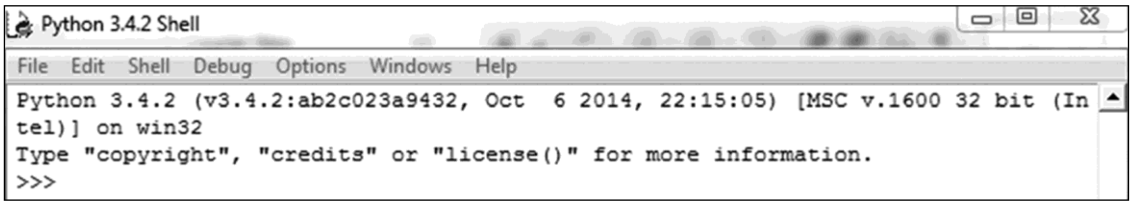
Python Assignment No. 6

Q1. Explain Turtle module in details.

Ans.

To start, a programmer can use interactive mode (command line) or script mode of Python. The steps required to start graphics programming using the Turtle module in interactive mode of Python are given as follows:

- STEP 1: Launch Python by pressing the start button in Windows and writing Python in the search box. Click on Python IDLE to start the interactive mode. The following windowwill then appear (Figure 12.1).



```
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct  6 2014, 22:15:05) [MSC v.1600 32 bit (In
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

Figure 12.1

- STEP 2: At the Python's statement prompt >>> type the following command to import theTurtle module.

```
>>> import Turtle                    #import Turtle module
```

- STEP 3: Type the following command to show the current location and direction of the Turtle.

```
>>> Turtle.showTurtle()
```

After the execution of the above statement, Python's Turtle graphics window will be displayedas shown in Figure 12.2.

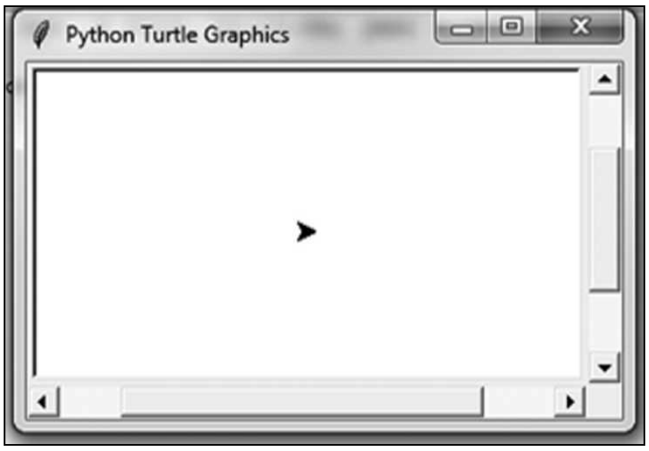


Figure 12.2 Python's Turtle Graphics Window

The Turtle is like a pen. The arrowhead indicates the current position and direction of the pen.

Initially, the Turtle is positioned at the center of the window.

Q2. Explain moving turtle in any direction.

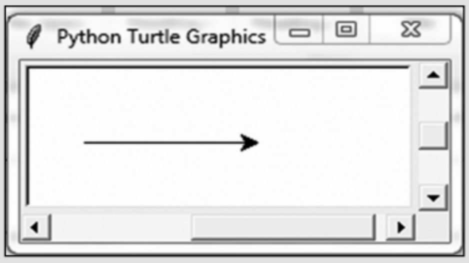
Ans.

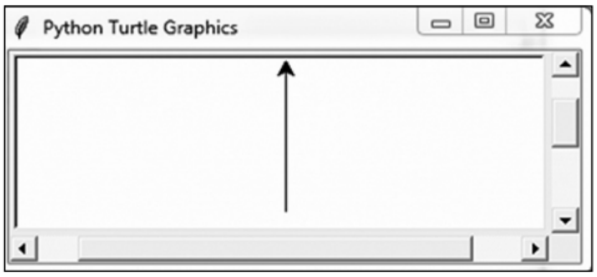
The Turtle is an object which is created when we import the Turtle module. As soon as the object is created its position is set at (0, 0), i.e. at the center of the Turtle graphics window. Also by default its direction is set to go straight to the right.

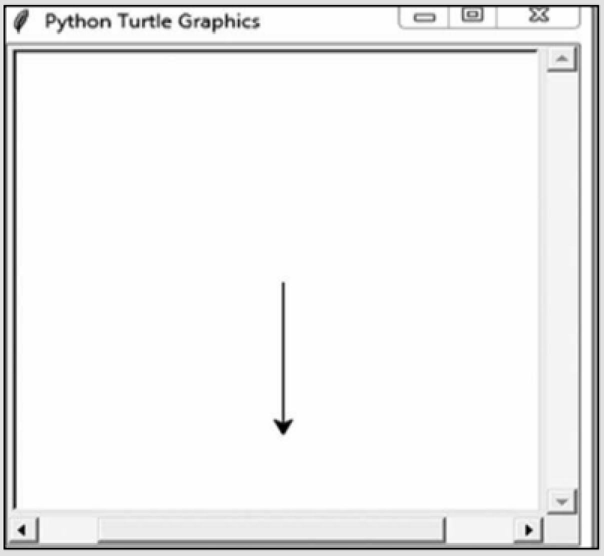
The imported Turtle module uses a pen to draw shapes. It can be used to move and draw lines in any direction. Python contains methods for moving the pen, setting the pen's size, lifting and putting the pen down. By default, the pen is down, i.e. it draws a line from the current position to the new position. Table 12.1 shows a list of methods to move the Turtle in specified directions.

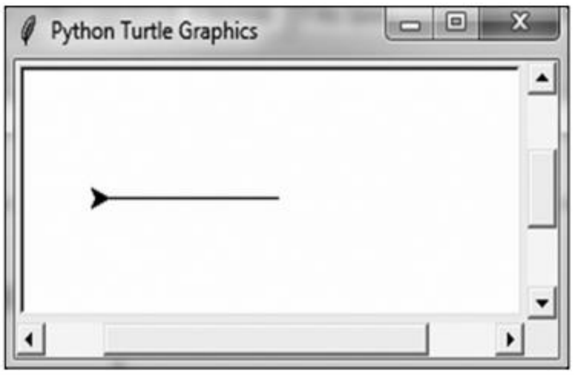
Table 12.1 Turtle methods related to directions

Method	*Meaning*

`Turtle.forward(P)`
Example:
`>>> import Turtle`
`>>> Turtle.forward(100)`
	Moves the Turtle P pixels in the direction of its current heading.
	Output:
	

`Turtle.left(angle)`
Example:
`>>> import Turtle`
`>>> Turtle.left(90)`
`>>> Turtle.forward(100)`
	Rotates the Turtle left by the specified angle.
	Output:
	
	Explanation: Initially the Turtle is placed at the centre by default. The command **Turtle.left(90)** changes the direction of the Turtle to the left by 90 degrees. Finally, the arrowhead moves 100 pixels forward.

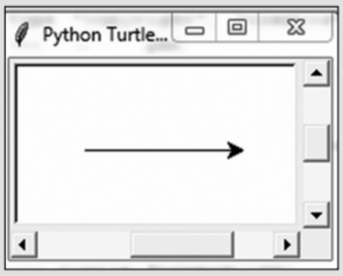
`right(P)`
Example:
`>>> import Turtle`
`>>> Turtle.right(90)`
`>>> Turtle.forward(100)`
	Rotates the Turtle in place a degree clockwise.
	Output:
	
	Explanation: Initially the Turtle is placed at the centre by default. The command **Turtle.right(90)** changes the direction of the Turtle to the right by 90 degrees. Finally, the arrowhead moves 100 pixels forward.

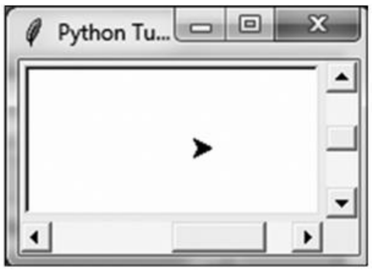
`backward(P)`
Example:
`>>> import Turtle`
`>>> Turtle.backward(100)`
	Moves the Turtle P pixels in a direction opposite to its current heading.
	Output:
	

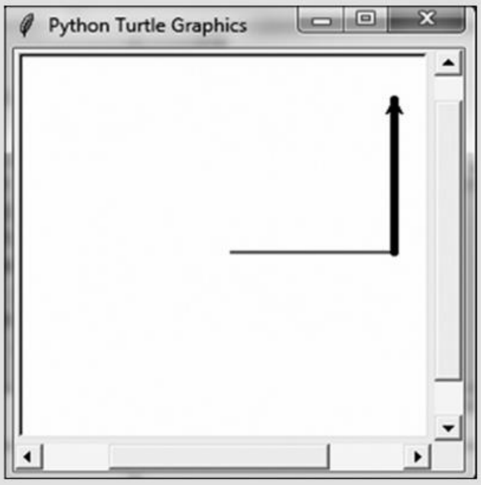
In Table 12.1, we used various methods to move the Turtle from one position to the other. As discussed above, the Turtle draws a line from one position to the other with the help of the pen. Table 12.2 illustrates various methods related to the state of a pen.

Table 12.2 Methods related to the state of a pen

Method	*Meaning*

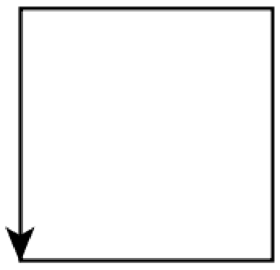
`Turtle.pendown()`
Example:
`>>> import Turtle`
`>>> Turtle.pendown()`
`>>> Turtle.forward(100)`
	Pulls the pen down. Draws when it moves from one place to the other.
	Output:
	
	Explanation: In the above example, the method **Turtle.pendown() draws different** shapes when it moves from one place to the other.

`Turtle.penup()`
Example:
`>>> import Turtle`
`>>> Turtle.penup()`
`>>> Turtle.forward(100)`
	Pulls the pen up. In this state, it just moves from one place to the other without drawing anything.
	Output:
	
	Explanation: The import Turtle method places the pen at the center of the circle. The **Turtle.penup()** doesn't allow a programmer to draw things, it just moves from one place to the other. When the statement Turtle.forward(100) is executed immediately after the `penup()` statement, it moves 100 pixels forward without drawing any line or shape.

`Turtle.pensize(width)`
Example:
`>>> import Turtle`
`>>> Turtle.forward(100)`
`>>> Turtle.pensize(5)`
`>>> Turtle.pensize(5)`
`>>> Turtle.left(90)`
`>>> Turtle.forward(100)`
	Sets the line thickness to the specified width.
	Output:
	
	Explanation: In the above code, initially the line is drawn 100 pixels ahead in the forward direction. The statement **Turtle.pensize(5)** increases the thickness to draw the figure from here onwards.

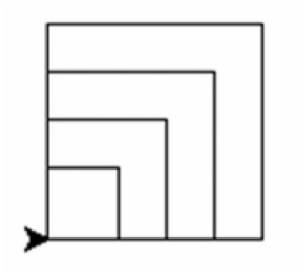
Q3. Write a program to draw square as shown below using Python turtle module.

a.



```
import Turtle            #import Turtle module
Turtle.forward(100)      #Move Turtle in forward direction
Turtle.left(90)  #Change the direction of Turtle to left by 90 degree
Turtle.forward(100)
Turtle.left(90)
Turtle.forward(100)
Turtle.left(90)
Turtle.forward(100)
```

b.



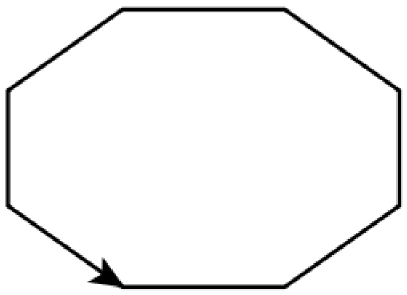
```
import Turtle
def  square(side):
    for i in range(4):
        Turtle.forward(side)
        Turtle.left(90)

square(20)
square(30)
square(40)
square(50)
```

Q4. Write a program to display polygon.

Ans.

```
import Turtle           #import Turtle module
Turtle.forward(50)
Turtle.left(45)
Turtle.forward(50)
Turtle.left(45)
Turtle.forward(50)
Turtle.left(45)
Turtle.forward(50)
Turtle.left(45)
Turtle.forward(50)
Turtle.left(45)
Turtle.forward(50)
Turtle.left(45)
Turtle.forward(50)
Turtle.left(45)
Turtle.forward(50)
```



Q5. Explain moving turtle to any location.

Ans.

When a programmer tries to run Python’s Turtle graphics program by default, the Turtle’s arrowhead (Cursor or Pen) is at the center of the graphics window at coordinate(0, 0) as shown inFigure 12.3.

```
>>> import Turtle    #import Turtle module
>>> Turtle.showTurtle ()
```

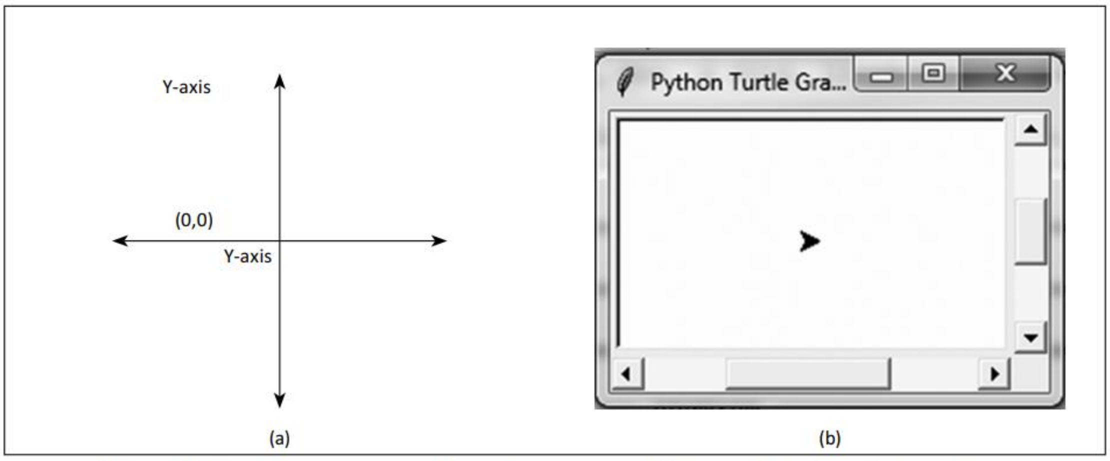


Figure 12.3 (a) Representation of coordinate system (b) Centre of Turtle graphics at (0, 0)

The method goto(x, y) is used to move the Turtle at specified points (x, y). The following exampleillustrates the use of goto(X, Y) method.

Example

```
>>> import Turtle
>>> Turtle.showTurtle ()
>>> Turtle.goto(0,-50)
```

Output

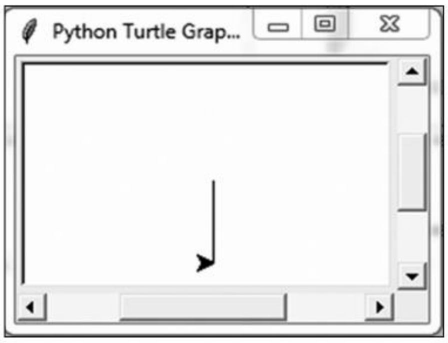


Figure 12.4

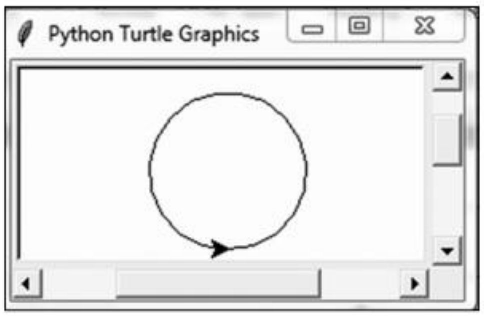
Explanation

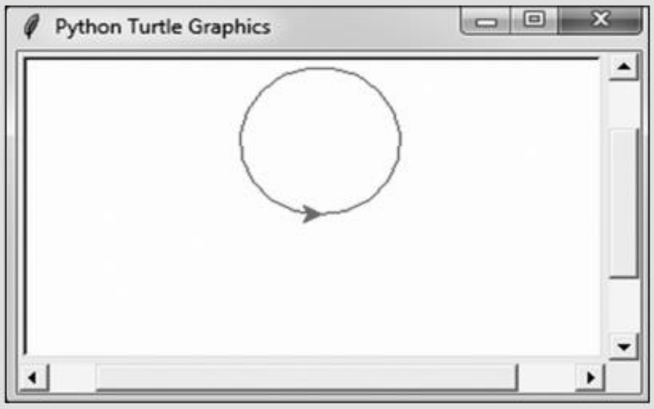
In the above example, the statement goto(0,-50) will move towards coordinate (0, -50).

Q6. Explain the color, bgcolor, circle and speed method of turtle.

Ans.

Method	*Meaning*
`Turtle.speed(integer _ paramter)`	The drawing speed of the Turtle must be in the range int 1 (slowest) to 10 (fastest) or 0 (instantaneous).

`Turtle.circle(radius, extent=None)`
Example:
`>>> import Turtle`
`>>> Turtle.circle (45)`
	Draws a circle with the given radius. The center is radius units left of the Turtle. The extent determines which part of the circle is drawn. If it is not given, the entire circle is drawn.
	Output:
	
	Explanation: The statement Turtle.circle (45) is used to draw a circle of radius 45 in an anti-clockwise direction.

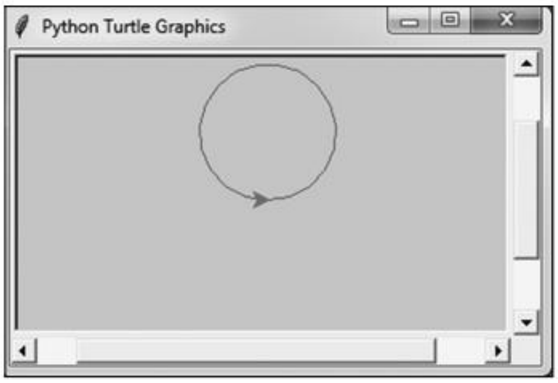
`Turtle.color(*args)`
Example:
`>>> import Turtle`
`>>> Turtle.color("red")`
`>>> Turtle.circle (45)`
	The color method is used to draw colorful animations.
	Output:
	
	Explanation: The above statement draws a circle in red color.

```
Turtle.bgcolor(*arg)
```
Returns the background color of the Turtle screen.

Example:

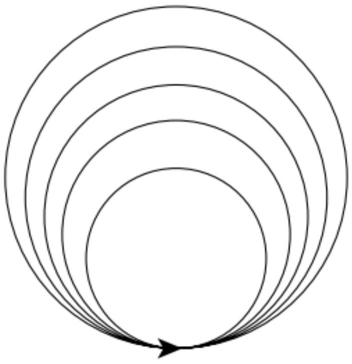
```
>>> import Turtle
>>> Turtle.color("red")
>>> Turtle.
bgcolor("pink")
```

Output:



Explanation: Changes the background color of the Turtle graphics window to pink.

Q7. Explain a program to display circles shown.



Ans.

```
import Turtle
Turtle.circle (45)
Turtle.circle (55)
Turtle.circle (65)
Turtle.circle (75)
Turtle.circle (85)
```

Explanation In the above program, the 5 circles are drawn with different radius, viz. 45, 55, 65, 75 and 85, respectively.

Q8. Explain methods for drawing turtles with colors.

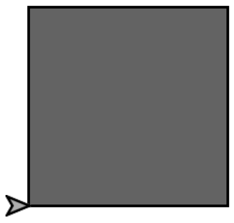
Ans.

A Turtle object contains methods for setting a color. In the above section, we have learnt how to draw different shapes. Table 12.4 lists methods to draw different shapes with different colors.

Table 12.4 More methods of Turtle related to color

Method	*Meaning*
`Turtle.color(c)`	Sets the pen's color
`Turtle.fillcolor(C)`	Sets the pen's fill color to **'C'**
`Turtle.begin _ fill()`	Calls this method before filling a shape
`Turtle.end _ fill()`	Fills the shape drawn before the last call to **begin_fill**
`Turtle.filling()`	Returns the fill state. True is filling, False if not filling.
`Turtle.clear()`	Clears the window. The state and position of window is not affected.
`Turtle.reset()`	Clears the window and resets the state and position to its original default value
`Turtle.screensize()`	Sets the width and height of the canvas
`Turtle.showTurtle()`	Makes the Turtle visible
`Turtle.hideTurtle()`	Makes the Turtle invisible
`Turtle.write(msg, move,align,font=fontname, fontsize, fonttype)`	Writes a message on the Turtle graphics window

- **Write a program to draw a color filled square box as shown.**

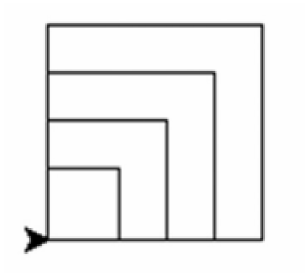


```
import Turtle
Turtle.fillcolor ("gray") #Fill gray color inside the square
Turtle.begin_fill ()
Turtle.forward(100)
Turtle.left(90)
```

```
Turtle.forward(100)
Turtle.left(90)
Turtle.forward(100)
Turtle.left(90)
Turtle.forward(100)
Turtle.left(90)
Turtle.end_fill()
```

Q9. Explain drawing basic shapes using iteration.

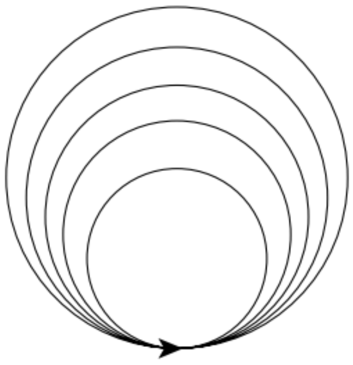
Ans.



```
import Turtle
def  square(side):
    for i in range(4):
        Turtle.forward(side)
        Turtle.left(90)

square(20)
square(30)
square(40)
square(50)
```

b.



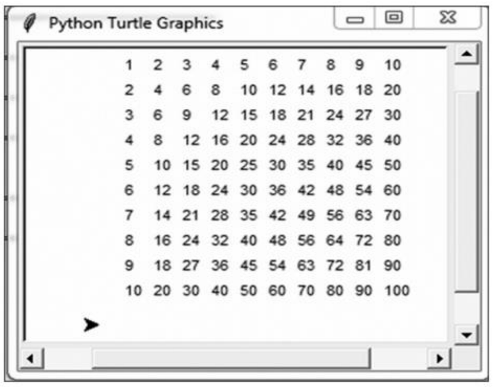
```
import Turtle
Turtle.circle (45)
Turtle.circle (55)
Turtle.circle (65)
Turtle.circle (75)
Turtle.circle (85)
```

Explanation In the above program, the 5 circles are drawn with different radius, viz. 45, 55, 65, 75 and 85, respectively.

Q9. Write a program to display the multiplication table from 1 to 10 in turtle graphics window.

```
import Turtle as t
t.penup()
```

```
x =  -100
y =  100
t.goto(x,y)  #Move pen at location x and y
t.penup()
for i in range(1,11,1):   # value of i varies from 1 to 10
    y = y - 20
    for j  in range(1,11,1):  # Value of j varies from 1 to 10
        t.penup()
        t.speed(1)
        t.forward(20)
        t.write(i*j)
   t.goto(x, y)
```



```
1   2   3   4   5   6   7   8   9   10
2   4   6   8   10  12  14  16  18  20
3   6   9   12  15  18  21  24  27  30
4   8   12  16  20  24  28  32  36  40
5   10  15  20  25  30  35  40  45  50
6   12  18  24  30  36  42  48  54  60
7   14  21  28  35  42  49  56  63  70
8   16  24  32  40  48  56  64  72  80
9   18  27  36  45  54  63  72  81  90
10  20  30  40  50  60  70  80  90  100
```

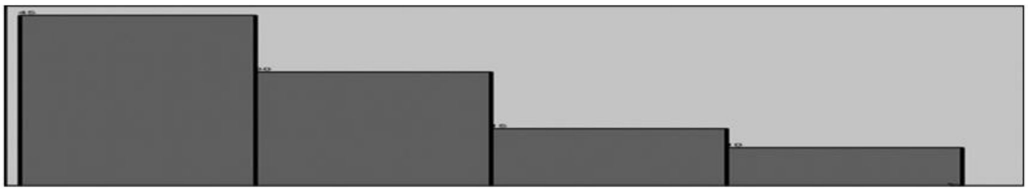
Q10. Write a program to draw a barchart using turtle for sample data given below.

Table 12.5 Sample data to draw a chart

Web Browser	*Percentage*
Mozilla Firefox	45%
Google Chrome	30%
Internet Explorer	15%
Others	10%

Ans.

```
import Turtle
def Draw_Bar_Chart(t, height):
    t.begin_fill()              # start filling this shape
    t.left(90)
    t.forward(height)
    t.write(str(height))
    t.right(90)
    t.forward(40)
    t.right(90)
    t.forward(height)
    t.left(90)
    t.end_fill()                # stop filling this shape
Mozilla_Firefox = 45
Chrome =  30
IE = 15
Others = 10
S = [Mozilla_Firefox, Chrome, IE,Others]  # Sample Data
maxheight = max(S)
num_of_bars = len(S)
border = 10
w = Turtle.Screen()              # Setting up attributes of Window
w.setworldcoordinates(0,0,40*num_of_bars + border, maxheight + border)
w.bgcolor("pink")
T1 = Turtle.Turtle()
T1.color("#000000")
T1.fillcolor("#DB148E")
T1.pensize(3)
for a in S:
    Draw_Bar_Chart(T1,a)
```



Q11. Describe the need of file handling.

Ans.

Often the output screen of a laptop or monitor is not enough to display all the data. This usually happens when the data is large and only a limited amount can be displayed on the screen and stored in the memory. Computer memory is volatile, so even if a user tries to store the data in the memory, its contents would be lost once a program is terminated. If the user needs the same data again, either it has to be entered through a keyboard or regenerated programmatically. Obviously, both these operations are tedious. Therefore, to permanently store the data created in a program, a user needs to save it in a File on a disk or some other device. The data stored in a file is used to retrieve the user's information either in part or whole.

Various operations carried out on a file are

(a) Creating a file

(b) Opening a file
(c) Reading from a file
(d) Writing to a file
(e) Closing a file

Q12. Explain opening file in Python in details.

Ans.

A file needs to open before we can perform read and write operations on it. To open a file, a user needs to first create a file object which is associated with a physical file. While opening a file, a user has to specify the name of the file and its mode of operation. The syntax to open a file is:

file object = open(File_Name, [Access_Mode],[Buffering])

The above syntax to open a file returns the object for file name. The mode operation used in the syntax above is a string value which indicates how a file is going to be opened. Table 13.1 describes the various modes used to open a file. The third parameter within the open function is an optional parameter, which controls the buffering of a file. If this parameter is set to 1, line buffering is performed while accessing the file. If the buffering value is set to 0 then no buffering takes place. If we specify the buffering value as an integer greater than 1 then the buffering action is performed with the indicated buffer size.

Mode	*Description*
R	Opens a file for reading
W	Opens a new file for writing. If a file already exists, its contents are destroyed.
A	Opens a file for appending data from the end of the file
Wb	Opens a file for writing binary data
Rb	Opens a file for reading binary data

```
F1 = open ("Demo.txt","r") #Open File from Current Directory
F2 = open("c:\Hello.txt","r")
```

The above example opens a file named Hello.txt located at C: in read mode.

Q13. Explain Writing text to a file along with methods and program to write the sentences.

Ans. The open function creates a file object. It is an instance of _io.TextIOWrapper class. This class contains the methods for reading and writing data. Table 13.2 lists the methods defined in the _io.TextIOWrapper class.

Table 13.2 Methods for reading and writing data

_io.TextIOWrapper	*Meaning*
str readline()	Returns the next line of a file as a string
list readlines()	Returns a list containing all the lines in a file

str read([int number])	Returns a specified number of characters from a file. If the argument is omitted then the entire content of the file is read.
Write (str s)	Writes strings to a file
close()	Closes a file

Once a file is opened, the write method is used to write a string to a file.

PROGRAM 13.1 | Write a program to write the sentences given below the file **Demo1.txt**.

Hello, How are You?
Welcome to The chapter File Handling.
Enjoy the session.

```
def main():
    obj1 = open("Demo1.txt","w") #Opens file in Write mode
    obj1.write(" Hello, How are You ? \n")
    obj1.write(" Welcome to The chapter File Handling. \n ")
    obj1.write(" Enjoy the session. \n ")
main()   # Call to main function
```

Explanation In the above program, initially the file **Demo1.txt** is opened in **'w'** mode, i.e. **write mode**. If the file **Demo1.txt** does not exist, the **open** function creates a new file. If the file already exists, the contents of the file will be over written with new data.

When a file is opened for reading or writing, a special pointer called file pointer is positioned internally in the file. Reading and writing operation within the file starts from the pointer's location. When a file is opened, the file pointer is set at the beginning of the file. The file pointer moves forward as soon as we start reading from the file or write the data to the file.

The step-wise execution and position of the file pointer is updated in the following manner by the Python interpreter.

Initially, a call is made to the main() function. The statement obj1 = open("Demo1.txt","w") opens Demo1.txt in write mode. The file is created and initially the file pointer is at the starting of the file as shown in Figure 13.1.

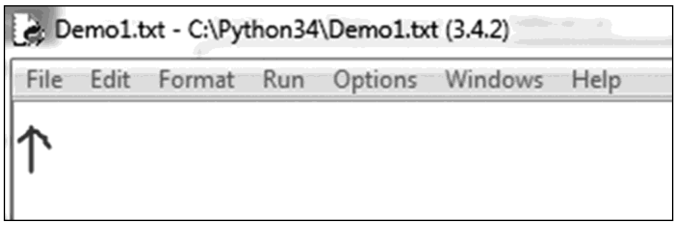
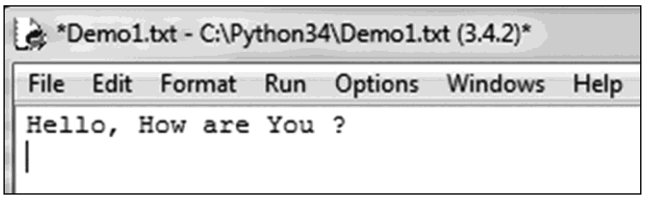

Demo1.txt - C:\Python34\Demo1.txt (3.4.2)
File Edit Format Run Options Windows Help

Figure 13.1 Initial position of the file pointer

The following statement within the program invokes the write method on the file object to write strings into the file.

obj1.write(" Hello, How are You ? \n")

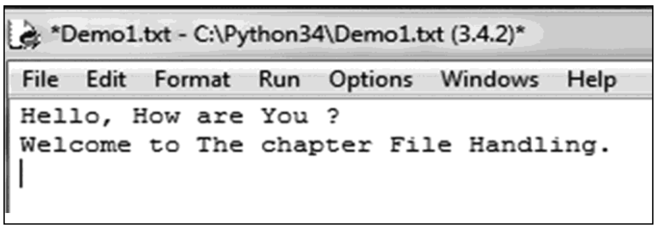
After successful execution of the above statement, the file pointer is located as shown in Figure 13.2.


Demo1.txt - C:\Python34\Demo1.txt (3.4.2)
File Edit Format Run Options Windows Help

```
Hello, How are You ?
```

Figure 13.2

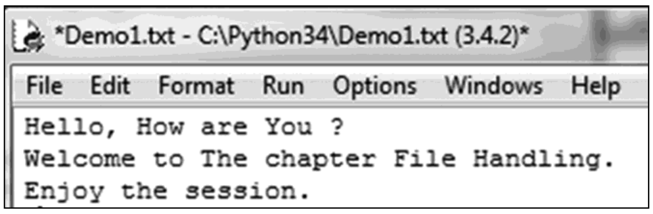
After successful execution of a second statement, i.e. obj1.write(“Welcome to The chapter FileHandling. \n “), the file pointer is located as shown in Figure 13.3.


Demo1.txt - C:\Python34\Demo1.txt (3.4.2)
File Edit Format Run Options Windows Help

```
Hello, How are You ?
Welcome to The chapter File Handling.
```

Figure 13.3

Finally, after the execution of the third statement, i.e. obj1.write(“ Enjoy the session.\n”), thecontents of the file are updated as shown in Figure 13.4.


Demo1.txt - C:\Python34\Demo1.txt (3.4.2)
File Edit Format Run Options Windows Help

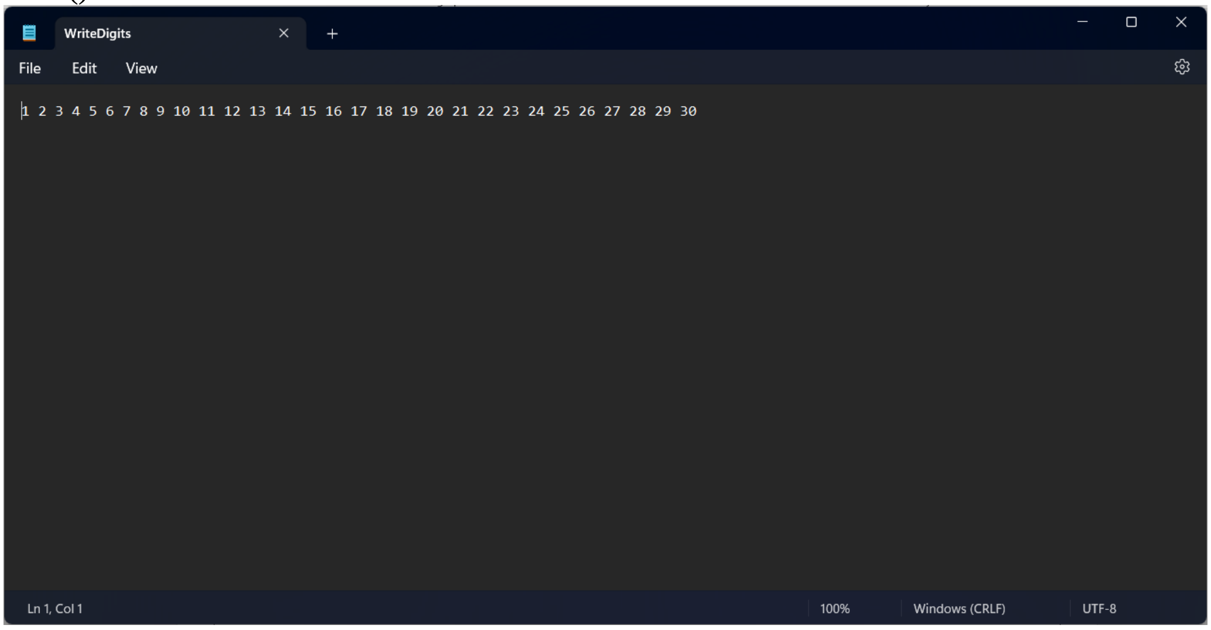
```
Hello, How are You ?
Welcome to The chapter File Handling.
Enjoy the session.
```

Figure 13.4

Q14. Write a program to print numbers from 1 to 30 to the output file writedigits.txt.

Ans.

```
def main():
    obj1 = open("WriteDigits.txt", "w")
    for x in range (1,31):
        x = str(x)
        obj1.write(x)
        obj1.write(" ")
    obj1.close()
main()
```

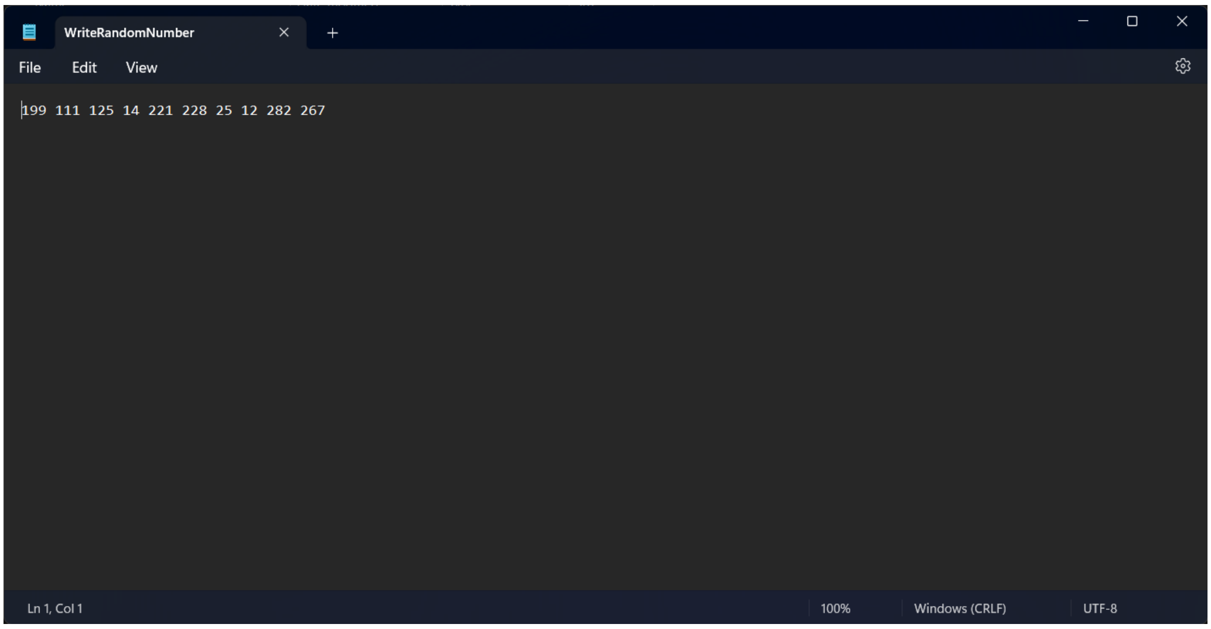


```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
```

Q15. Generate 10 random numbers within a range 1 to 30 and write them to a file WriteRandomNumbers.txt.

Ans.

```
from random import randint
fp1 = open("WriteRandomNumber.txt", "w")
for x in range(10):
    x = randint(1, 300)
    x = str(x)
    fp1.write(x + " ")
fp1.close()
```



```
199 111 125 14 221 228 25 12 282 267
```

Q16. Explain reading text from a file and write down a program using read method.
Ans.

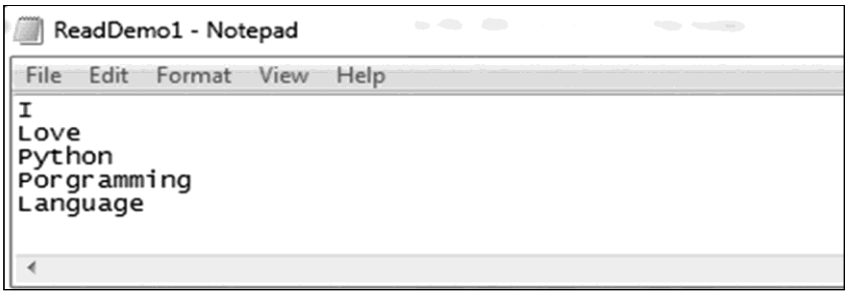
Once a file is opened using the open () function, its content is loaded into the memory. The pointer points to the very first character of the file. To read the content of the file, we open the file in 'r' (read) mode. The following code is used to open the file ReadDemo1.txt.

```
>>> fp1 = open("ReadDemo1.txt","r")
```

There are several ways to read the content of a file. The two common approaches are:

a. Use read()method to read all the data from a file and return as one complete string.
b. Use readlines()method to read all data and return as a list of strings.

The following program demonstrates the use of the read()method to read the content of thefile ReadDemo1.txt. The content of the file is as shown in Figure.



```
I
Love
Python
Porgramming
Language
```

Program 1:

```
fp = open("ReadDemo1.txt","r") #Open file in read mode
text = fp.read()         # Read Whole File exactly  once
print(text)          #Print the contents of file
```

Output

```
I
Love
Python
Programming
Language
```

Explanation Initially the file ReadDemo1.txt is opened in read mode. The content of the file is read using the `read()` method. It reads all the content of the file exactly once and returns all the data as a single string.

Program 2:

```
fp = open("ReadDemo1.txt","r")
for line in fp:
    print(line)
```

Output

```
I

Love

Python

programming

Language
```

Explanation In the above program, the for loop views the file object as a sequence of lines of text. In each iteration of the for loop, the loop variable **line** is bound to the next line from the sequences of lines present in the text file. Note the output of above program. The **print**() statement prints one extra new line. This is because each line of the input file retains its new line character.

Q17. Explain reading multiple items on one line.
Ans.

Many text files contain multiple items in a single line. The method split () for strings allows us to read more than one piece of information in a line. The split () returns all the items in a list. In short, it splits a string into separate items and all the items are separated by spaces or tabs.

The following example written in Python IDLE interpreter gives more details about the split() method.

```
>>> str = 'I am Loving The Concepts of File Handling'
>>> str.split()
['I', 'am', 'Loving', 'The', 'Concepts', 'of', 'File', 'Handling']
#
>>> for i in range(len(str)):
         print(str[i])

I
am
Loving
The
Concepts
of
File
Handling
```

Explanation The above example simply splits the string and stores the content to a list. Finally, the for loop is used to access and display each item of the list.

Q18. Describe the following in short.

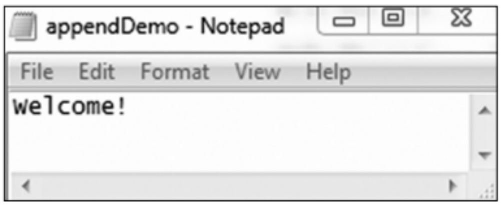
a. Appending Data.

Ans.

The append 'a' mode of a file is used to append data to the end of an existing file. The following program demonstrates the use of append mode.

PROGRAM 13.10 | Write a program to append extra lines to a file name **appendDemo.txt**.

The content of **appendDemo.txt** file is as shown in Figure 13.10.

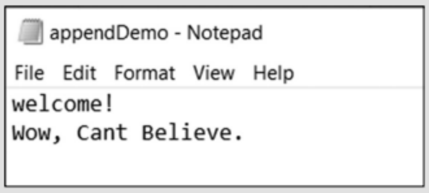


```
welcome!
```

Figure 13.10

```
fp1=open('appendDemo.txt','a') # Open file in append file
fp1.write('\nWow, Cant Believe.')# Append contents to a file
fp1.close() #Close file
```

Output



```
welcome!
Wow, Cant Believe.
```

b. The seek() function.

Ans.

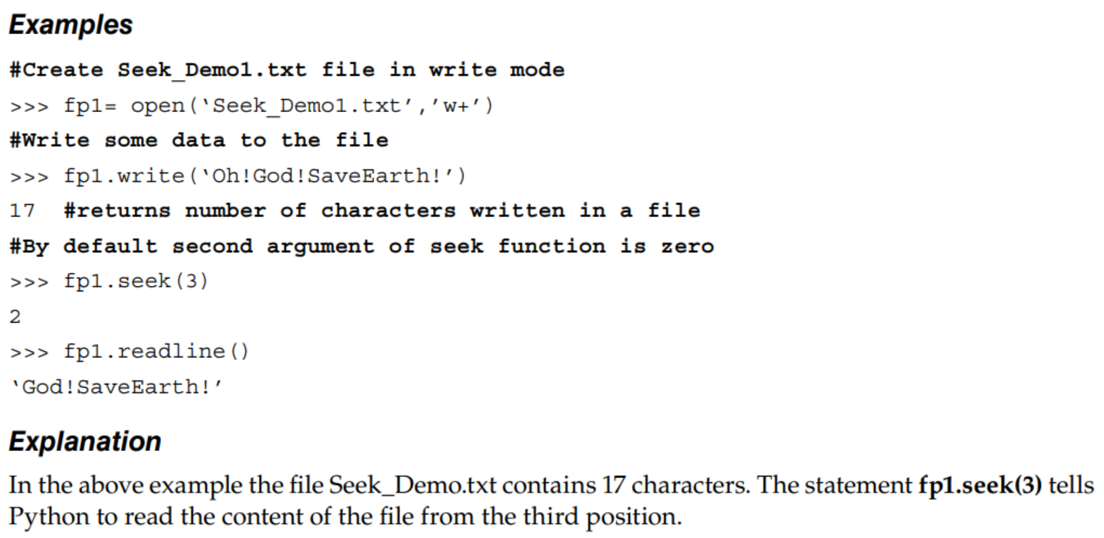
So far, we have learnt that data is stored and subsequently read from a file in which it is stored. When a file is opened, we can imagine an imaginary pointer positioned at the beginning of the file. What about reading the content of files from random positions? Python provides an inbuilt function called seek()for moving the pointer explicitly to any position in a file.

Thus, the seek()method is used to set the file pointer to a specific position in a file. The syntaxfor seek()function is:

File_object.seek(offset, whence)

where offset indicates the number of bytes to be moved from the current position of the pointer and whence indicates the point of reference from where the bytes are to be moved from. The valueof whence can be determined from Table 13.3.

Value	*Meaning*
0	The position is relative to the start of the file, i.e. it sets the pointer at the beginning of the file. This is a default setting if we don't supply '0' as the second argument to the seek() function.
1	The position is relative to the current position.
2	The position is relative to the end of the file.



Examples

```
#Create Seek_Demo1.txt file in write mode
>>> fp1= open('Seek_Demo1.txt','w+')
#Write some data to the file
>>> fp1.write('Oh!God!SaveEarth!')
17  #returns number of characters written in a file
#By default second argument of seek function is zero
>>> fp1.seek(3)
2
>>> fp1.readline()
'God!SaveEarth!'
```

Explanation

In the above example the file Seek_Demo.txt contains 17 characters. The statement **fp1.seek(3)** tells Python to read the content of the file from the third position.