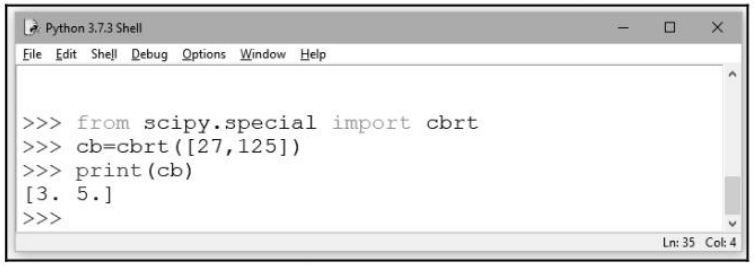
The package will start installing. Following screenshot illustrates it.

```
C:\Windows\System32\cmd.exe - python -m pip install scipy
C:\>python -m pip install scipy
Collecting scipy
  Downloading https://files.pythonhosted.org/packages/d7/b9/224bb1aa3db49f3c45b
9e1450a0dd5960c4ab07eba945ec1d7b135547946/scipy-1.3.2-cp37-cp37m-win32.whl (27.
2MB)
    3% |                                | 901kB 570kB/s eta 0:00:47
```

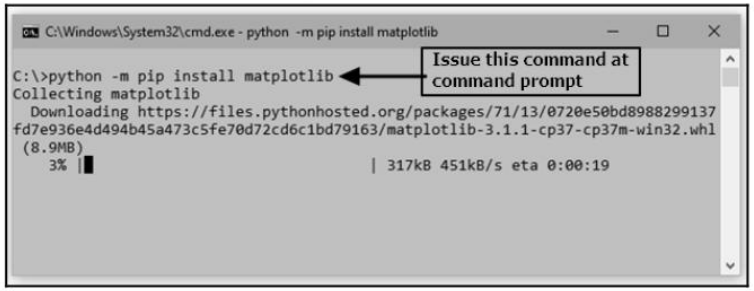
What is scipy Package ?

- SciPy is a collection of mathematical algorithms and convenience functions built on the NumPy extension of Python.
- SciPy is an Open Source Python-based library, which is used in mathematics, scientific computing, Engineering, and technical computing.
- SciPy also pronounced as 'Sigh Pi.'
- SciPy is a fully-featured version of Linear Algebra while Numpy contains only a few features.
- Most new Data Science features are available in Scipy rather than Numpy.
- **Example :** Following example shows how to compute cuberoot of values –

Notes Page: 120 of 184 70%



(3) Installation of matplotlib



What is matplotlib Package ?

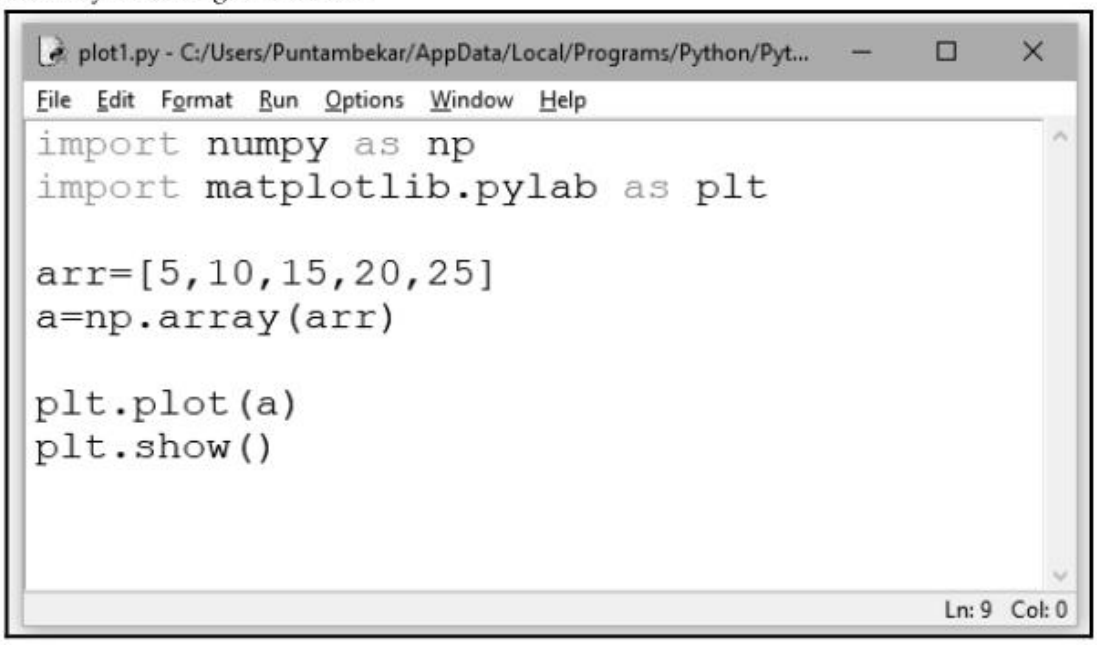
- Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy.
- Matplotlib consists of several plots like line, bar, scatter, histogram etc.
- We have to import the **pyplot** module of **matplotlib** library under the alias plt.
- With the help of plot function we can start preparing data for plotting and finally with the help of show function we can display the drawing on the screen.
- Let us understand how to plot the objects with the help of matplotlib.

Example 4.4.1 : Write a Python program to plot a line.

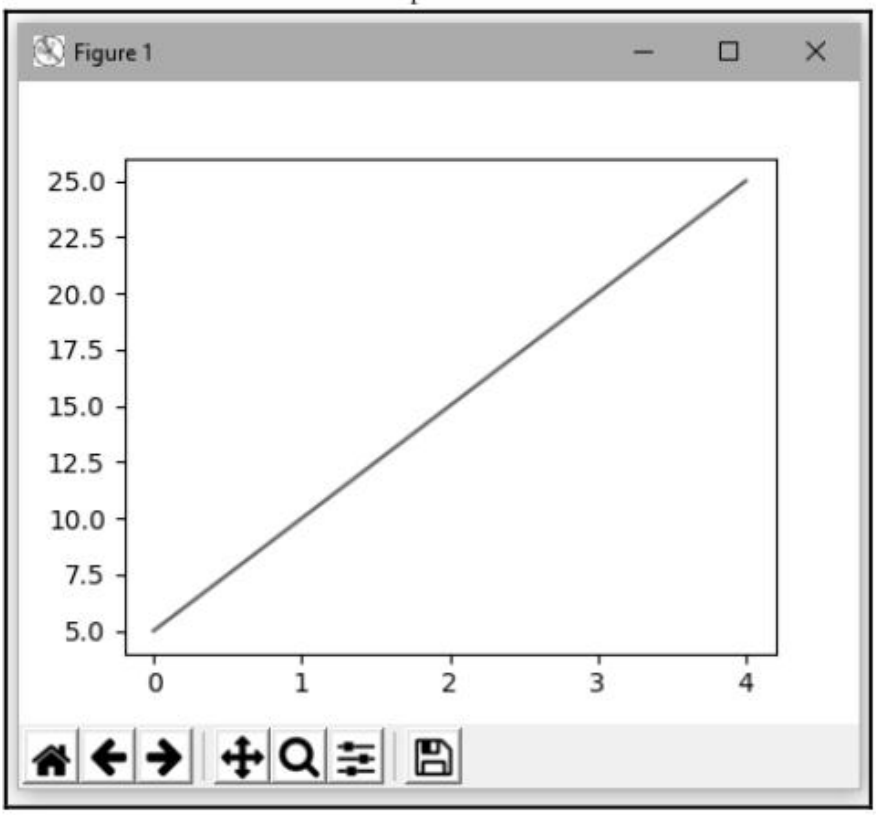
Solution : Step 1 : Open **File->New File** and type the following code in the file. Save this file with suitable name.

```
import numpy as np
import matplotlib.pylab as plt

arr=[5,10,15,20,25]
a=np.array(arr)

plt.plot(a)
plt.show()
```

It is shown by following screenshot.



Step 2 : Press F5 and run the above code. The output will be as follows-



Code Explanation : In above Python Program,

(1) We have imported two libraries

```
import numpy as np
import matplotlib.pylab as plt
```

The **numpy** library helps us to create an data array. This array stores the points on the graph that can be used for plotting.

(2) The plot() is used to draw the line with the help of data points provided through array **a.**

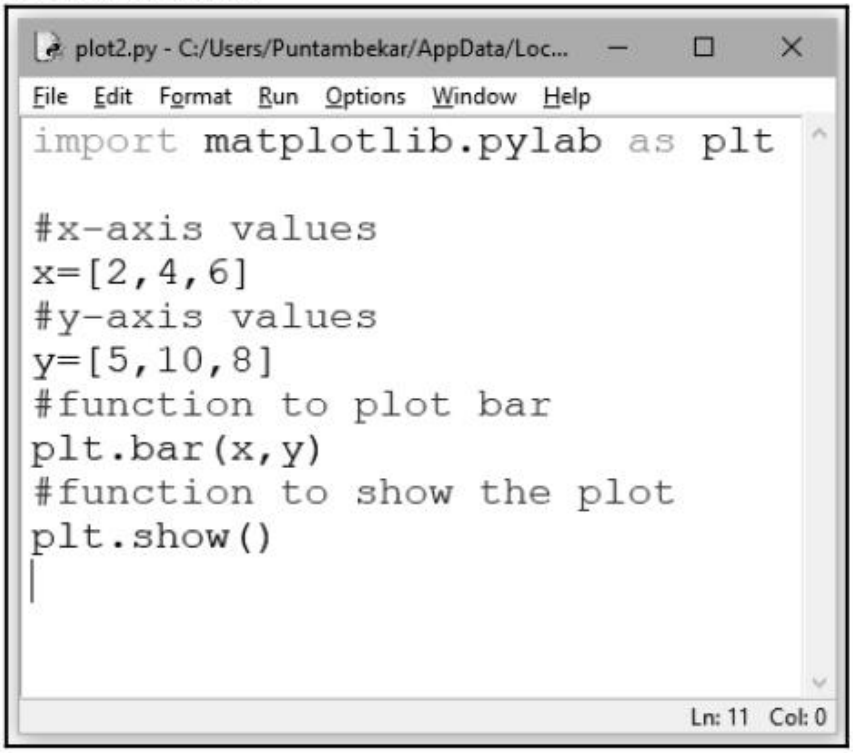
(3) Then the show() method is used to display the plotted line on the window.

Example 4.4.2 : Write a Python program to plot the bar graph.

Solution : The code is as follows

```
import matplotlib.pylab as plt
#x-axis values
x=[2,4,6]
#y-axis values
y=[5,10,8]
#function to plot bar
plt.bar(x,y)
#function to show the plot
plt.show()
```

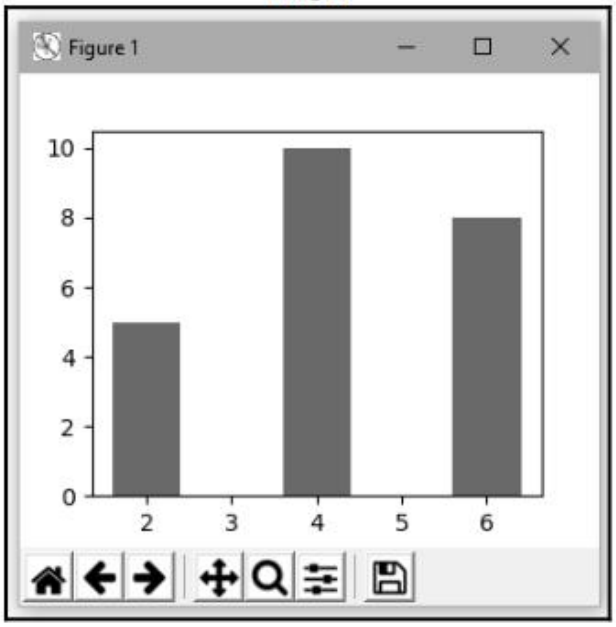
The screen shot for the above code is



Press F5 to run the above code. The output will be –

4 70%

Output

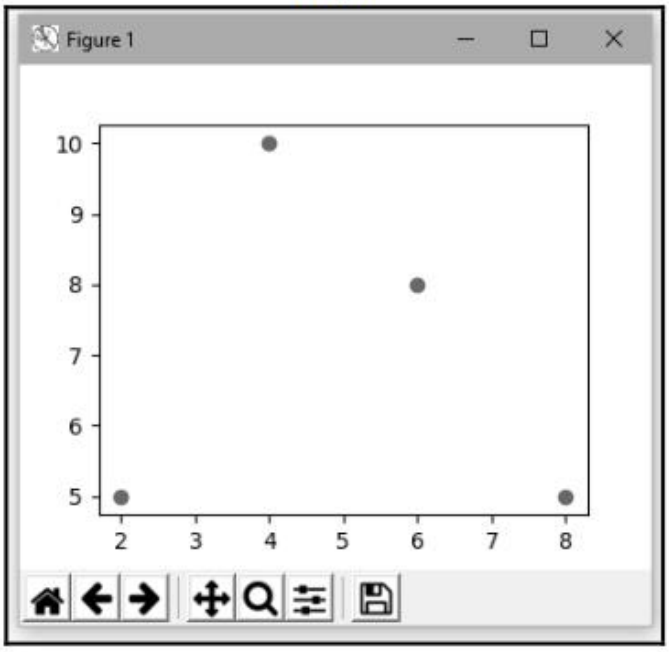


Example 4.4.3 : Write a Python program to plot the scattered points.

Solution :

```
import matplotlib.pylab as plt
#x-axis values
x=[2,4,6,8]
#y-axis values
y=[5,10,8,5]
#function to plot scatter
plt.scatter(x,y)
#function to show the plot
plt.show()
```

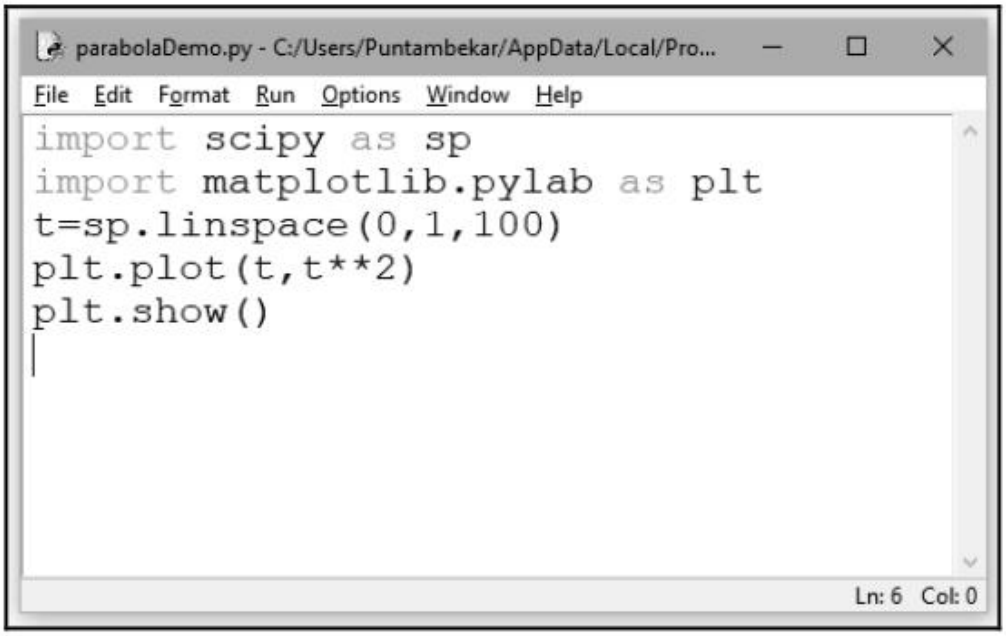
Output



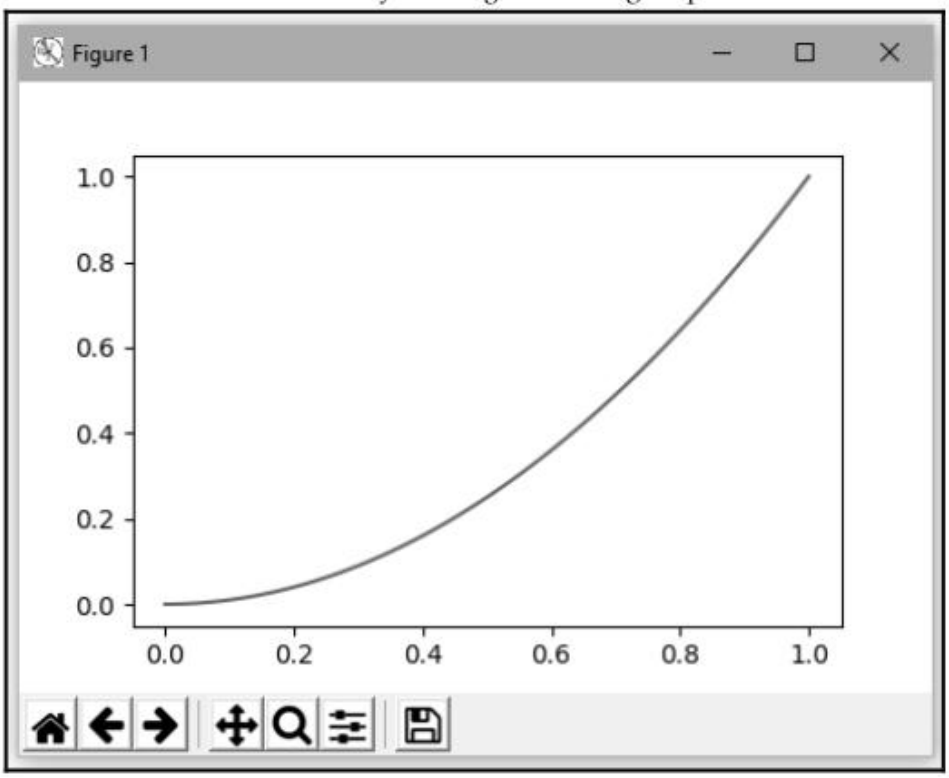
Example 4.4.4 : Write a Python program to plot the parabola.

Solution : Open **File->New File** and type the following code in the file. Save this file with suitable name.

```
import scipy as sp
import matplotlib.pylab as plt
t=sp.linspace(0,1,100)
plt.plot(t,t**2)
plt.show()
```



Step 2 : Press F5 to run the above code and you will get following output.



(4) Installation of pandas

Similar to above installation procedures issue the following command at command prompt.

Python –m pip install pandas

The package will start installing. Following screenshot illustrates it.

Finally you will get the successful installation message on your command prompt.

What is Pandas Package ?

- Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with structured (tabular, multidimensional, potentially heterogeneous) and time series data both easy and intuitive.
- Pandas is well suited for many different kinds of data :
- **Tabular data** with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet
- **Ordered and unordered** (not necessarily fixed-frequency) time series data.
- **Arbitrary matrix data** (homogeneously typed or heterogeneous) with row and column labels
- Any other form of **observational / statistical data sets**. The data actually need not be labeled at all to be placed into a pandas data structure.
- There are three data structures used in Panda
  - Series
  - DataFrame
  - Panel
- **Series :** It is a one dimensional homogeneous array. The size of this array is immutable, i.e. this size can not be changed.

A	B	C	D	E	F

Following is a Python script that represents the series

SeriesDemo.py

```
import pandas as pd
import numpy as np
a=np.array(['A','B','C','D','E'])
s=pd.Series(a)
print(s)
```

34 − + 70%

Output

```
0   A
1   B
2   C
3   D
4   E
dtype: object
>>>
```

DataFrame : It is a two dimensional heterogeneous array. The size is mutable i.e. it can be changed.

EmpNo	Name	Salary	Designation
101	AAA	10000	Accountant
102	BBB	15000	Executive
103	CCC	25000	Manager

The DataFrame can be implemented using following Python script.

Dframe.py

```
import pandas as pd
import numpy as np
a=[[10,20],[30,40]]
data_fr=pd.DataFrame(a)
print(data_fr)
```

Output

```
0  1
0 10 20
1 30 40
>>>
```

Another Example : We can create labeled column using dictionary

```
import pandas as pd
import numpy as np
dic={'EmpNo':[101,102,103],'Name':['AAA','BBB','CCC'],'Salary':[10000,15000,25000]}
data_fr=pd.DataFrame(data=dic)
print(data_fr)
```

Output

```
   EmpNo  Name  Salary
0    101   AAA   10000
1    102   BBB   15000
2    103   CCC   25000
>>>
```

- **Panel :** It is a three dimensional labeled array. The size is mutable.

  Syntax : The panel can be created using following syntax

  pandas.Panel(data, items, major_axis, minor_axis, dtype, copy)

where

- **data :** The data can be arrays,series,dict,list or another DataFrame.
- **items :** axis=0
- **major_axis:** axis=1

34 70%

- **minor_axis:** axis=2
- **dtype:** The data type of each column
- **copy:** Copy data.

Review Questions

1. *What is package ? How will you create package in Python ?*
2. *What is packages ?*
3. *Explain the numpy package with suitable example.*
4. *What is the use of matplotlib package? Give an example.*

❑❑❑

UNIT - V

5 Object Oriented Programming in Python

5.1 Introduction to Object Oriented Programming

- In this language everything is modeled as **object**. Hence is the name.
- This language has a modular programming approach in which data and functions are bound in one entity called class.
- This programming paradigm has gained a great popularity in recent years because of its characteristic features such as data abstraction, encapsulation, inheritance, polymorphism and so on.
- Examples of object oriented programming languages are - Smalltalk, C++, Java, Python.

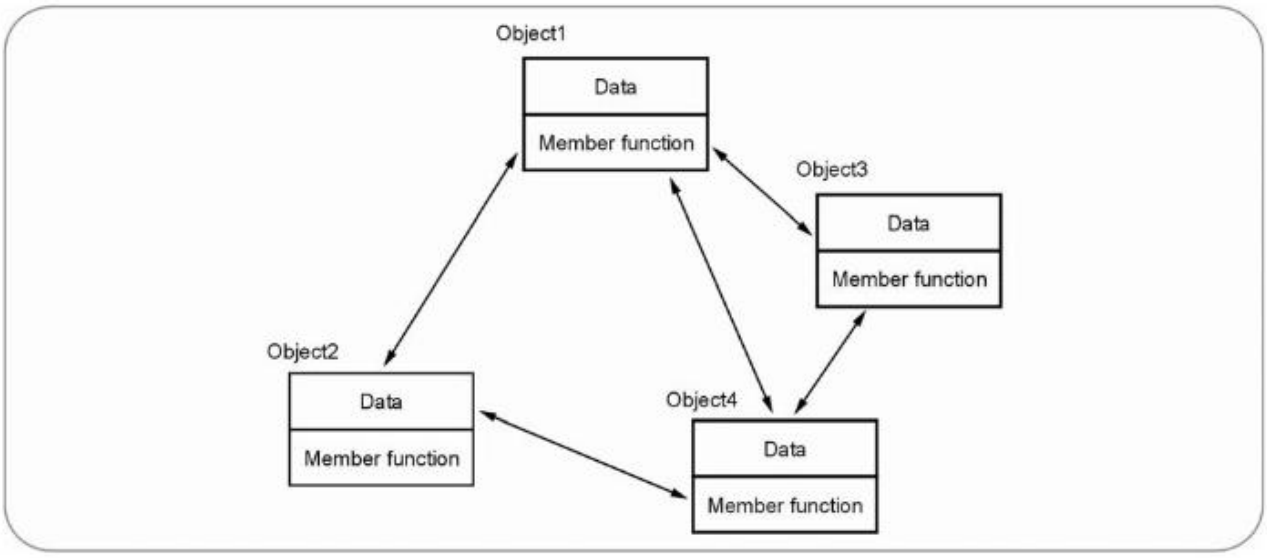


Fig. 5.1.1 Structure of object oriented programming

Advantages

1. It provides a modular programming approach.
2. It provides abstract data type in which some of the implementation details can be hidden and protected.
3. Modifying the code for maintenance become easy, due to modules in the program. Modification in one module can not disturb the rest of the code.
4. Finding bugs become easy.

34 − | + 70%

5. Object oriented programming provides good framework for code library from which the software components can be easily used and modified by the programmer.

Disadvantages

1. Sometimes real world objects can not be realized easily. Hence it is **complex to implement**.
2. Some of the members (data or methods) of the class are **hidden**. Hence they can not be accessed by the objects of other class.
3. In object oriented programming everything is arranged in the form of classes and modules. For the **lower level** applications it is not desirable feature.

Difference between procedural and object oriented programming approach

Sr. No.	Procedural Programming Language	Object Oriented Programming Language (OOP)
1.	The procedural programming executes **series of procedures** sequentially.	In object oriented programming approach there is a **collection of objects.**
2.	This is a **top down** programming approach.	This is a **bottom up** programming approach.
3.	The major focus is on **procedures or functions**.	The main focus is on **objects.**
4.	**Data reusability** is not possible.	**Data reusability** is one of the important feature of OOP.
5.	Data hiding is not possible.	**Data hiding** can be done by making it private.
6.	It is **simple** to implement.	It is **complex** to implement.
7.	For example : C, Fortran, COBOL	For example : C++, JAVA

Review Questions

1. *What is object oriented programming? Give the advantages and disadvantages of it.*
2. *What is the difference between procedural and object oriented programming languages ?*

5.2 Features of Object Oriented Programming

5.2.1 Classes

- Class is a basic entity in object oriented programming.
- In any object oriented program the class is declared using the keyword **class**.
- Every class consists of data members and methods.
- For example -

```
Class Student:
    Roll=101
    Name="ABC"
obj=Student()
print("Roll Number: ",obj.Roll)
print("Name: ",obj.Name)"
```

5.2.2 Objects

- An object is an instance of a class.

- **For example**

```
Obj=Student() #creation of object for class 'Student'
```

- Objects are runtime entities of the object oriented programs.
- Objects communicate with each other using messages. While communicating with each other, it is not necessary to know the details of communicating object.
- Thus the internal details of each object can be hidden from another object.

Difference between class and objects

Sr.No.	Classes	Objects (instances)
1.	The class is collection of data and the functions. These functions are used to manipulate data.	An object is an instance of a class.
2.	The class represent the basis for the object.	Object represents the real world entity.
3.	Single class can create multiple objects.	Every object is created from one at only one class.
4.	Example - Class vehicle	Example - Alto, ZEN, SANTRO are the objects of class vehicle.

5.2.3 Methods and Message Passing

- An object provides some service. A message for an object is a request made for execution of **method** for obtaining the service.

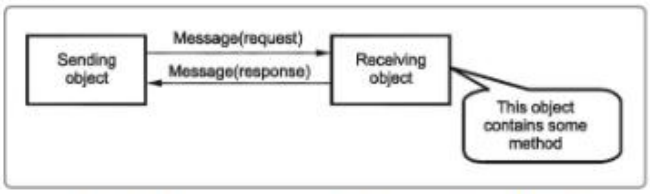


Fig. 5.2.1 Communication between two objects

- Another object containing the method, executes that method and then responds back by passing the message. Thus action is initiated in object oriented programming by the transmission of a message to the object responsible for the action.

A **message** is a request to an object to invoke one of its methods. A message therefore contains,

- The **name** of the method and
- The **arguments** of the method.

5.2.4 Inheritance

Inheritance *is a property by which the new classes are created using the old classes. In other words the new classes can be developed using some of the properties of old classes.*

- The old classes are referred as base classes and the new classes are referred as **derived classes**. That means the derived classes inherit the properties (data and functions) of base class. This allows to reuse the existing properties of base class in derived class.

4 70%

For example

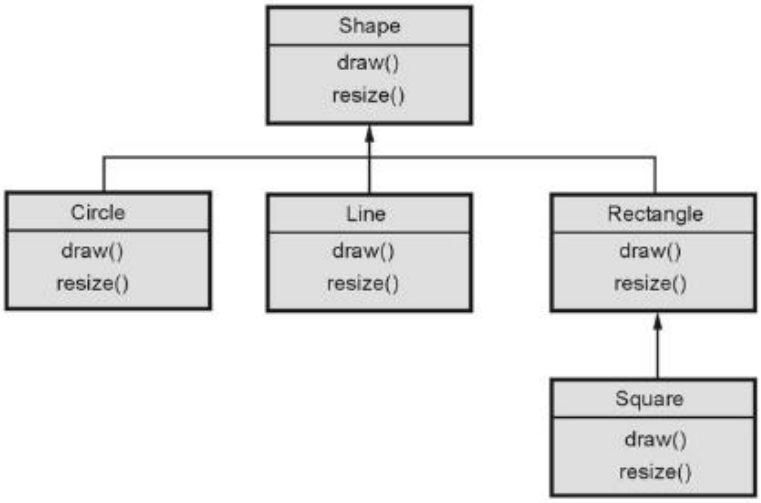


Fig. 5.2.2 Example of inheritance

- Here the shape is a base class from which the **circle, line** and **rectangle** are the derived classes. These classes inherit the functionality **draw()** and **resize()**. Similarly the rectangle is a base class for the derived class square. Along with the derived properties the derived class can have its own properties. For example, the class circle may have the function like **backgrcolor( )** for defining the background color.

5.2.5 Polymorphism

- Polymorphism means many structures.
- ***Polymorphism*** *is the ability to take more than one form and refers to an operation exhibiting different behavior in different instances (situations).*
- The behavior depends on the type of data used in the operation. It plays an important role in allowing objects with different internal structures to share the same external interface.
- Object oriented programs use polymorphism to carry out the same operation in a manner customized to the object. Without polymorphism, you would have to create separate module names for each method.
- **For example** the method **clean** is used to clean a **dish** object, one that cleans a **car** object, and one that cleans **vegetable** object.
- With polymorphism, you create a single "clean" method and apply it for different objects.

5.3 Creating Classes and Objects

- Python is an object oriented programming language. That means almost all the code is implemented using a special construct called classes.
- Programmers make use of classes to keep the data values and methods together.
- Object is simply a collection of data (variables) and methods (functions) that act on those data. And, class is a blueprint for the object.

Terminologies used in object oriented programming

1. **Class :** Class is a collection of data attributes and the methods that can access or manipulate the data attributes. For defining the class in Python the keyword **class** is used.
2. **Class variable :** This is a kind of variable which is shared by all instances of class. The class variable are defined within a class but outside the class method.
3. **Data member :** The class variable that holds data associated with a class and its objects.
4. **Instantiation :** The creation of instance of a class.
5. **Class method :** It is a special type of function which is defined inside the class definition.
6. **Object :** It is an instance of a class. An object comprises both data members and methods.
7. **Instantiation :** Creating a new object is called instantiation and the object is called **instance** of a class.

5.3.1 How to Create a Class and Object ?

The keyword **class** is used to define the class followed by the class name.

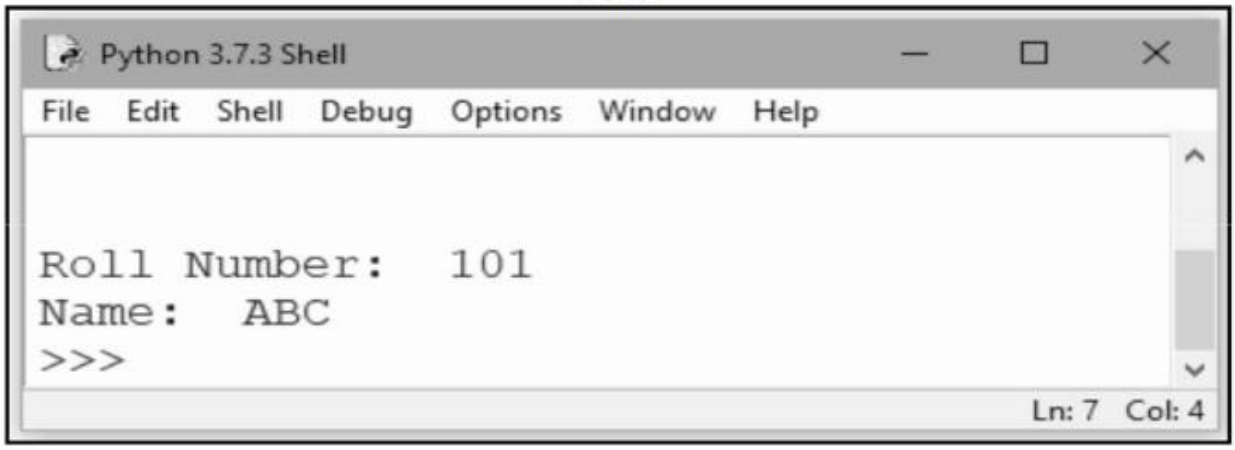
Syntax of class

```
Class class_name:
 Statement1
 Statement2
 ...
Statementn
Syntax of Object
Object_name=class_name()
```

For example

```
Class Student:
 Roll=101
 Name="ABC"
obj=Student()
print("Roll Number: ",obj.Roll)
print("Name: ",obj.Name)
```

Output



5.3.2 Class Method and Self Object

- The class methods are the functions defined inside the class. These are actually the normal functions but belong to that class.
- The class method defines the first argument as **self**. This is always defined as the first argument. If there is no argument to the class method then only self will be the argument for that method. And if there are some arguments present in the class method, then always the first argument will be **self** and then remaining will be the list of arguments of that class method.
- The **self**-argument refers to the object itself.
- The class methods are accessible from outside the class using object. Hence these methods are also called as **instance methods**.

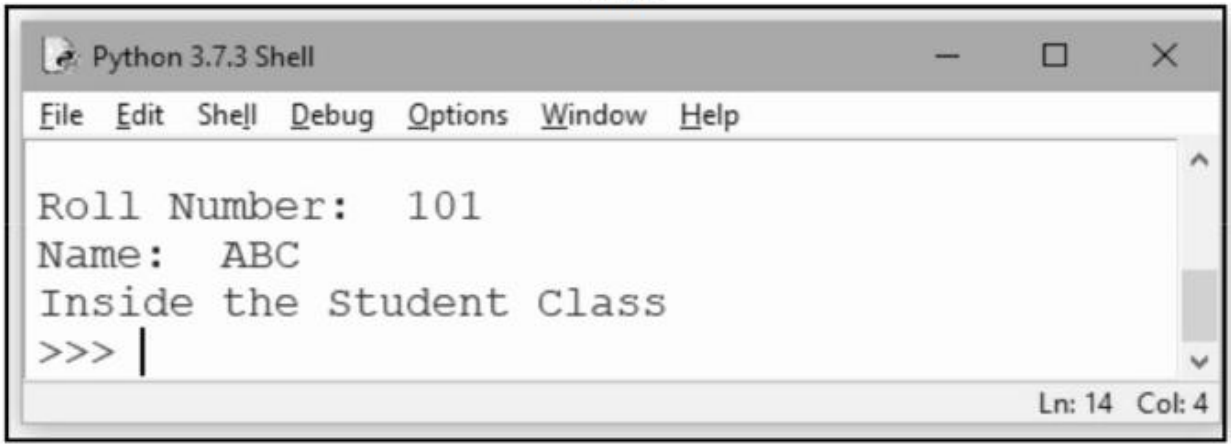
Programming example

In the following example we define the class method and demonstrate the use of **self** object

StudentDemo.py

```
class Student:
    Roll=101 #Data member
    Name='ABC' #Data member
    def show(self):
        print("Inside the Student Class")#definition of class method
obj=Student() #creating object of class
print("Roll Number: ",obj.Roll)#Accessing data member using object
print("Name: ",obj.Name)#Accessing data member using object
obj.show()#Accessing class method using object
```

Output



5.3.3 Public and Private Members

- Public variables are those variables that are defined in the class and can be accessed from within the class and from outside the class.
- All members in a Python class are public by default.

For example

StudentDemo.py

```
class Student:
  def __init__(self,R,N):
    self.Roll=R
    self.Name=N
```

These are the public members of class **Student**

```
obj=Student(101,"Jayashree")
print("Roll No: ",obj.Roll) #Accessing public variable outside the class
print("Name: ",obj.Name)
```

Output

```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
Roll No:  101
Name:  Jayashree
>>>
Ln: 22 Col: 4
```

Program explanation : In above program,

Two variables - **Roll** and **Name** are the public variables and we are accessing them outside the class.

- Private variables are those variables that are defined in the class but can be accessible by the methods of that class only. The private variables are used with double underscore(__) prefix.

For example

StudentDemo.py

```
class Student:
  def __init__(self,R,N):
    self.__Roll=R #private variable 'Roll'
    self.__Name=N #private variable 'Name'

obj=Student(101,"Jayashree")
print("Roll No: ",obj.__Roll) #trying to access private attribute
                              #outside the class and that will cause error
```

Output

```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
    print("Roll No: ",obj.__Roll)#trying to access private attribute
AttributeError: 'Student' object has no attribute '__Roll'
>>>
Ln: 34 Col: 4
```

4 70%

Program explanation : In above program,

1) There are two private variables - Roll and Name. Note that to make these variables private they are written using the prefix as double underscore.
2) When we try to access these variables outside the class using the object of that class, it gives an Attribute error. Refer the screenshot for the output.

Example 5.3.1 : Write a definition for a class named Circle with attributes center and radius, where center is a Point object and radius is a number.
Instantiate a Circle object that represents a circle with its center at (150, 100) and radius 75.
Solution :

CircleDemo.py

```
class Point():
   x=0
   y=0
class Circle():
   radius=0

def display(pt,cir):
      print("Center: ",pt.x,pt.y)
      print("Radius: ",cir.radius)

radius=int(input("Enter radius of circle: "))
a=int(input("Enter x co-ordinate of center: "))
b=int(input("Enter y co-ordinate of center: "))
pt=Point()
pt.x=a
pt.y=b
cir=Circle()
cir.radius=radius
display(pt,cir)
```

Output

```
Python 3.6.2 Shell
File Edit Shell Debug Options Window Help
>>>
 RESTART: C:/Users/Puntambekar/AppData/Local/Programs/Python/Python36-32
/CircleDemo.py
Enter radius of circle: 75
Enter x co-ordinate of center: 150
Enter y co-ordinate of center: 100
Center:  150 100
Radius:  75
>>>
Ln: 78 Col: 4
```

Review Question

1. *Explain how to create class and object in object oriented programming.*

5.4 Method Overloading and Overriding

5.4.1 Method Overloading

- Method overloading is an ability of a function to behave in different ways based on the number of parameters passed to it.
- In method overloading the same function name is used for different behavior. Following is an example program which shows method overloading concept.

```
OverloadDemo.py
class Color:
  def selectColor(self, name=None):
    if name is not None:
      print('My Favourite color is ' + name)
    else:
      print('I do not like any color')

# Create instance
obj = Color()
# Call the method
obj.selectColor()
# Call the method with a parameter
obj.selectColor('White')
```

Output

```
I do not like any color
My Favourite color is White
>>>
```

5.4.2 Method Overriding

- Method overriding is a mechanism in object oriented programming which allows to change the implementation function in child class which is defined in parent class.
- There are two essential criteria for method overriding :
  1. For performing method overriding, there must be inheritance.
  2. The function that is redefined in child class must have same number of arguments in child class.

Following is a Python program that illustrates the concept of method overriding.

```
class Parent: #Parent Class
  def __init__(self):
    self.age=45

  def display(self):
    print("Age: ",self.age)

class Child(Parent): #Child Class
  def display(self): #Note that the same method name but different implementation
    self.age=self.age-25
    print("Age: ",self.age)
```

34 − + 70%


```
p=Parent()
c=Child()
print("Parent's Age:")
p.display()
print("Child's Age:")
c.display()
```

Output

```
Parent's Age :
Age : 45
Child's Age:
Age : 20
>>>
```

Difference between Method Overloading and Method Overriding

Method Overloading	Method Overriding
Different number of parameters can be passed to the function.	The number of parameters that are passed to the function are the same.
The overloaded functions may have different return types.	In the method overriding all the methods will have same return types.
Method overloading is performed within a class.	Method overriding is always performed between two classes that have **inheritance relationship.**

Review Questions

1. *What is method overloading ? Write a Python program that implements the concept of method overloading.*
2. *What is method overriding ? Write a Python program that implements the concept of method overriding.*
3. *What is the difference between method overloading and method overriding.?*

5.5 Data Hiding

- **Definition :** Data hiding is a mechanism which does not allow to access the attributes outside the class.
- Data hiding is an important property of object oriented programming.
- In Python the data hiding capability can be achieved by making the members of the class **private** or **protected.**
- In Python, the **double underscore** is used before the name to make the attribute **private.**
- Similarly the **single underscore** is used before the name to make the attribute **protected.**

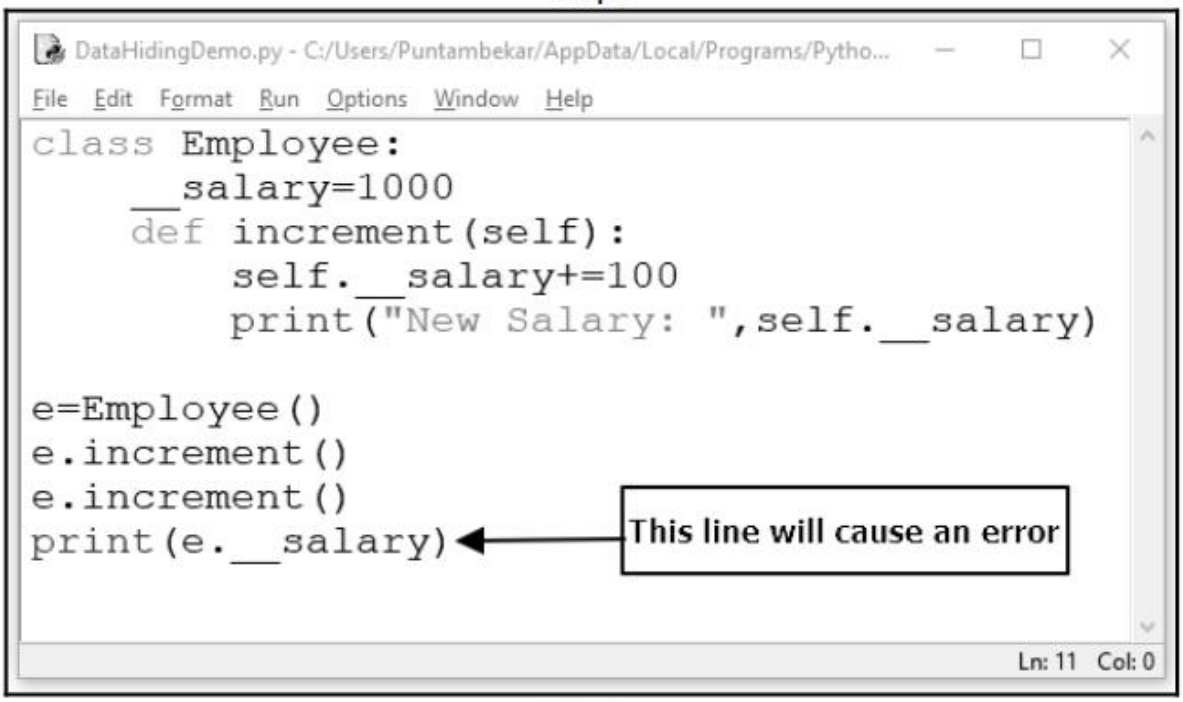
Python Program on Data Hiding

```
class Employee:
  __salary=1000
  def increment(self):
    self.__salary+=100
    print("New Salary: ",self.__salary)
```

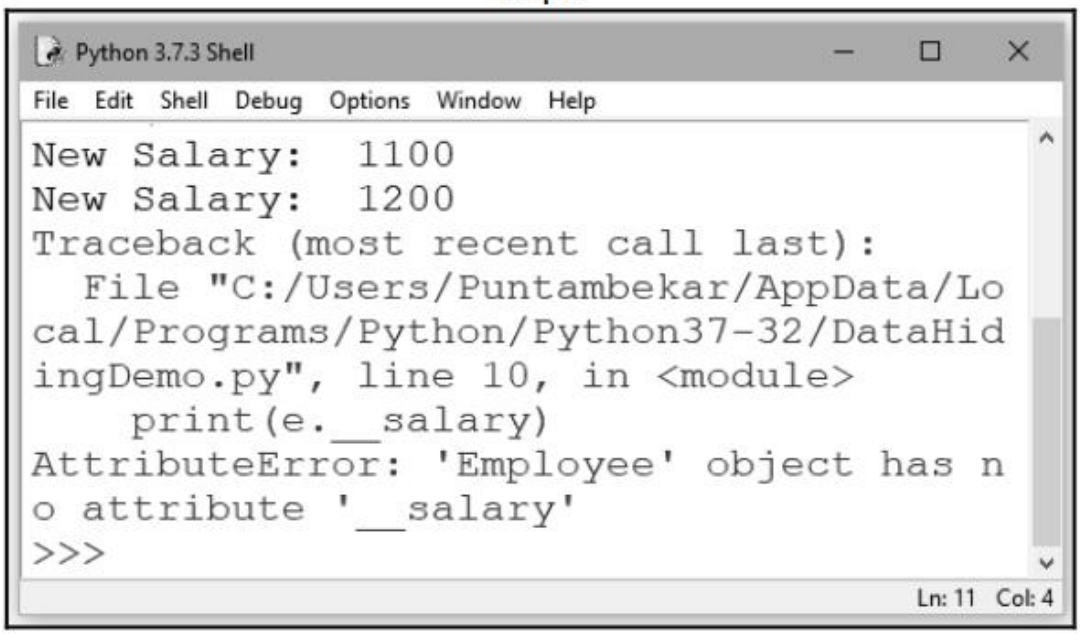
```
e=Employee()
e.increment()
e.increment()
print(e.__salary)
```

This can be represented by following screenshot.

Output



Output



Code Explanation : In above Python Program,

(1) We have created on class named **Employee.** This class contains a variable named **salary.** This variable is a private variable.

(2) The **salary** is incremented inside the method named **increment.** This method belongs to the same class **Employee.**

(3) Outside the class, we have created an instance or an object of the class **Employee** by the statement

```
e=Employee()
```

(4) Using this instance or object e we can invoke or call the method **increment** with the help of dot operator. In our program we have invoked **increment** function twice. Hence salary will become 1100 and 1200. Hence is the output.

(5) But as soon as we try to access the private variable **salary** directly with the help of object **e** outside the class **Employee** then the error occurs. That means we can not access the private variable directly outside the class. This achieves data hiding property.

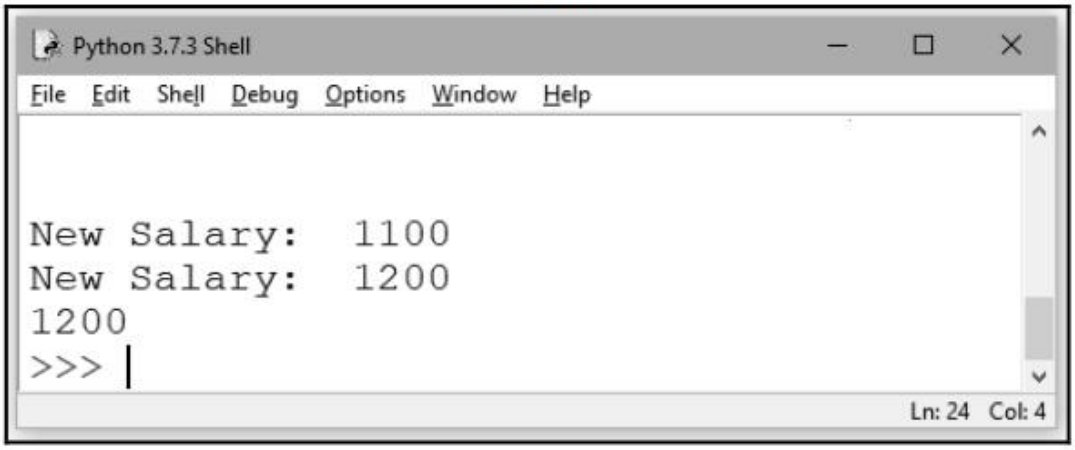
However, Python allows to access the private member of the class outside the class using a special construct

```
Object._classname__attributeName
```

Following program illustrates this. Note that the following program is just similar to above program. The only change made in above program is shown by boldfaced line -

```
class Employee:
  __salary=1000
  def increment(self):
    self.__salary+=100
    print("New Salary: ",self.__salary)

e=Employee()
e.increment()
e.increment()
print(e._Employee__salary)
```

Output



This shows that accessing the private attributes in Python is not easy but nothing in Python is truly private.

Review Question

1. *How to achieve data hiding concept in a Python program. Explain it with suitable example.*

5.6 Data Abstraction

Definition : Data abstraction is a mechanism in which the abstract class is created. The abstract class is a class in which abstract method is defined. An abstract method is a method which does not have any implementation or definition. However, this abstract method is defined inside the child classes.

Following is an example program in which abstract class **Color** is created.

```
from abc import ABC,abstractmethod
class Color:
    @abstractmethod
    def showColor(self):
        none

class Red(Color):
    def showColor(self):
        print("This is a red color")

class Green(Color):
    def showColor(self):
        print("This is a green color")

def main():
    r=Red()
    r.showColor()

    b=Green()
    b.showColor()

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

This is a abstract class with abstract method **showColor**

Output

```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help

This is a red color
This is a green color
>>>
Ln: 7 Col: 4
```

Code Explanation : In above Python program,

(1) In Python, it is not possible to use abstract classes by default. For supporting abstraction, python comes up with a module which provides the base for defining **Abstract Base classes(ABC)** and that module name is **ABC.** The **ABC** works by marking methods of the base class as abstract and then registering concrete classes as implementations of the abstract base. A method becomes an abstract by decorated it with a keyword @abstractmethod.

4 70%

Hence at the beginning of the program we have written,

```
from abc import ABC,abstractmethod
```

(2) Now we have defined an abstract class named **Color.** This class contains an abstract method named **showColor.** By writing **none** we are having no implementation code for this method.

(3) Then we have written two child classes namely **Red** and **Green** in which the method **showColor** is redefined.

(4) Inside the main() function the objects for **Red** and **Blue** classes are created. Using the corresponding objects the **showColor** method of respective classes is invoked.

(5) Every module in Python has a special attribute called **__name__.** The value of **__name__** attribute is set to '**__main__**' when module run as main program.

Review Questions

1 *What is data abstraction ?*

2. *How to use the abstract method in Python program ? Give suitable example.*

5.7 Inheritance and Composition Classes

Inheritance

Definition : Inheritance is a relationship which is similar to a family tree. In this relationship **type-of** relations is defined. For example – A Person can be a Student or a Teacher. It is represented as,

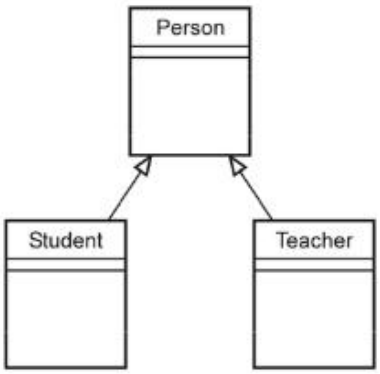


Fig. 5.7.1 Representing Inheritance

Types of Inheritance

There are two form of inheritance – single inheritance and multiple inheritance.

(1) Single Inheritance : In single inheritance, there is one base class and one derived class. Following figure represents single inheritance.

4 70%

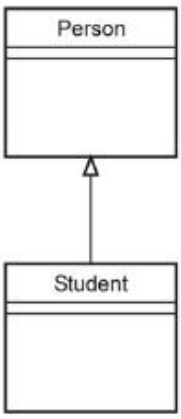
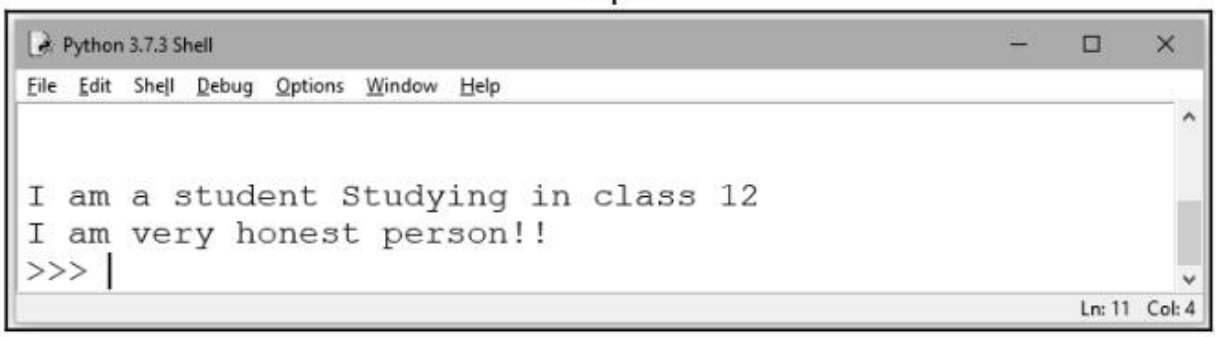


Fig. 5.7.2 Single Inheritance

Example program to implement Single level inheritance

```
class Person:
   def display(self):
      print("I am very honest person!!")
#child class Dog inherits the base class Animal
class Student(Person):
   def show(self):
      print("I am a student Studying in class 12")
s = Student()
s.show()
s.display()
```

Output



(2) Multiple Inheritance : In multiple inheritance, more than one child classes can be derived from single base class. Following figure represents the multiple inheritance –

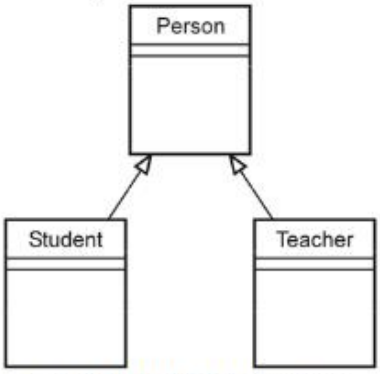


Fig. 5.7.3 Multiple inheritance

Example Program to Implement Inheritance

```
class Person : #Base class
    def __init__(self, first, last) :
        self.firstname = first
        self.lastname = last

    def Name(self):
        return self.firstname + " " + self.lastname


class Student(Person): #Derived Class
    def __init__(self, first, last, marks) :
        Person.__init__(self,first, last)
        self.studMarks = marks

    def GetStudent(self) :
        return self.Name() + ", " + self.studMarks

class Teacher(Person): #Derived Class
    def __init__(self,first,last,subject):
        Person.__init__(self,first,last)
        self.subject = subject

    def GetTeacher(self):
        return self.Name() + ", " + self.subject

print("The student details are ...\n")
s = Student("AAA", "BBB", "100")
print("Student Name and his marks: ",s.GetStudent())
print("--------------------------------------------------")
print("The Teacher details are ...\n")
t = Teacher("PPP", "TTT", "Mathematics")
print("Teacher Name and subject taught: ",t.GetTeacher())
```

Base class or parent class

First derived class or child class

Second derived class or child class

Calling the child classes and displaying the contents of these classes.

Output

```
Python 3.7.3 Shell
File  Edit  Shell  Debug  Options  Window  Help
The student details are ...

Student Name and his marks:  AAA BBB, 100
--------------------------------------------------
The Teacher details are ...

Teacher Name and subject taught:  BBB DDD, Mathematics
>>>
                                                Ln: 21  Col: 4
```

Composition

Composition is a relationship in which one object is defined as a part of another object.

For example - Car has engine or Student has marks

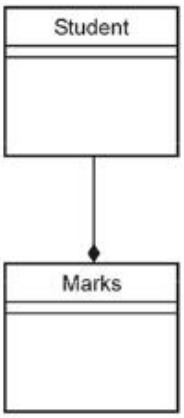
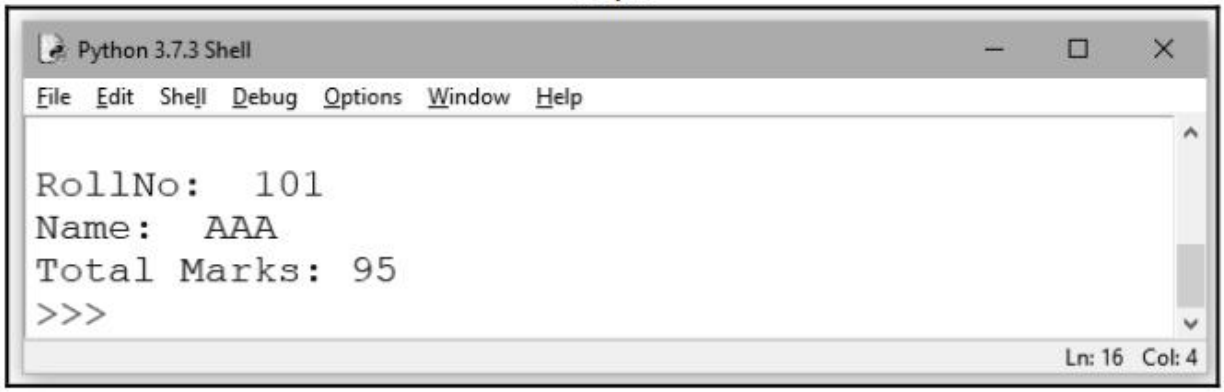


Fig. 5.7.4 Representing Composition

Example Program to implement Compostion

```
class Marks:
   def __init__(self,theory,practical):
      self.theory=theory
      self.practical=practical

   def calculate_marks(self):
      return (self.theory+self.practical)

class Student:
   def __init__(self,rollNo,name,theory,practical):
      self.rollNo=rollNo
      self.name=name
      self.theory=theory
      self.practical=practical
      self.obj_Marks=Marks(self.theory,self.practical) #composition

   def total_marks(self):
      return self.obj_Marks.calculate_marks()

s=Student(101,'AAA',70,25)
print("RollNo: ",s.rollNo)
print("Name: ",s.name)
print("Total Marks:",s.total_marks())
```

Output



4 70%

In composition one of the classes is composed of one or more instance of other classes.

In above code there are two classes Student and Marks.

The Student class is composed of an instance of Marks class. This is done by using following code

```
self.obj_Marks=Marks(self.theory,self.practical)
```

Here, the **Student** class has delegated the responsibility of calculating total marks to the **Marks** class. When one class delegates the responsibilities to other class,it is called **composition.**

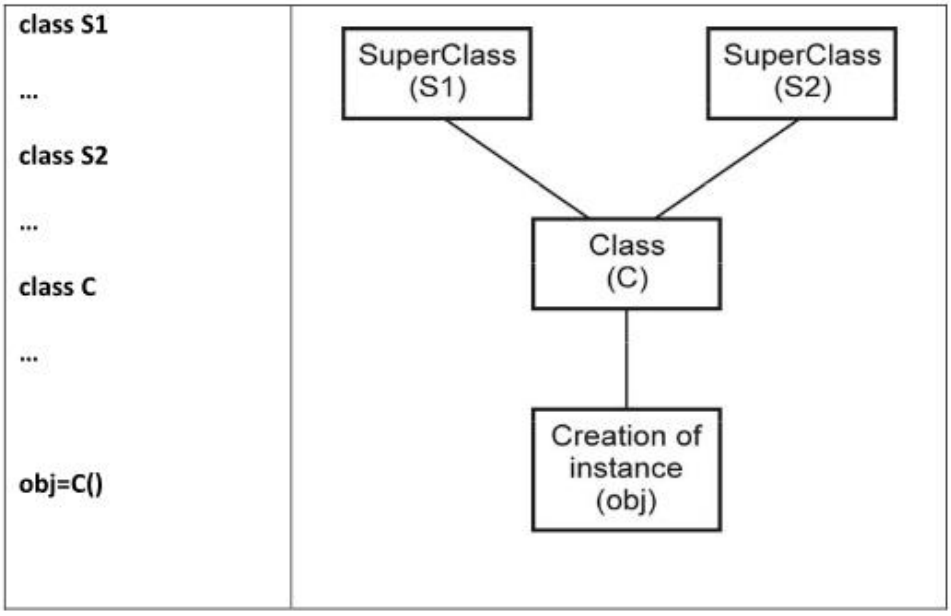
Review Questions

1. *What is inheritance ? Explain it with suitable example.*
2. *What is composition ? Write a Python program to implement the composition relation.*

5.8 Customization via Inheritance

- Inheritance uses attribute definition tree.
- The object.attr searches in upward direction of the tree for first occurrence of the word "attr".
- Lower definitions of the methods in tree redefine or override the upper definitions of the methods in the tree structure of inheritance.

Construction of Attribute Tree



5.8.1 Specializing Inherited Methods

During execution of inheritance tree, the subclass name is located prior to the superclass name.

The direct superclass method calls

```
Class.method(self,...)
```

Example

```
class A:
  def display(self):
    print("Executing in Super Class")
```

```
class B(A):
   def display(self):
      print("Beginning of sub Class")
      A.display(self)
      print("Ending of sub Class")

obj=A()
A.display()
obj=B()
obj.display()
```

Output

Executing in Super Class

Beginning of sub Class

Ending of sub Class

Code Explanation :

Note that when we create an instance of a super class, and call the method using this instance, then the method defined in super class is executing directly, but when we create an instance of sub class, and calling a method using this instance then first the method of super class gets executed.

Use of Specialized Method

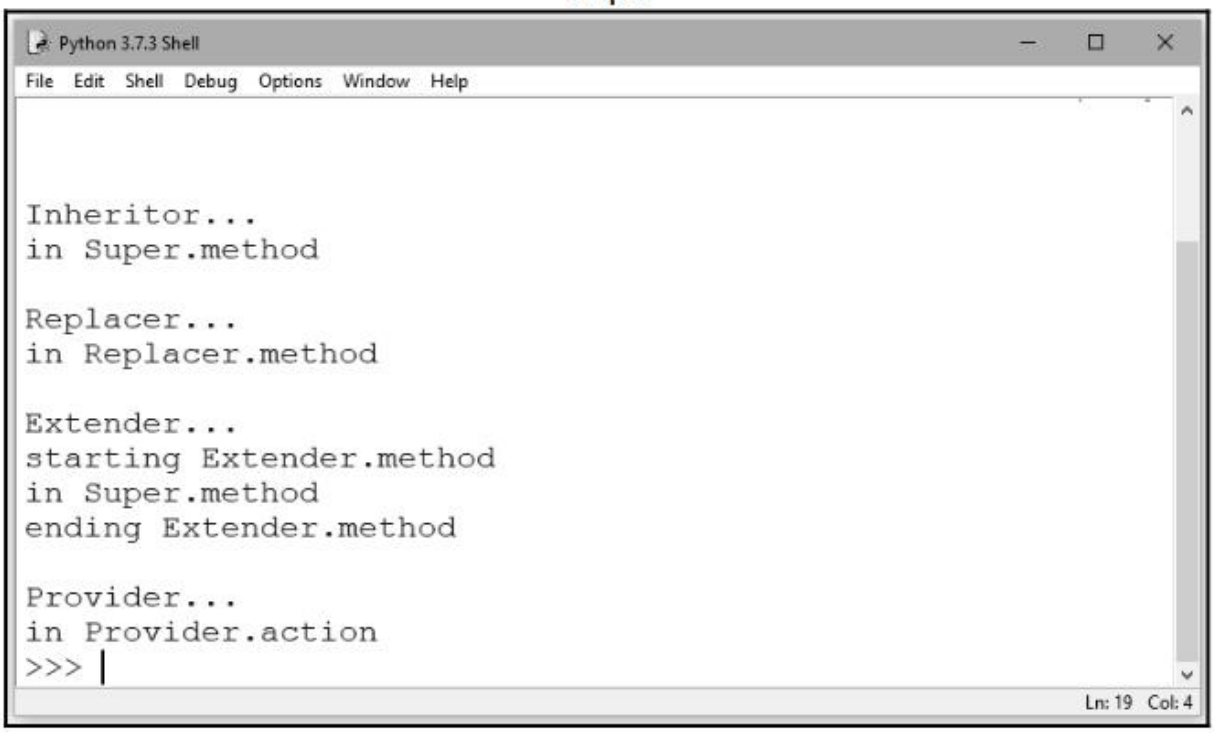
The execution of specializing inherited methods is represented by following example -

```
class Super:
   def method(self):
      print ('in Super.method')   # default
   def delegate(self):
      self.action()               # expected

class Inheritor(Super):
   pass

class Replacer(Super):
   def method(self):
      print ('in Replacer.method')

class Extender(Super):
   def method(self):
      print ('starting Extender.method')
      Super.method(self)
      print ('ending Extender.method')

class Provider(Super):
   def action(self):
```

4 70%


```
        print ('in Provider.action')

if __name__ == '__main__':
    for obj in (Inheritor, Replacer, Extender):
        print ('\n' + obj.__name__ + '...')
        obj().method()
    print ('\nProvider...')
    Provider().delegate()
```

Output



Review Question

1. *Write a short note on – Customization via inheritance.*

❑❑❑

UNIT - VI

6 File I/O Handling and Exception Handling

6.1 I/O Operations

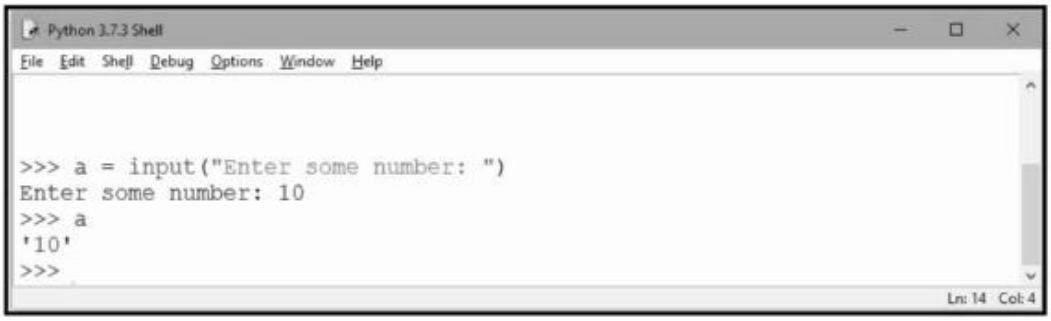
6.1.1 Reading Keyboard Input

- In python it is possible to input the data using keyboard.
- For that purpose, the function **input()** is used.
- **Syntax**

input([prompt])

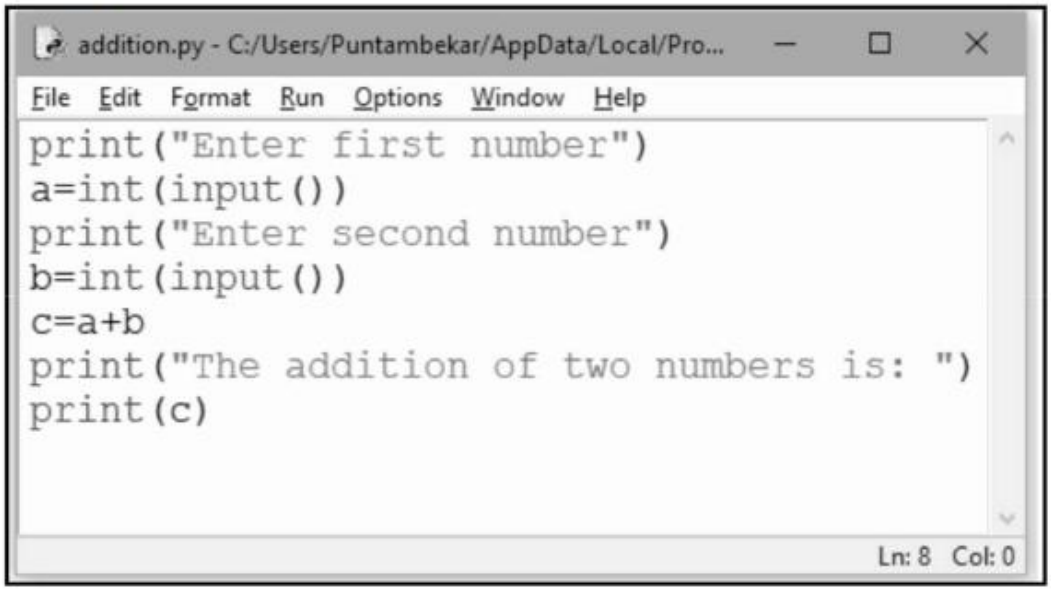
where prompt is the string we wish to display on the screen. It is optional.

Example 6.1.1 : *In the following screenshot, the input method is used to get the data*



Example 6.1.2 : *Write a python program to perform addition of two numbers. Accept the two numbers using keyboard*

Solution : addition.py



Output

For getting the output click on **Run-> Run Module** or press F5 key, following shell window will appear -

```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
addition.py
Enter first number
10
Enter second number
20
The addition of two numbers is:
30
>>>
Ln: 22 Col: 4
```

Program Explanation :

- In above program, we have used **input()** function to get the input through keyboard. But this input will be accepted in the form of string.
- For performing addition of two numbers we need numerical values and not the strings. Hence we use **int()** function to which the **input()** function is passed as parameter. Due to which whatever we accept through keyboard will be converted to integer.
- Finally the addition of two numbers as a result will be displayed.
- The above program is run using F5 and on the shell window the messages for entering first and second numbers will be displayed so that user can enter the numbers.

Example 6.1.3 : *Write a Python program to find the square root of a given number*

Solution:

SqrtDemo.py

```
print("Enter the number:")
num=float(input())
result=num**0.5
print("The sqaure root of ",num," is ",result)
```

Output

```
Enter the number:
25
The sqaure root of  25.0  is  5.0
>>>
```

Example 6.1.4 : *Write a program in Python to obtain principle amount, rate of interest and time from user and compute simple interest.*

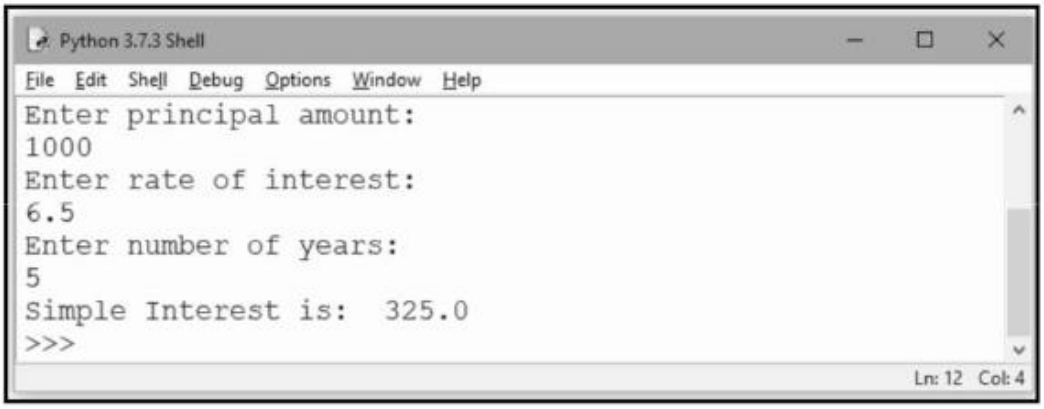
Solution :

Interest.py

```
print("Enter principal amount: ")
p=float(input())
print("Enter rate of interest: ")
r=float(input())
print("Enter number of years: ")
n=float(input())
I=(p*n*r)/100
print("Simple Interest is: ",I)   # Output will be displayed on console
```

Here we are reading the values through keyboard. Note we are reading the values as float

Output



Example 6.1.5 : *To calculate salary of employee given his basic pay(take as input from user). Calculate gross salary of employee. Let HRA be 10 % of basic pay and TA be 5 % of the basic pay. Let employee pay professional tax as 2 % of total salary. Calculate net salary payable after deductions.*

Solution :

```
print("Enter basic salary: ")
basic=float(input())
HRA=(basic*10)/100
TA=(basic *5)/100
gross_salary=basic+HRA+TA
print("Gross Salary is: ",gross_salary)
PTax=(gross_salary*2)/100
net_salary=gross_salary-PTax
print("Net Salary is:",net_salary)
```

Output

```
Enter basic salary:
10000
Gross Salary is:  11500.0
Net Salary is: 11270.0
>>>
```

Example 6.1.6 : *To accept an object mass in kilogram and velocity in meters per second and display its momentum. Momentum is calculated as $e = mc^2$ where m is the mass of the object and c is its velocity.*

Solution :

```
print("Enter mass of the object(kg)")
m=float(input())
print("Enter velocity of the object(m/s)")
c=float(input())
```

```
e=m*(c**2)
print("Momentum is: ",e)
```

Output

```
Enter mass of the object(kg)
10
Enter velocity of the object(m/s)
2.5
Momentum is:  62.5
>>>
```

Example 6.1.7 : *To accept numbers from user and print digits of number in reverse order.*

Solution :

```
print("Enter some number")
n=int(input())
reverse=0
while (n >0):
   reminder = n %10
   reverse = (reverse *10) + reminder
   n = n //10
print("The reversed number is: ",reverse)
```

Output

```
Enter some number
1234
The reversed number is:  4321
>>>
```

Example 6.1.8 : *To input binary number from user and convert it to decimal number*

Solution :

```
print("Enter a binary number: ")
binary = list(input())
value = 0

for i in range(len(binary)):
        digit = binary.pop()#reading each digit of binary number from LSB
        if digit == '1':#if it is 1
                value = value + pow(2, i) #then get 2 ^ i and add it to value
print("The decimal value of the number is", value)
```

Output

```
Enter a binary number:
1100
The decimal value of the number is 12
>>>
```

6.1.2 Printing to Screen

Using the print function we can print the information to the screen. Following are examples that illustrate how to print to the screen

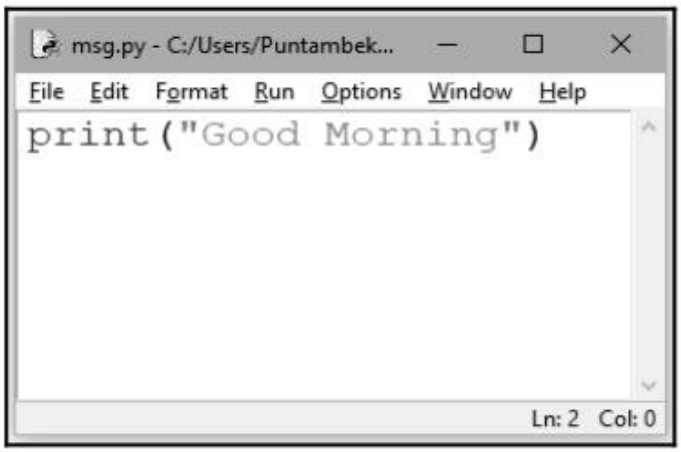
Printing simple text message

Example 6.1.9 : *Display the 'Good Morning' message to screen*

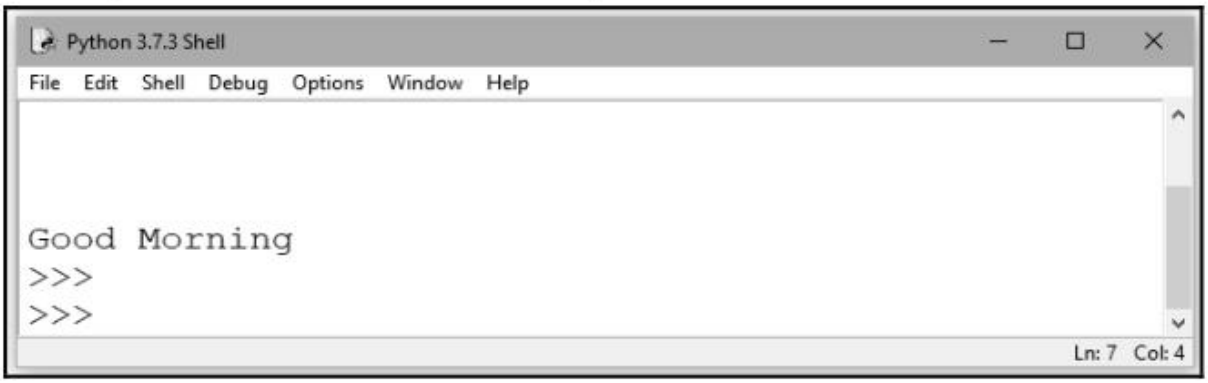
Solution:

Method 1 : You can save the following code in a separate file. I have named the file as msg.py

msg.py

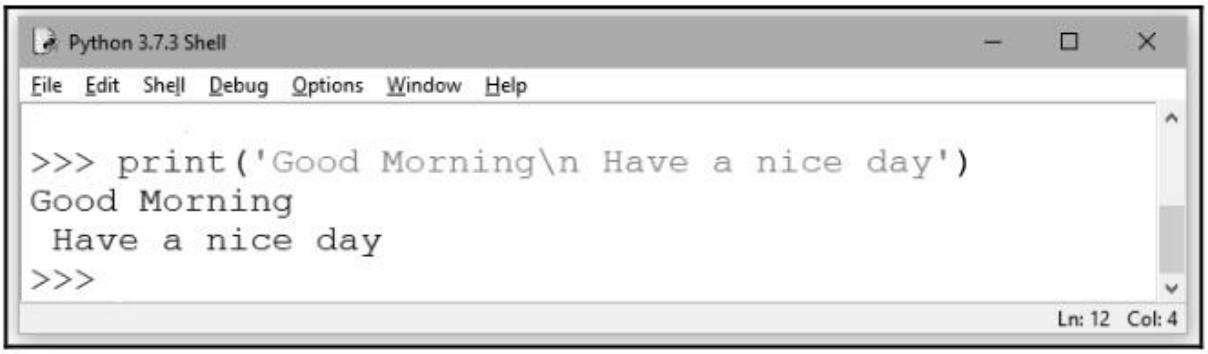


And then press F5 to run the above code – The output will be



Method 2 : Open the Python Shell window and type the print command. It is as shown below -

Printing Simple text message with new line



Printing Variables

```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>> a=10
>>> print(a)
10
>>>
Ln: 15 Col: 4
```

We can display the value of variable along with the message

```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>> x=100
>>> print("Value of x= ",x)
Value of x=  100
>>>
Ln: 18 Col: 4
```

Printing variables using format

```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>> a=100
>>> b=200
>>> print("a = {}\n b = {}".format(a,b))
a = 100
 b = 200
>>>
Ln: 23 Col: 4
```

Review Questions

1. *Write a Python program to read some input using keyboard*
2. *What are different ways of printing data to screen ? Explain it with illustrative examples.*

Printing Variables

```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>> a=10
>>> print(a)
10
>>>
Ln: 15 Col: 4
```

We can display the value of variable along with the message

```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>> x=100
>>> print("Value of x= ",x)
Value of x=  100
>>>
Ln: 18 Col: 4
```

Printing variables using format

```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>> a=100
>>> b=200
>>> print("a = {}\n b = {}".format(a,b))
a = 100
 b = 200
>>>
Ln: 23 Col: 4
```

Review Questions

1. *Write a Python program to read some input using keyboard*
2. *What are different ways of printing data to screen ? Explain it with illustrative examples.*

6.2 File Handling

Definition : File is a named location on the disk to store information.

- File is basically used to store the information permanently on secondary memory.
- Due to this, even if the computer is switched off, the data remains permanently on secondary memory (hard disk). Hence it is called **persistent data structure.**

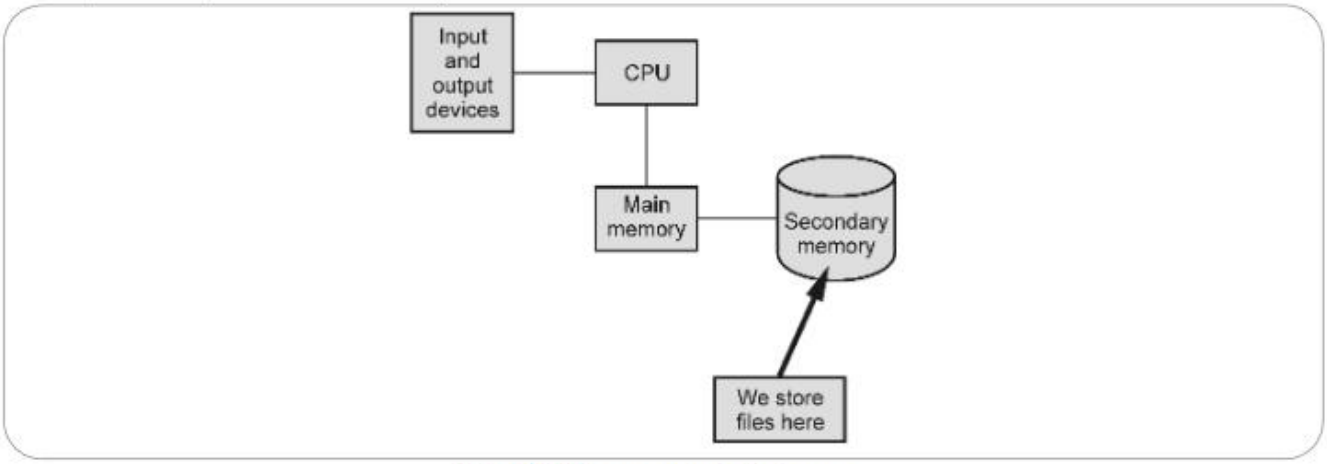


Fig. 6.2.1 File as persistent storage

- Every file is located by its path. This path begins at the root folder. In windows it can C:\, D:\ or E;\ etc.
- The file path is also called as **pathname.**

  For example c:\MyPythonPrograms\displayStrings.py is a pathname. In this path name C:\ is a root folder in which the sub-folder **MyPythonProgram** has a file called **displayStrings.py**
- The character \ used in the pathname is called **delimiter**.
- **Concept of Relative and Absolute Path :**
  - The absolute path is a complete path from the root directory. For example – the path **C: \MyPythonPrograms\Strings\displayStrings.py** is an absolute path using which the file can be located.
  - The relative path is a path towards the file from the current working directory. For example - **\Strings\displayStrings.py** can be a relative path if you are currently in the working directory C:\MyPythonPrograms

6.2.1 Opening a File in Different Mode

Opening a file

- In python there is a built in function open() to open a file.

Syntax

```
File_object=open(file_name,mode)
```

Where File_object is used as a handle to the file.

Example

```
F=open("test.txt")
```

Here file named **test.txt** is opened and the file object is in variable **F.**

We can open the file in text mode or in binary mode. There are various modes of a file in which it is opened. These are enlisted in the following table.

Mode	Purpose
'r'	Open file for reading
'w'	Open file for writing. If the file is not created, then create new file and then write. If file is already existing then truncate it.
'x'	Open a file for creation only. If the file already exists, the operation fails.
'a'	Open the file for appending mode. Append mode is a mode in which the data is inserted at the end of existing text. A new file is created if it does not exist.
't'	Opens the file in text mode
'b'	Opens the file in binary mode
'+'	Opens a file for updation i.e. reading and writing.

For example :

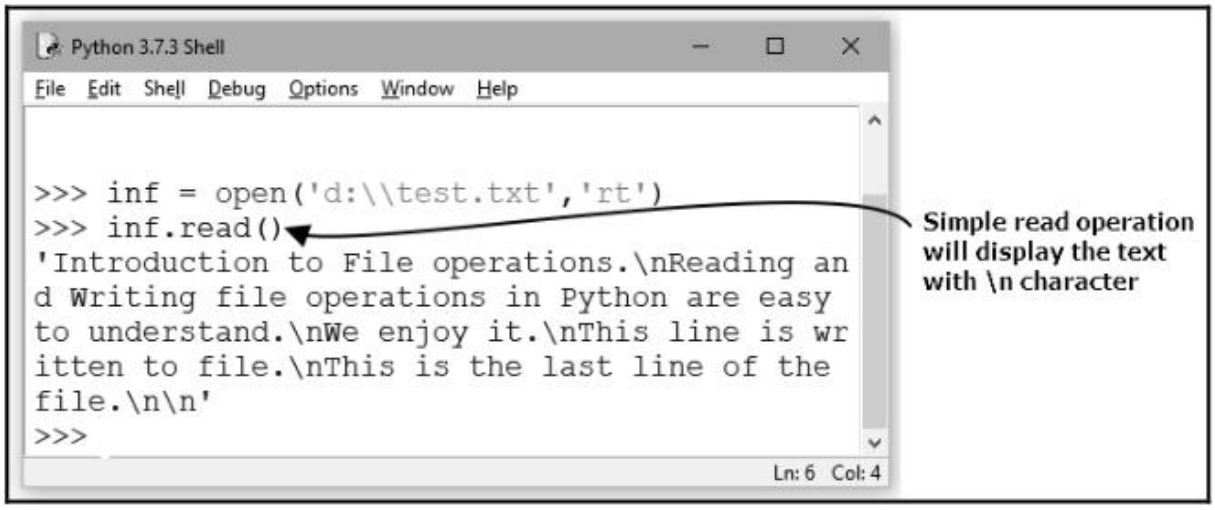
```
fo=open("test.txt",w) #opens a file in write mode
fo=open("text.txt",rt) #Open a file for reading in text mode
```

6.2.2 Accessing File Contents using Standard Library Functions

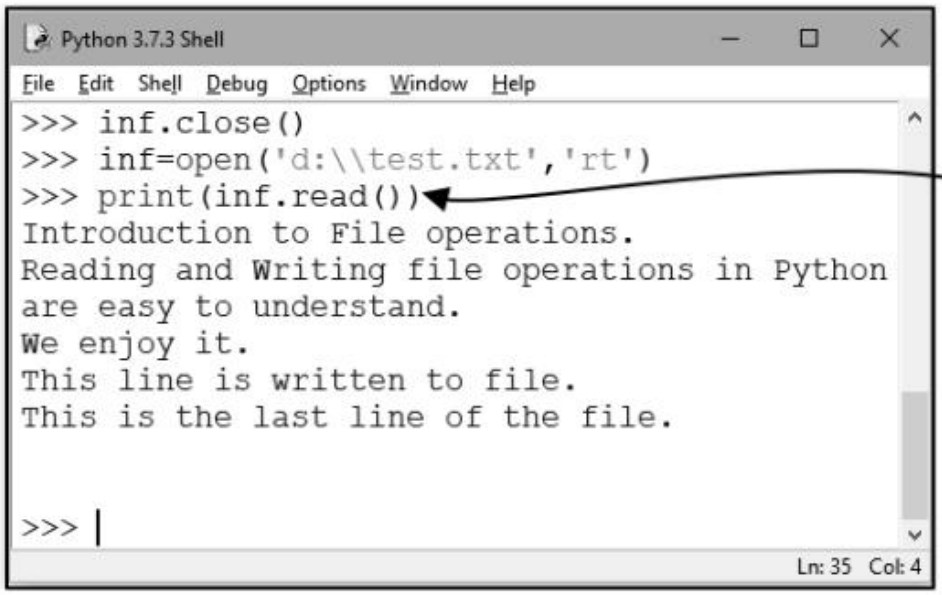
6.2.2.1 Reading Files

For reading the file , we must open the file in r mode and we must specify the correct file name which is to be read

The read() method is used for reading the file. For example –



Let us close this file and reopen it. Then call read statement inside the print statement, illustrated as follows –



If we write read() statement inside the print statement. Then the contents will be displayed in the same manner as they are stored inside the file

Example 6.2.1 : *Write a Python program to read the contents of the file named 'test.txt*
Solution :

ReadFile.py

```
inf = open('D:\\test.txt','rt')
print(inf.read())
inf.close()
```

Output

```
Introduction to File operations.
Reading and Writing file operations in Python are easy to understand.
We enjoy it.
This line is written to file.
This is the last line of the file.
>>>
```

Note that a blank line is returned when the file reaches the end of the file.

There some other useful methods for reading the contents of the file. Let us discuss them with the help of necessary illustrations

The readline() Method

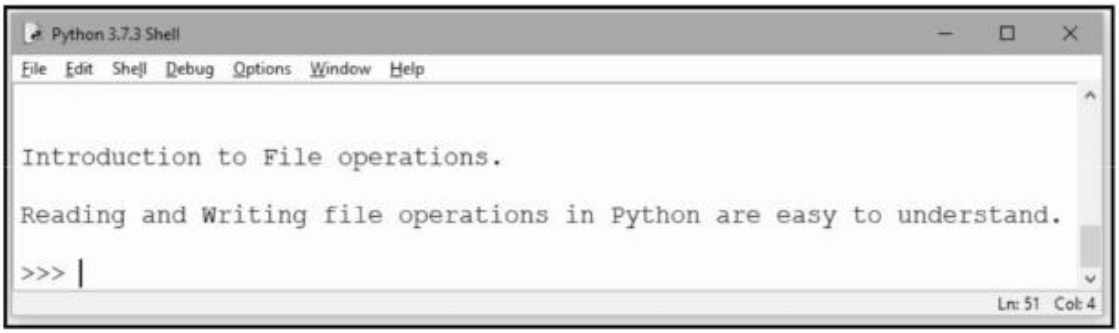
The readline() method allows us to read a single line from the file. When file reaches to the end, it returns an empty string.

Example 6.2.2 : *Write a Python program to read and display first two lines of the text file.*
Solution :

ReadLineDemo.py

```
inf = open('D:\\test.txt','rt')
print(inf.readline())
```

```
print(inf.readline())
inf.close()
```



Program Explanation : In above program,

1) The file is opened using **open** statement.
2) The we call **readline()** statements inside two subsequent print statement. After reading from the file using the **readline()** method, the control automatically passes to the next line. Hence we call **readline()** inside the **print** statement again.
3) Finally we must not forget to close the file using **close()** method.

The readLines() Method

The readLines() method is used to print all the lines in the program. Following program illustrates it –

ReadLinesDemo.py

```
inf = open('D:\\test.txt','rt')
print(inf.readlines())
inf.close()
```

Output

```
Introduction to File operations.

Reading and Writing file operations in Python are easy to understand.

We enjoy it.

This line is written to file.

This is the last line of the file.

>>>
```

The list() Method

The list method is also used to display the contents of the file as a list. The program is as follows –

ListDemo.py

```
inf = open('D:\\test.txt','rt')
print(list(inf))
inf.close()
```

34 70%

Output

```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help

['Introduction to File operations.\n', 'Reading and Writ
ing file operations in Python are easy to understand.\n'
, 'We enjoy it.\n', 'This line is written to file.\n', '
This is the last line of the file.\n', '\n']
>>>
Ln: 54 Col: 4
```

Note that we passed the file object as an argument to the list method.

Displaying File using Loop

This is the most commonly used method of reading the file. In this method the contents of the file are read line by line using for loop.

Example 6.2.3 : *Write a program to display the contents of the file using for loop.*

Solution :

DisplayFile.py

```
inf = open('D:\\test.txt','rt')
for line in inf:
   print(line)
inf.close()
```

Output

```
Introduction to File operations.

Reading and Writing file operations in Python are easy to understand.

We enjoy it.

This line is written to file.

This is the last line of the file.

>>>
```

Opening a file using with

We can open the file using keyword **with**. The advantage of this is that the file gets closed properly after the read or write operations

OpenWithDemo.py

```
with open('D:\\test.txt','rt') as inf:
   for line in inf:
      print(line)
inf.close()
```

Output

```
Introduction to File operations.
Reading and Writing file operations in Python are easy to understand.
We enjoy it.
This line is written to file.
This is the last line of the file.
>>>
```

Example 6.2.4 : *Write a Python program to find the line that starts with the word "This" from the following text which is stored in a file*

Test.txt

This is a python program

python is superb.

This is third line of program

this python program is nice

Solution :

FileDemo1.py

```
fh=open('d:\\test.txt')
i=0
for line in fh:
   line=line.rstrip()
   if line.find('This')==-1:continue
   print(line)
```

Output

Example 6.2.5 : *Write a program in Python to split the text line written in the file into words.*

Solution :

```
with open('d:\\test.txt','rt') as inf:
   line=inf.readline()
   word_list=line.split()
   print(word_list)
```

Output

```
['This', 'is', 'a', 'python', 'program']
>>>
```

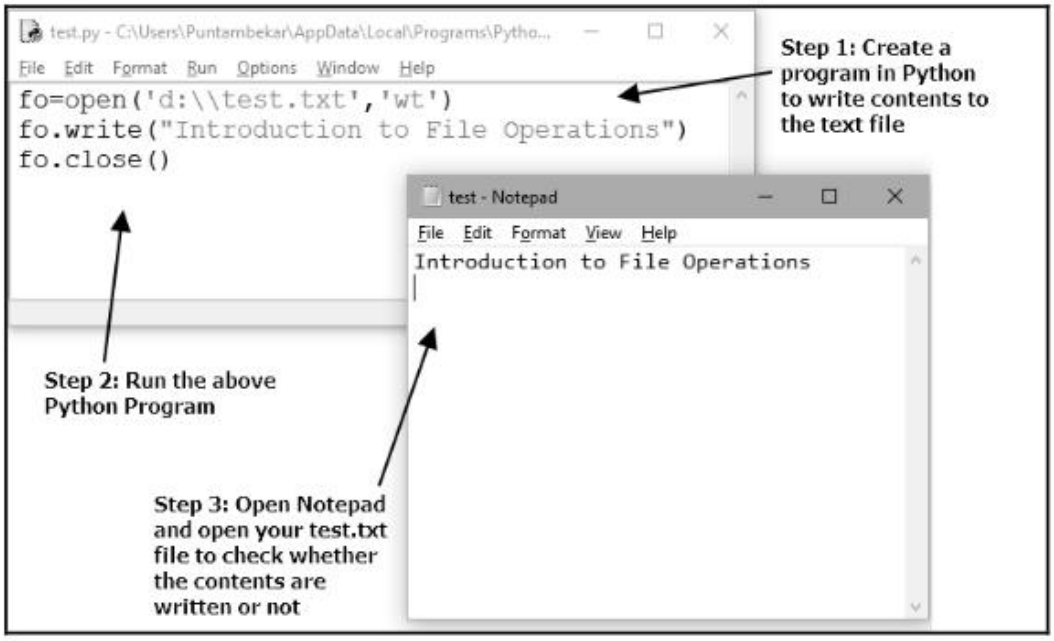
6.2.2.2 Writing Files

- For writing the contents to the file, we must open it in writing mode. Hence we use 'w' or 'a' or 'x' as a file mode.
- The write method is used to write the contents to the file.

Syntax

File_object.write(contents)

- The write method returns number of bytes written.
- While using the 'w' mode in open, just be careful otherwise it will overwrite the already written contents. Hence in the next subsequent write commands we use "\n" so that the contents are written on the next lines in the file.
- For example :



The writelines() method

The writelines() method is used to write list of strings to the file. Following program illustrates this

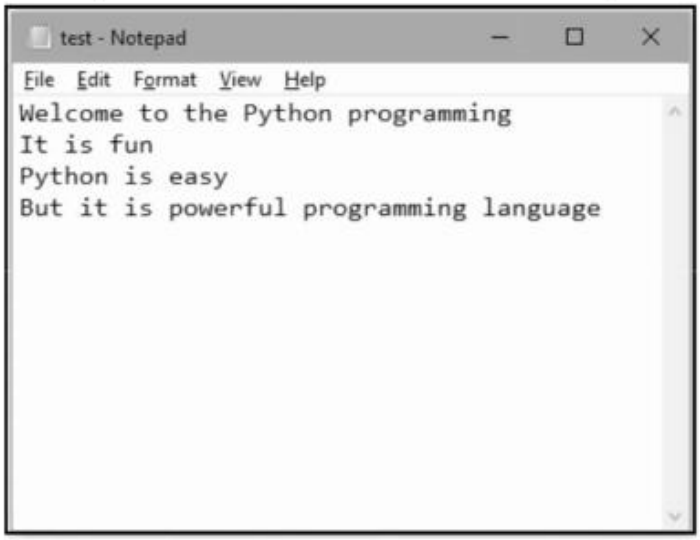
Example 6.2.6 : *Write a python program to write multiple lines to a text file using writelines() method*

Solution :

```
fo=open('d:\\test.txt','wt')
lines=["Welcome to the Python programming\n","It is fun\n",
    "Python is easy\n", "But it is powerful programming language"]
fo.writelines(lines)
fo.close()
```

Output

Now open the Notepad and open the test.txt file in it. It will be something like this -



Appending the file

Appending the file means inserting records at the end of the file.

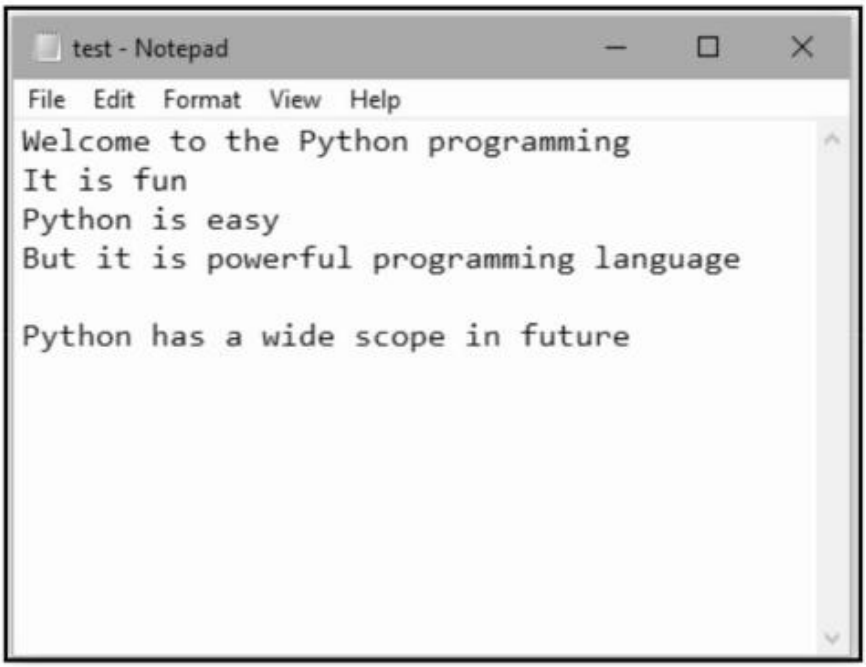
For appending the file we must open the file in 'a' or 'ab' mode.

For example

```
fo=open('d:\\test.txt','a')
fo.write("Python has a wide scope in future")
fo.close()
```

Output

Just open the existing d:\test.txt file to check the contents. It will be as follows –

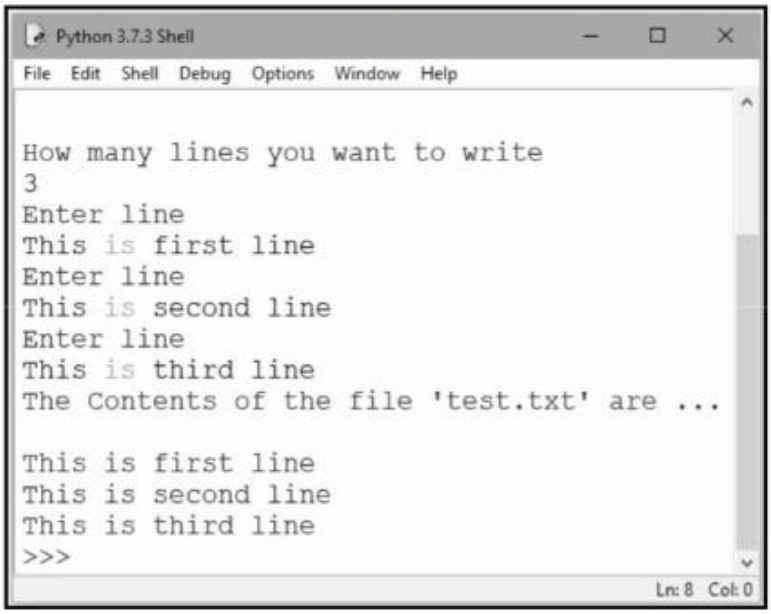


Example 6.2.7 : *Write a python program to write n number of lines to the file and display all these lines as output.*

Solution :

```
print("How many lines you want to write")
n=int(input())
outFile=open("d:\\test.txt",'wt')
for i in range(n):
   print("Enter line")
   line=input()
   outFile.write("\n"+line)
outFile.close()
print("The Contents of the file 'test.txt' are ...")
inFile=open("d:\\test.txt",'rt')
print(inFile.read())
inFile.close()
```

Output



Example 6.2.8 : *Write a python program to write the contents in 'one.txt' file. Read these contents and write them to another file named 'two.txt'*

Solution :

```
print("How many lines you want to write")
n=int(input())
outFile=open("d:\\one.txt",'wt')
for i in range(n):
   print("Enter line")
   line=input()
   outFile.write("\n"+line)
outFile.close()
inFile=open("d:\\one.txt",'rt')
outFile=open("d:\\two.txt",'wt')
i=0
for i in range(n+1):
```

34 70%


```
    line=inFile.readline()
    outFile.write("\n"+line)
inFile.close()
outFile.close()
print("The Contents of the file 'two.txt' are ...")
inFile=open("d:\\two.txt",'rt')
print(inFile.read())
inFile.close()
```

6.2.3 Closing Files

After performing all the file operations, it is necessary to close the file.

Closing of file is necessary because it frees all the resources that are associated with file.

Example -

```
fo=open("test.txt",rt)
fo.close()  #closing a file.
```

6.2.4 Renaming and Deleting Files

(i) Renaming File

For renaming a file in Python, the **rename()** method is used. For that purpose, it is required to import the module **os**.

Syntax :

```
os.rename(src, dst)
```

Where

src: The file name with its complete path which is to be renamed.

dst: The new name for the file.

renameDemo.py

```
import os
os.rename(r'd:\test1.txt',r'D:\test2.txt')
print("File is renamed")
```

Output

```
File is renamed
```

To check if the file name is changed or not, we must find the file at corresponding location, and check the contents by opening it.

(ii) Deleting File

For removing the file, we use **remove** method belonging to the **os** module.

Syntax :

```
os.rename(filename)
```

The code is as follows

```
import os
os.rename('d:\test1.txt')
print("File is removed")
```

Output

```
File is removed
```

6.2.5 Directories in Python

Finding File Position

The seek and tell method : The **seek** method is used to change the file position. Similarly the **tell** method returns the current position.

The method seek() sets the file's current position at the offset.

Syntax

fileObject.seek(offset[, whence])

Where

- offset – This is the position of the read/write pointer within the file.
- whence – This is optional and defaults to 0 which means absolute file positioning, other values are 1 which means seek relative to the current position and 2 means seek relative to the file's end.

The method tell() returns the current position of the file read/write pointer within the file.

Syntax

fileObject.tell()

Programming Example

```
inf=open('d:\\test.txt','rt')
print("\tThe contents of the file are...")
print(inf.read())
print("\tThe current position is...")
print(inf.tell())#get current postion of file
inf.seek(0) #moves the file cursor to initial position
print("\tThe current position is...")
print(inf.tell())#get current position of file
```

Output

```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
        The contents of the file are...
Introduction to File operations.
Reading and Writing file operations in Python are easy to understand.
We enjoy it.
This line is written to file.
This is the last line of the file.

        The current position is...
186
        The current position is...
0
>>>
Ln: 63 Col: 4
```

We can pass number of bytes to the seek method. These many bytes are skipped and then remaining contents are displayed. For example

```
>>> inf.seek(10)
10
>>> print(inf.read())
  #note that 10 bytes are skipped and then the remaining contents are displayed
on to File Operations
 Reading and writing operations in Python are easy
 We enjoy it
 This line is written in file
 This is a last line of this file
>>>
```

Directory Methods

In this section we will discuss various directory methods.

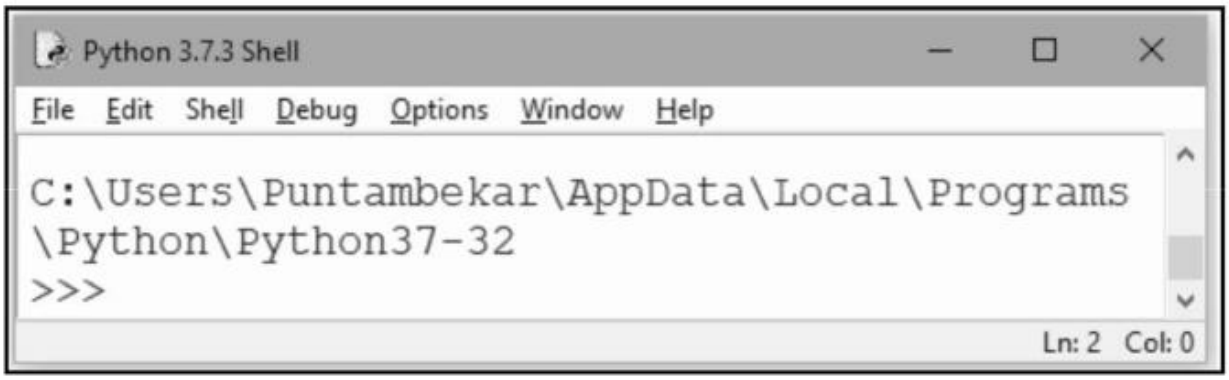
1) Getting current working directory

For getting the name of current working directory we use the function named **getcwd()** method. This method returns the name of current working directory. For example –

getwdDemo.py

```
import os
print(os.getcwd())
```

Output



2) Displaying all the files and sub-directories inside a directory

For listing the contents of a directory we use the method **listdir().** For example –

```
import os
print(os.listdir())
```

3) Creating a new Directory

We can create a new directory using mkdir() method

mkdirDemo.py

```
import os
os.mkdir("mypythonPrograms") #creates a folder named mypythonPrograms
print(os.listdir()) #displaying the directory contents
```

4) Removing the directory or a file

A file can be deleted using remove() method.

The rmdir() method removes an empty directory.

For example

```
import os
os.remove("test.txt")#removes file test.txt
print(os.listdir()) #displays the directory contents
os.rmdir("mypythonPrograms") #removes the directory named 'mypythonPrograms'
print(os.listdir())#displays the directory contents
```

6.2.6 Programs on File Handling

Example 6.2.9 : *Write a program in Python to count number of vowels and consonants present in the text file*

Solution :

```
infile = open("d:\\test.txt", "r")
vowels = set("AEIOUaeiou")
cons = set("bcdfghjklmnpqrstvwxyzBCDFGHJKLMNPQRSTVWXYZ")
count_Vowels = 0
count_Cons = 0
for c in infile.read():
   if c in vowels:
      count_Vowels += 1
   elif c in cons:
      count_Cons += 1
print("Number of vowels in a file are: ",count_Vowels)
print("Number of consonants in a file are: ",count_Cons)
infile.close()
```

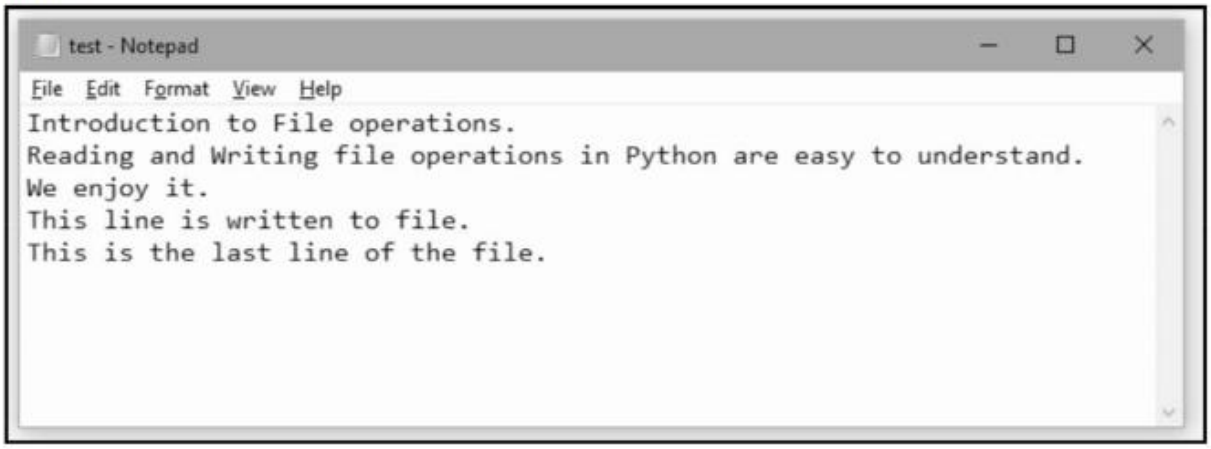
Output

```
Number of vowels in a file are:  59
Number of consonants in a file are:  85
>>>
```

Example 6.2.10 : *Write a Python program to count total number of lines in a file*

Solution :

Step 1 : Create a text file named test.txt as follows



Step 2 : Now write the python code to count the total number of lines in a file

```
count=0
with open("D:\\test.txt",'r') as f:
   for line in f:
      word=line.split()
```

```
        count+=1
f.close()
print("Total number of lines in file are: ",count)
```

Output

```
Total number of lines in file are:  5
>>>
```

Example 6.2.11 : *Write a Python program to count total number of words in a file*

Solution :

Step 1 : Create a text file names test.txt as follows –

Introduction to File operations.

Reading and Writing file operations in Python are easy to understand.

We enjoy it.

This line is written to file.

This is the last line of the file.

Step 2 : Now write a Python code to count number of words in the file. The code is as follows –

```
count=0
with open("D:\\test.txt",'r') as f:
   for line in f:
      word=line.split()
      count+=len(word)
f.close()
print("Total number of words in file are: ",count)
```

Output

```
Total number of words in file are:  32
>>>
```

Example 6.2.12 : *Write a Python program to count total number of characters in a file*

Solution :

Step 1 : Create a text file names test.txt as follows –

Introduction to File operations.

Reading and Writing file operations in Python are easy to understand.

We enjoy it.

This line is written to file.

This is the last line of the file.

Step 2 : Now write a Python code to count number of characters in the file. The code is as follows –

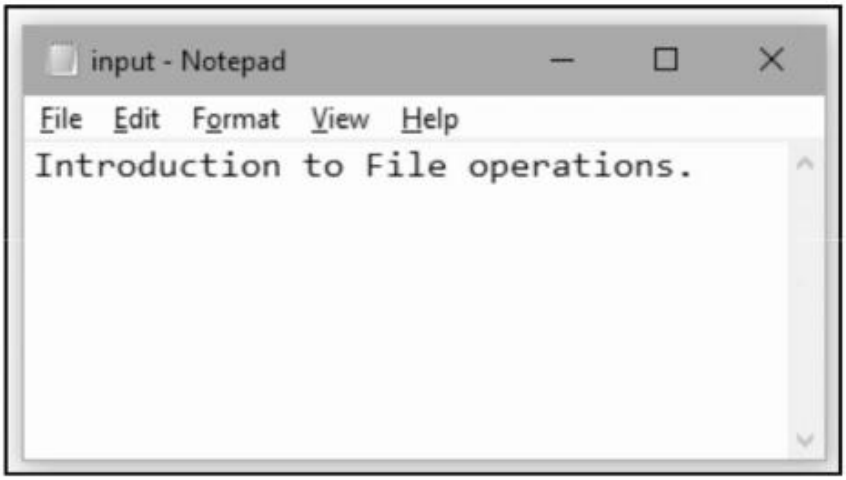
```
count=0
with open("D:\\test.txt",'r') as f:
   for line in f:
      word=line.split()
      count+=len(line)
f.close()
print("Total number of characters in file are: ",count)
```

Output

```
Total number of characters in file are:  181
>>>
```

Example 6.2.13 : *Write a Python program to count the total number of blank spaces in the file*

Solution : First of all create a text file and write some contents to it. Your contents must have some blank characters. Following is input.txt file



Then write the python code to count the number of blanks. Note that it will count newline character as a blank character as well.

```
with open("D:\\input.txt",'r') as f:
   count=0
   for line in f:
      word = line.split()#getting each word of every line
      for letter in word: #getting each letter from each word
            if(letter.isspace):#checking if it is blank
               count=count+1
print("Total Number of blank spaces are: ",count)
```

Output

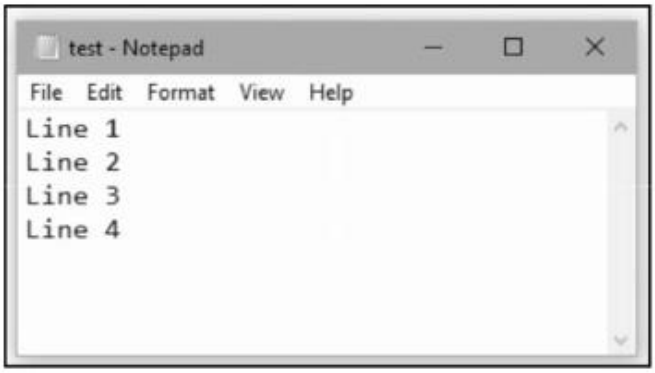
```
Total Number of blank spaces are:  4
>>>
```

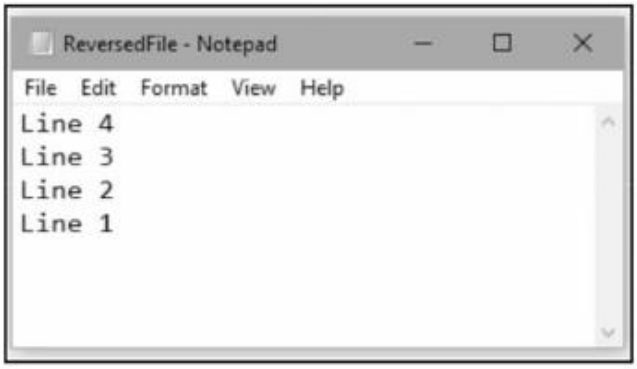
Example 6.2.14 : *Write a Python program to write the contents of one file to another file from end.*

Solution :

```
lines = []
with open('d:\\test.txt') as f:
   lines = f.readlines()
f.close()
with open('d:\\ReversedFile.txt', 'w') as f:
   for line in reversed(lines):
      f.write(line)
f.close()
```

Output





Review Questions

1. *What is file ? Explain the concept of Relative and Absolute Path.*
2. *Explain the file open method.*
3. *Explain different functions used with respect to file positions*
4. *What is file ? Explain how to perform file read and write operations in Python ?*

6.3 Exception Handling

- **Definition of exception :** An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program.
- In general, when a python script encounters a situation that it cannot cope with, it raises an exception.
- When a python script raises an exception, it must either handle the exception immediately otherwise it terminates and quits.

6.3.1 try:Except Statement

- The exception handling mechanism using the **try…except…else** blocks.
- The suspicious code is placed in try block.
- After try block place the except block which handles the exception elegantly.
- If there is no exception then the else block statements get executed.
- **Syntax of try…except…else**

```
try:
                    write the suspicious code here
except Exception 1:
                    If Exception 1 occurs then execute this code
except Exception 2:
                    If Exception 2 occurs then execute this code
...
...
else:
                    If there is no exception then execute this code.
```

- Example : python program to perform division of two numbers.

```
try :
  a=int(input("Enter value of a: "))
  b=int(input("Enter value of b: "))
```

```
    c=a/b
except ValueError:
    print("You have entered wrong data")
except ZeroDivisionError:
    print("Divide by Zero Error!!!")
else:
    print("The result: ",c)
```

6.3.2 The Raise Statement

- Raising an exception is a technique for interrupting the normal flow of execution in a program, signaling that some exceptional circumstance has arisen, and returning directly to an enclosing part of the program that was designated to react to that circumstance.
- The Python interpreter raises an exception each time it detects an error in an expression or statement.
- However, Users can also raise exceptions with raise statement.
- **Syntax**

```
Raise [Exception [, args [,traceback]]]
```

Here,

Exception is a type of exception (For example NameError)
args is value for exception argument.
traceback is object of traceback

- **Example**

```
def NegativeNum( num ):
  if num < 0:
    raise "Number is negative!!", num
```

6.3.3 Standard Built in Exceptions

Python provide standard built in Exceptions. These exceptions are enlisted in the following table.

Name	Purpose
Exception	Base class for all exceptions
ArithmeticError	Base class for all errors that occur for numeric calculation.
OverflowError	Raised when a calculation exceeds maximum limit for a numeric type.
FloatingPointError	Raised when a floating point calculation fails.
ZeroDivisionError	Raised when division or modulo by zero takes place for all numeric types.
EOFError	Raised when there is no input from either the raw_input() or input() function and the end of file is reached.
ImportError	Raised when an import statement fails.
KeyboardInterrupt	Raised when the user interrupts program execution, usually by pressing Ctrl+c.

NameError	Raised when an identifier is not found in the local or global namespace.
IOError	Raised when an input/ output operation fails.
SystemError	Raised when the interpreter finds an internal problem, but when this error is encountered the python interpreter does not exit.
SystemExit	Raised when python interpreter is quit by using the sys.exit() function. If not handled in the code, causes the interpreter to exit.
TypeError	Raised when an operation or function is attempted that is invalid for the specified data type.
ValueError	Raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified.
RuntimeError	Raised when a generated error does not fall into any category.

6.3.4 User Defined Exception

- Although standard exceptions in Python provide adequate functionality, It is advantageous to create user defined exception.
- With user defined exception, one can add the desired information to the exception handlers.
- **IOError** is a generic exception used for input/output problems, which may arise from invalid file accessor other forms of communication.
- Similarly, the user defined exception class can be derived from **Exception** class.
- **For example :** Following program makes use of user defined exception for accepting the positive value only. If user enters negative or zero value, then exception is raised.

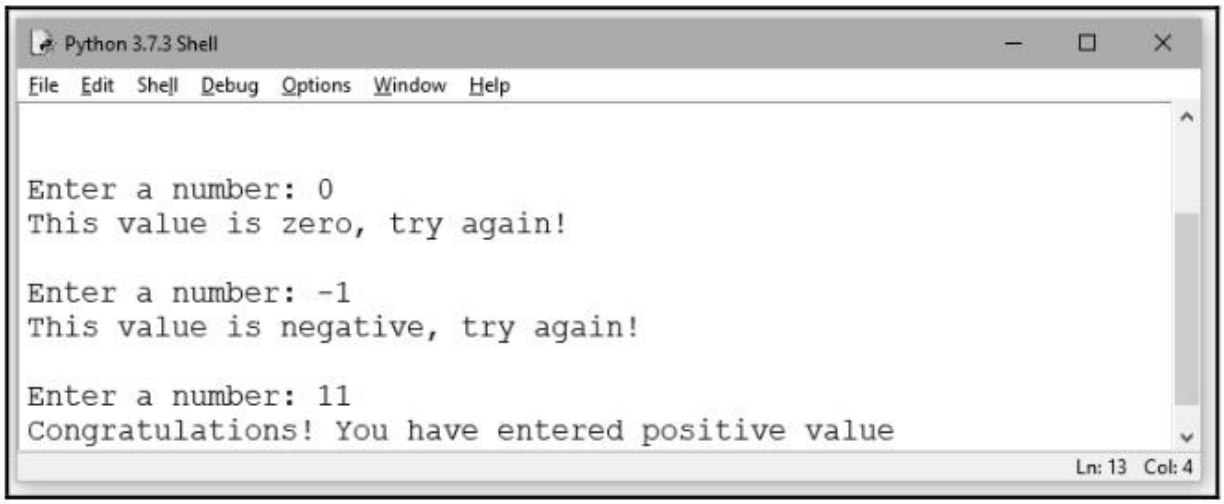
usrEx.py

```
class Error(Exception):
  """Base class for other exceptions"""
  pass
class NegativeValue(Error):
  """Raised when the input value is negative"""
  pass
class ZeroValue(Error):
  """Raised when the input value is zero"""
  pass
# our main program
while True:
  try:
    i_num = int(input("Enter a number: "))
    if i_num < 0:
      raise NegativeValue
    elif i_num == 0:
      raise ZeroValue
    break
  except NegativeValue:
```

4 70%


```
        print("This value is negative, try again!")
        print()
    except ZeroValue:
        print("This value is zero, try again!")
        print()
print("Congratulations! You have entered positive value")
```

Output



Example 6.3.1 : *Write a Python program to raise divide by zero error using user defined exception*

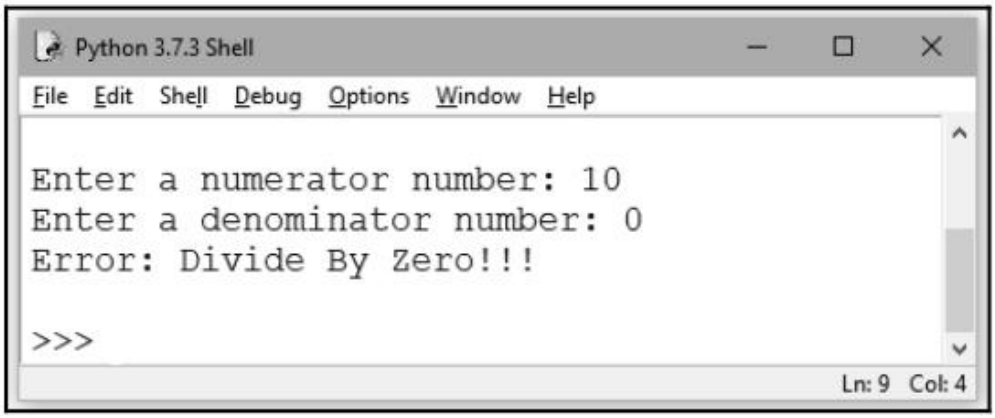
Solution:

UsrdefEx1.py

```
# define Python user-defined exceptions
class Error(Exception):
  """Base class for other exceptions"""
  pass
class DivideByZero(Error):
  """Raised when the denominator value is zero"""
  pass
# our main program
try:
  a = int(input("Enter a numerator number: "))
  b = int(input("Enter a denominator number: "))
  if b == 0:
    raise DivideByZero
  c=a/b
  print("The result = ",c)
except DivideByZero:
  print("Error: Divide By Zero!!!")
  print()
```

4 70%

Output



Review Questions

1. *What is exception ?*
2. *Explain exception handling mechanism in Python.*
3. *Explain the use of raise statement in Python with suitable example.*
4. *Explain how to create and use the user defined exception.*

❑❑❑