**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 window will then appear (Figure 12.1).

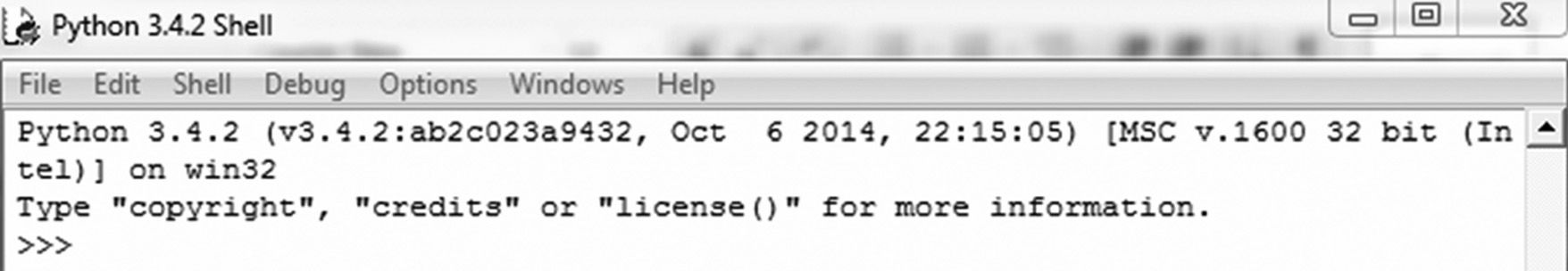


Figure 12.1

* STEP 2: At the Python’s statement prompt >>> type the following command to import the Turtle 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 displayed as shown in Figure 12.2.

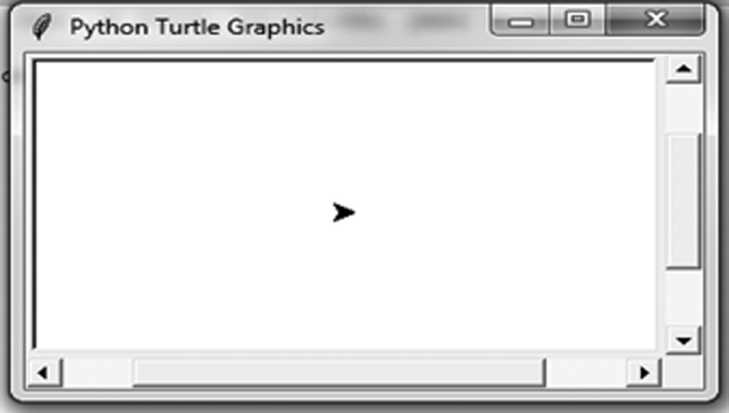


Figure 12.2 Python’s Turtle Graphics Window

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

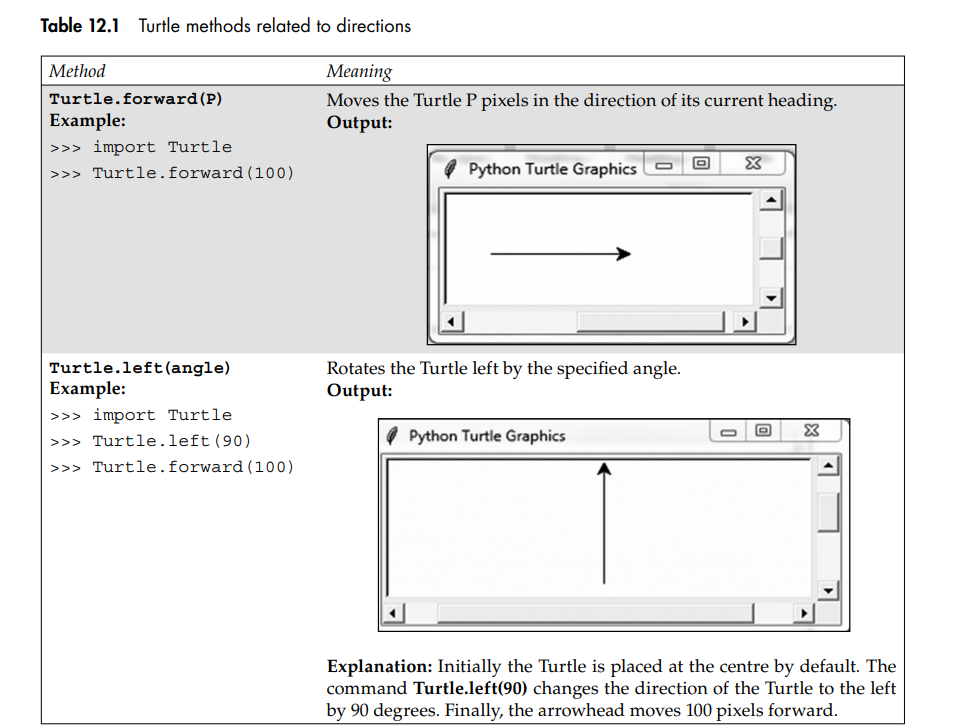
Initially, the Turtle is positioned at the center of the window.

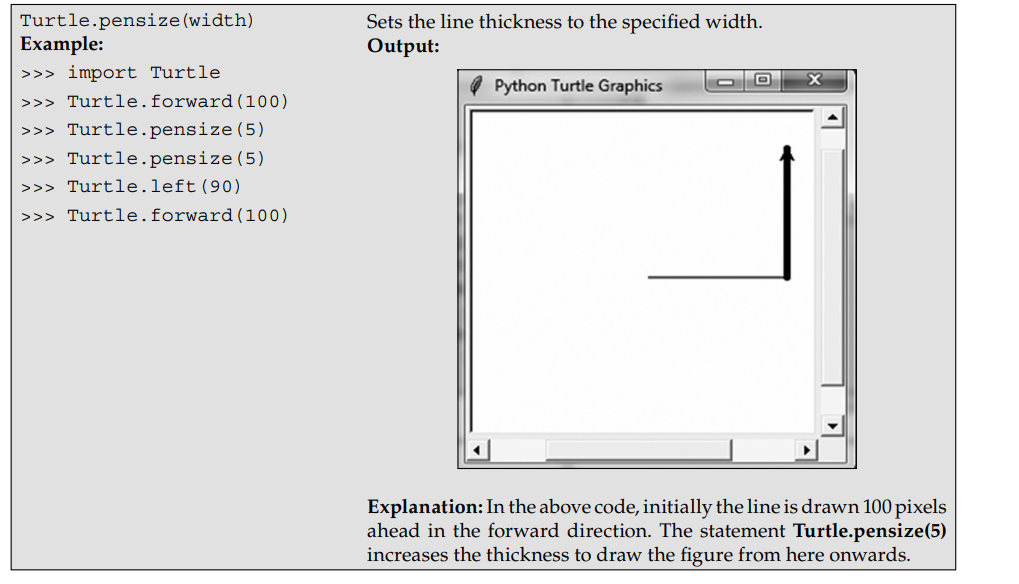
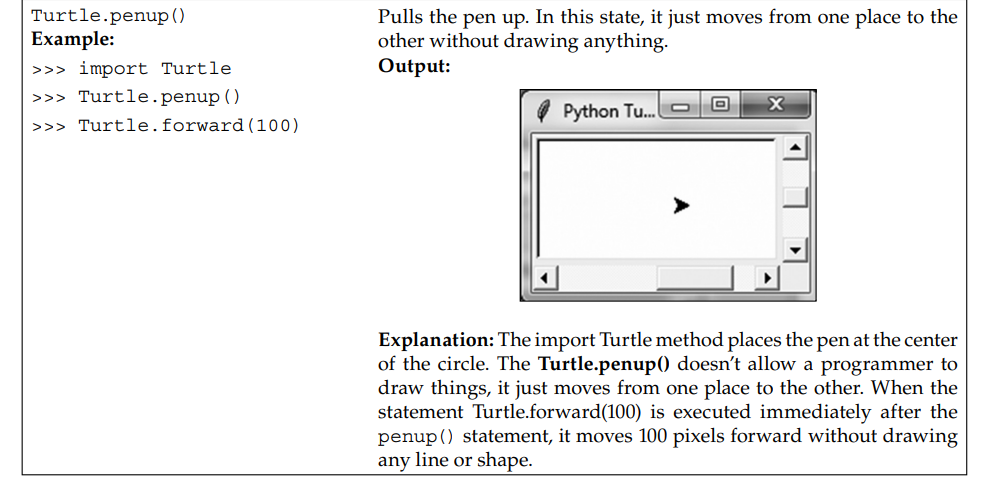
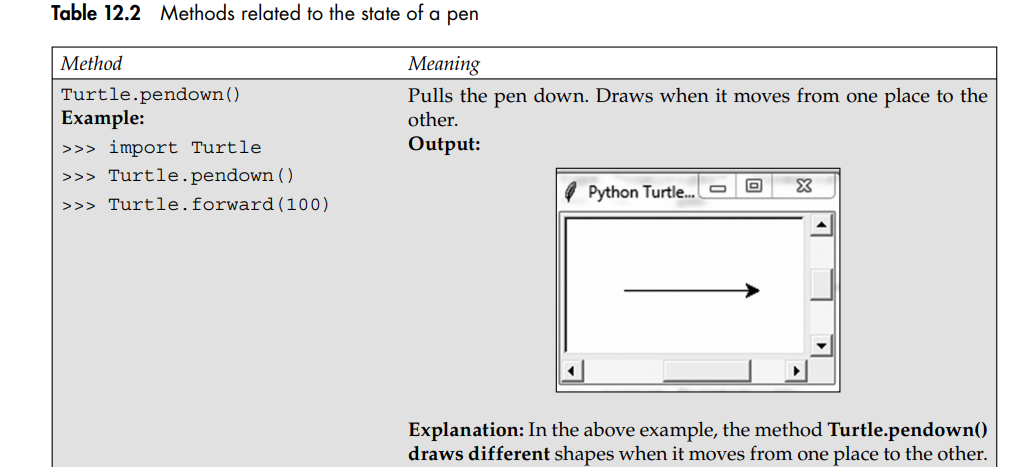
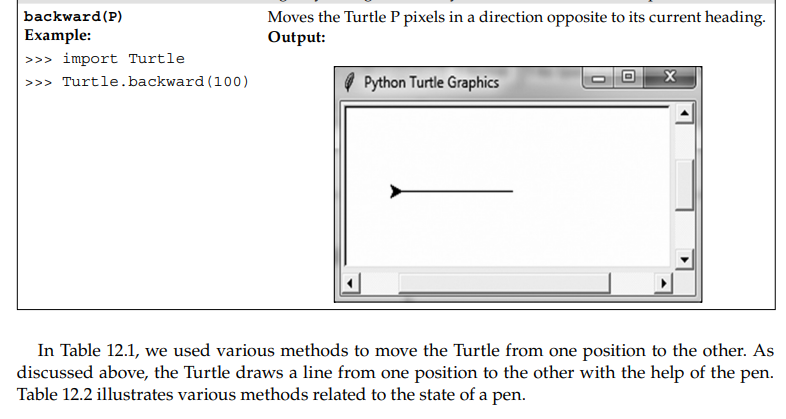
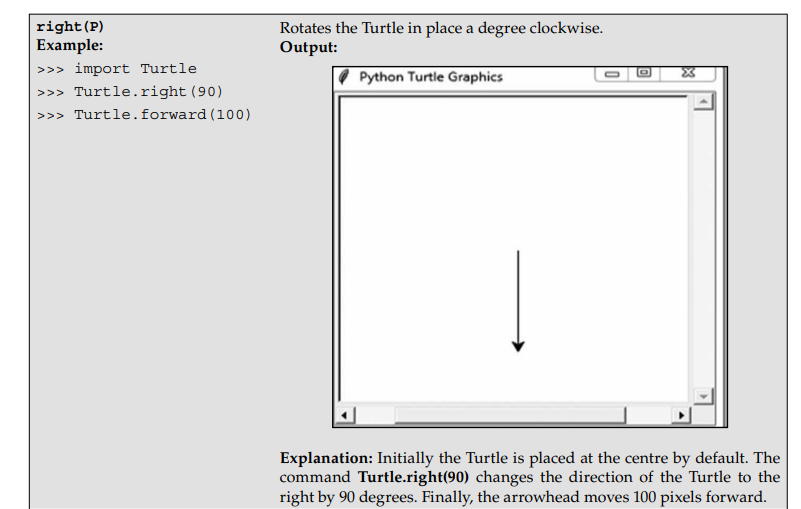
**Q2. Explain moving turtle in any direction.**

**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.

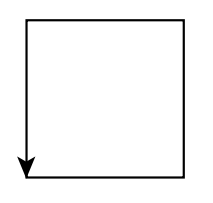
The imported Turtle module uses a pen to draw shapes. It can be used to move and draw lines in any direction. Python contains methods for moving the pen, setting the pen’s size, lifting and putting the pen down. By default, the pen is down, i.e. it draws a line from the current position to the new position.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

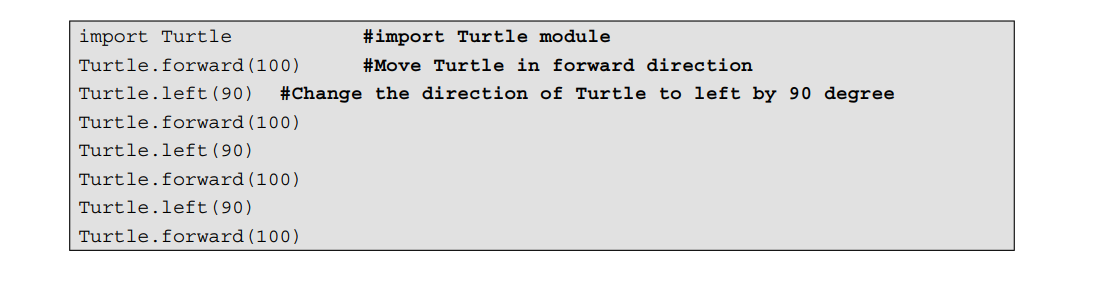
****

****

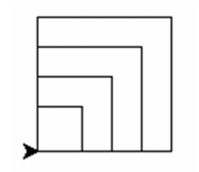
**Q3. Write a program to draw square as shown below using Python turtle module.**



****

****

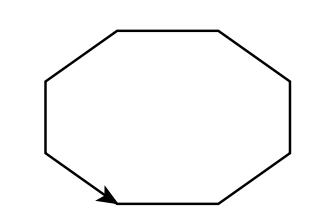
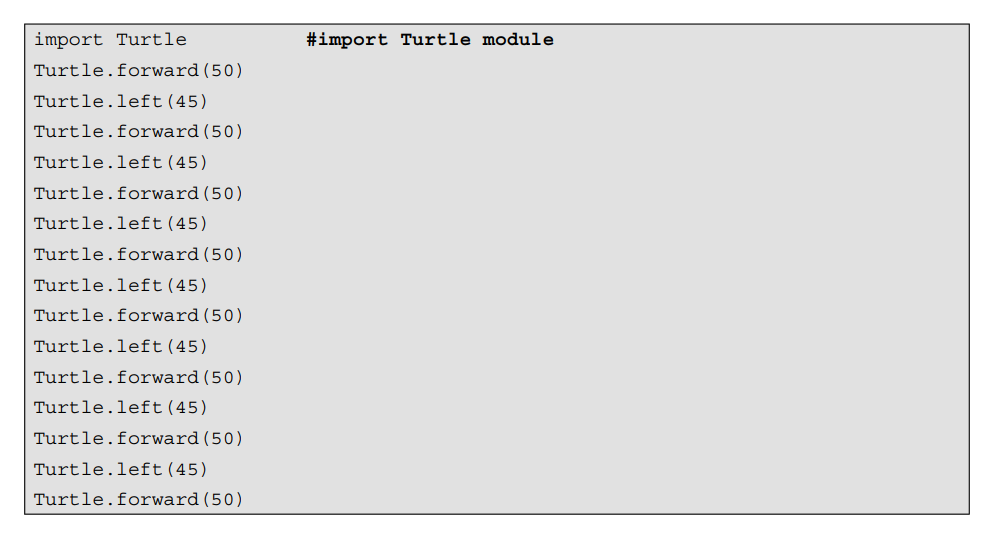


****

****

**Q4. Write a program to display polygon.**

**Ans.**

****

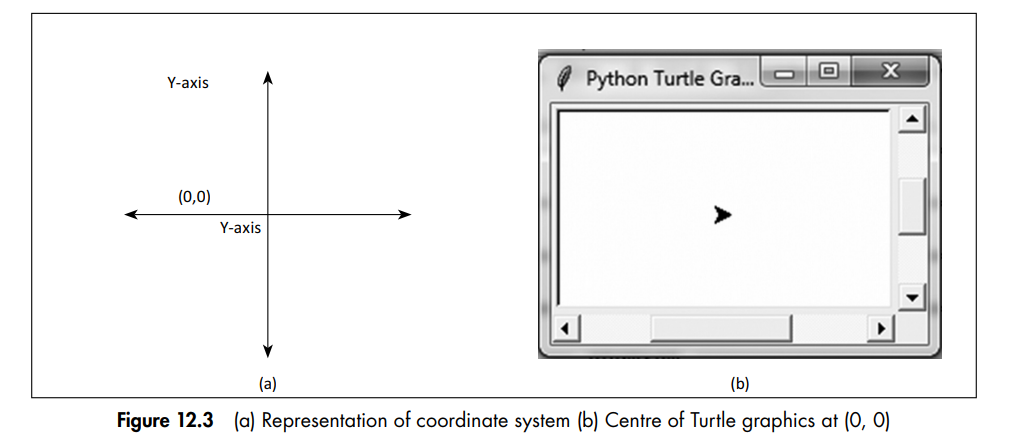
**Q5. Explain moving turtle to any location.**

**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 in Figure 12.3.

>>> import Turtle #import Turtle module

>>> Turtle.showTurtle ()

****

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

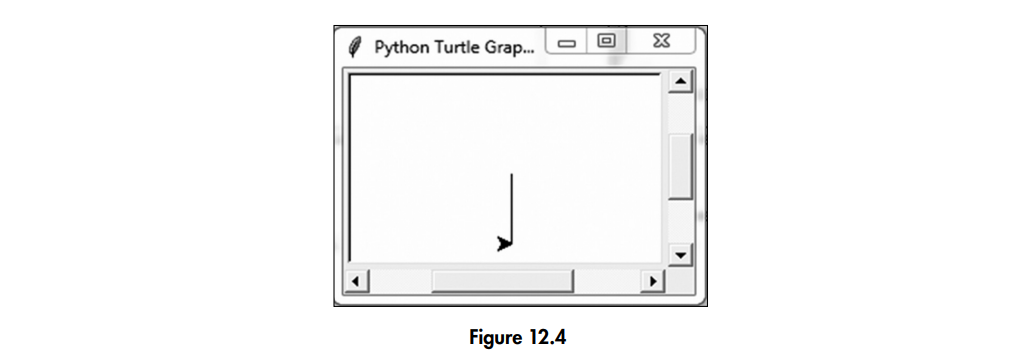
#### Example

>>> import Turtle

>>> Turtle.showTurtle ()

>>> Turtle.goto(0,-50)

#### Output

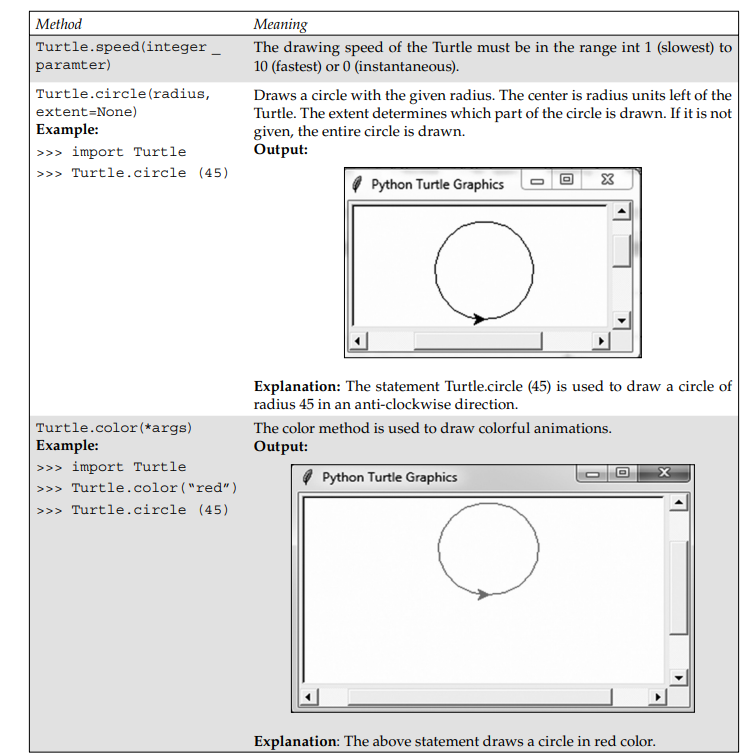
****

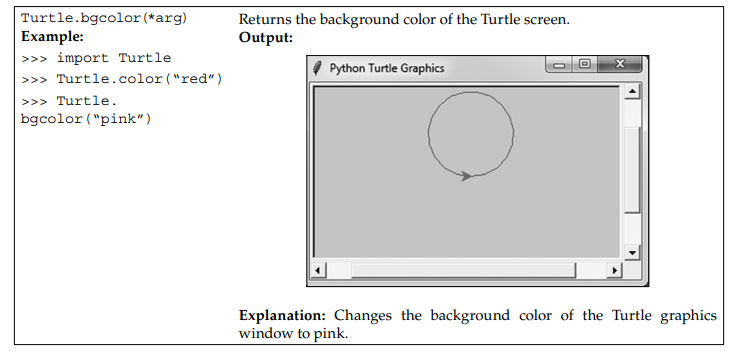
#### Explanation

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

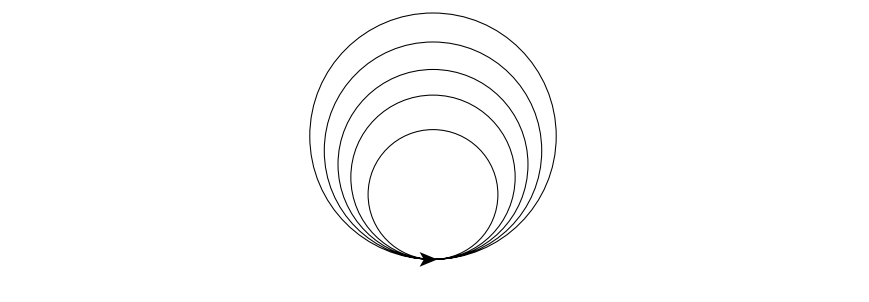
**Q6. Explain the color, bgcolor, circle and speed method of turtle.**

**Ans.**

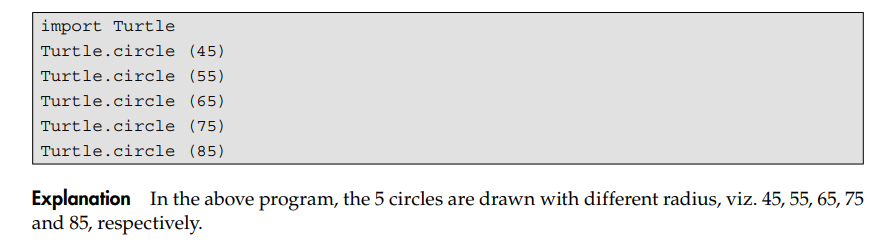
****

****

**Q7. Explain a program to display circles shown.**

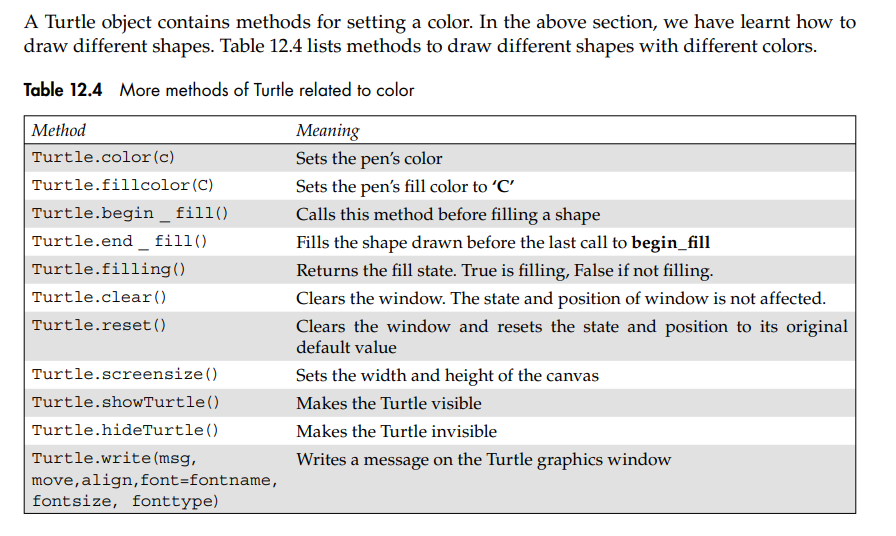
****

**Ans.**

****

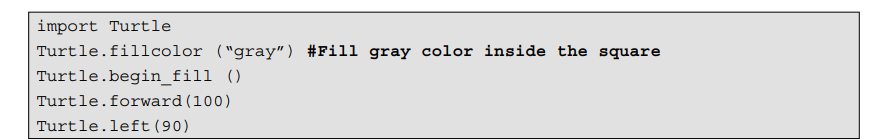
**Q8. Explain methods for drawing turtles with colors.**

**Ans.**

****

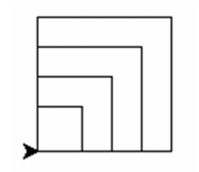
* **Write a program to draw a color ﬁlled square box as shown**.



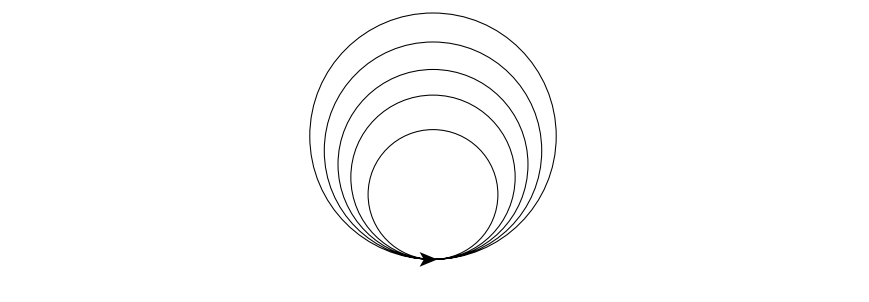
**** ****

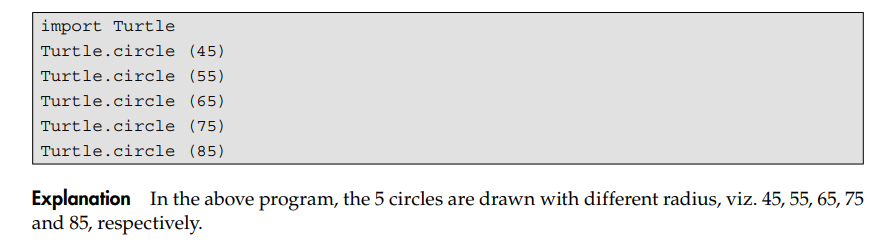
**Q9. Explain drawing basic shapes using iteration.**

**Ans.**

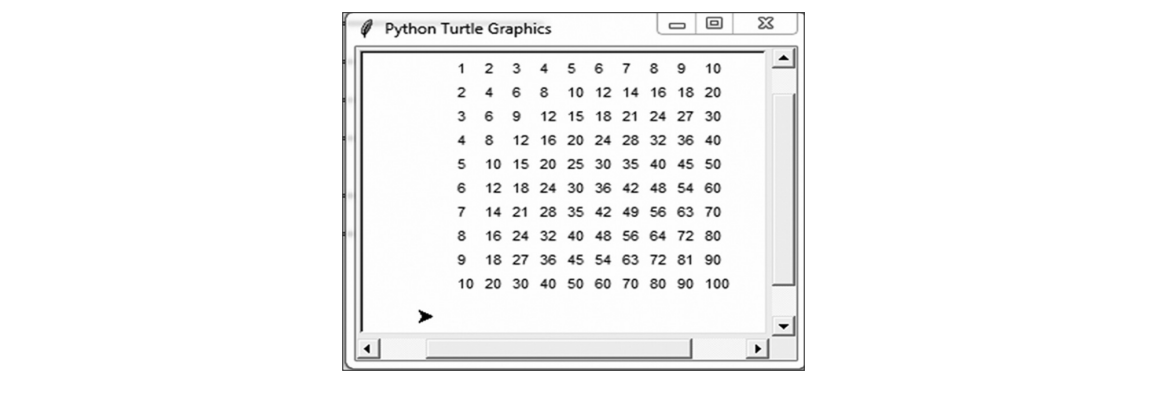
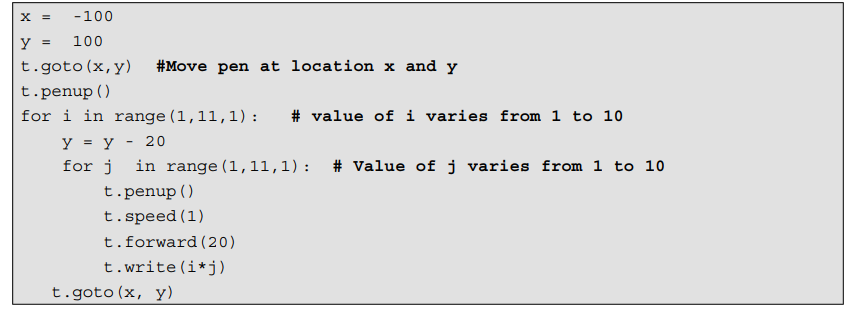
****

****

**b. **

****

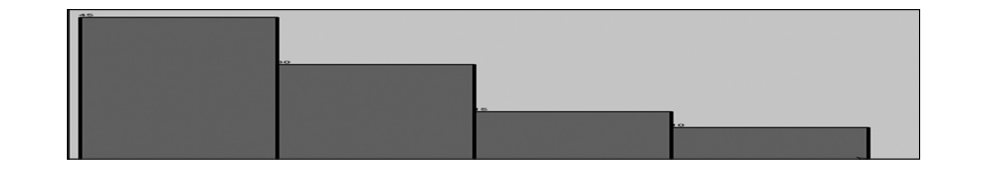
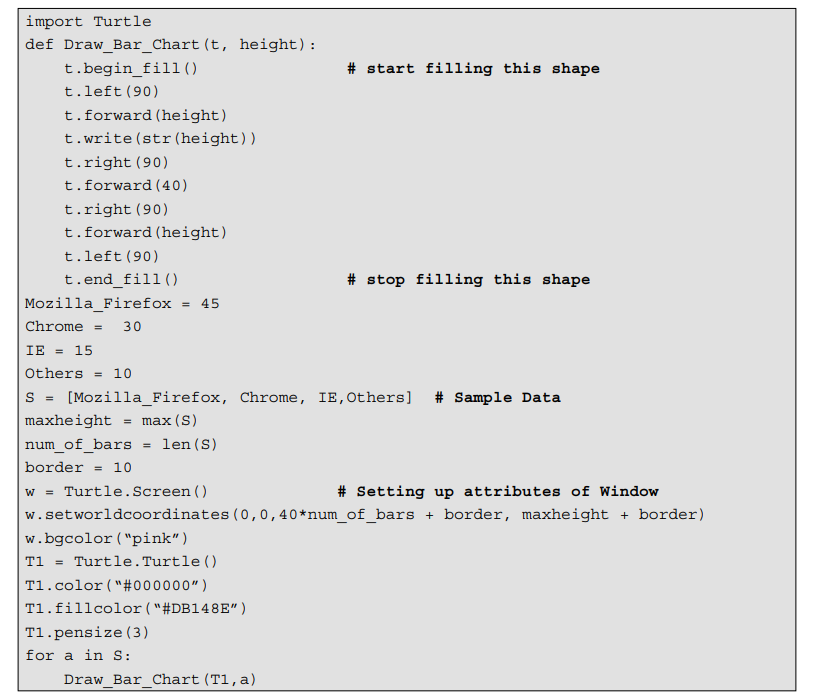
**Q9. Write a program to display the multiplication table from 1 to 10 in turtle graphics window.**

****

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

****

**Ans.**

****

**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

* + 1. Creating a file
    2. Opening a file
    3. Reading from a file
    4. Writing to a file
    5. Closing a file

**Q12. Explain opening file in Python in details.**

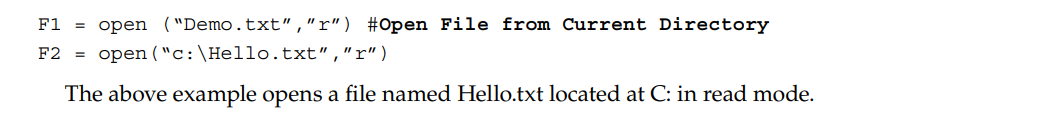
**Ans.**

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 |

****

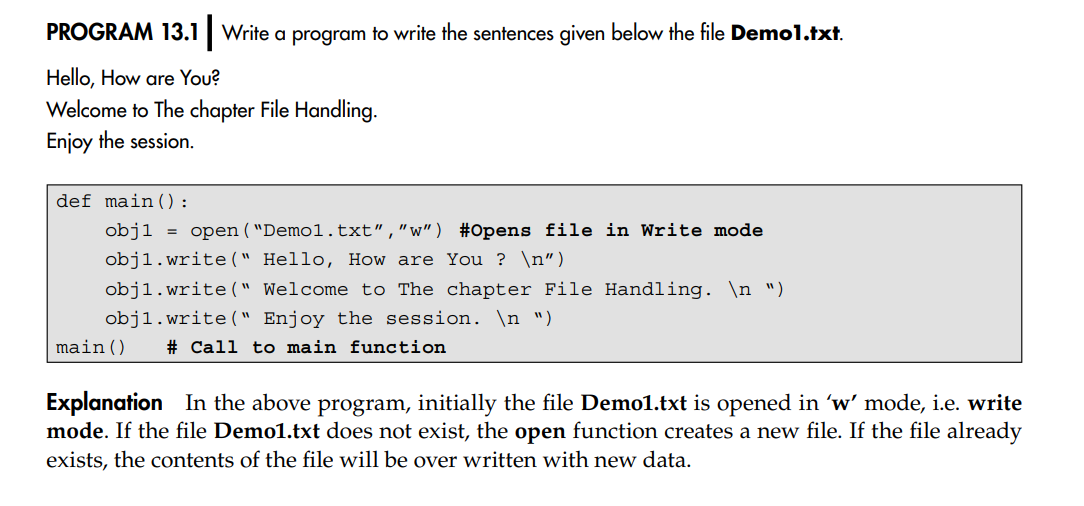
**Q13. Explain Writing text to a file along with methods and program to write the sentences.**

**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.

****

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.

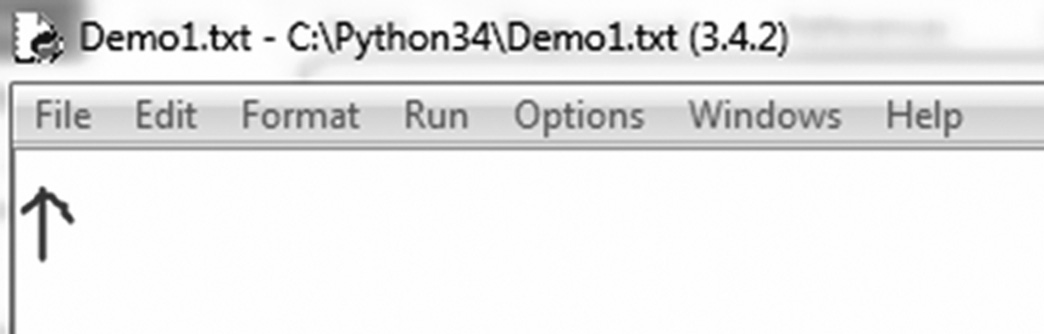


Figure 13.1 Initial position of the ﬁle pointer

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.

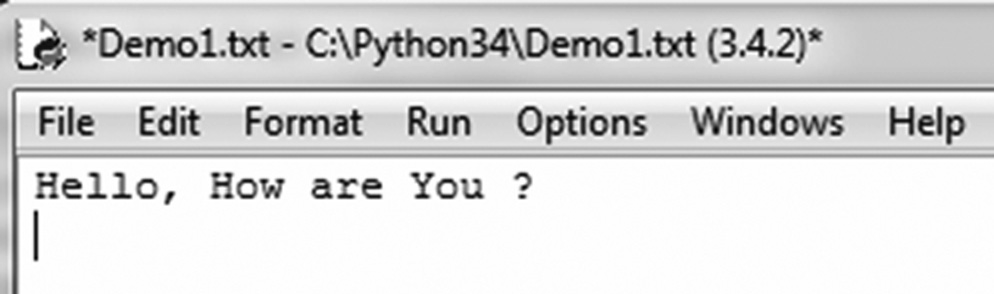


Figure 13.2

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

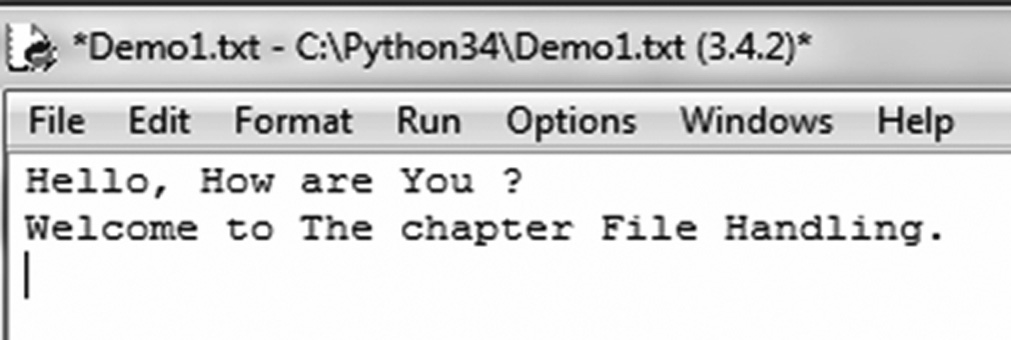


Figure 13.3

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

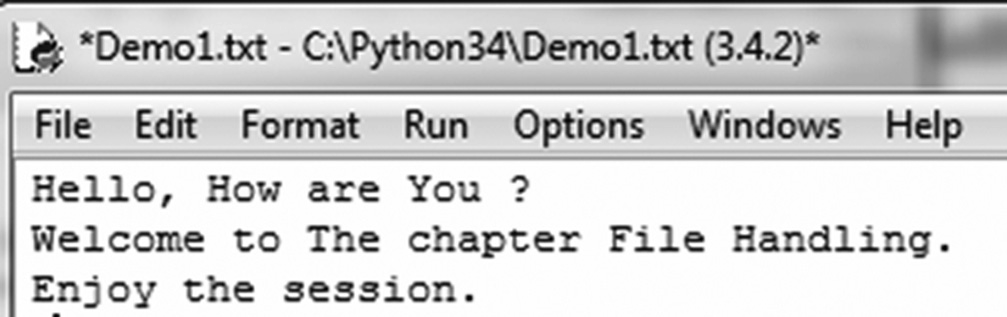
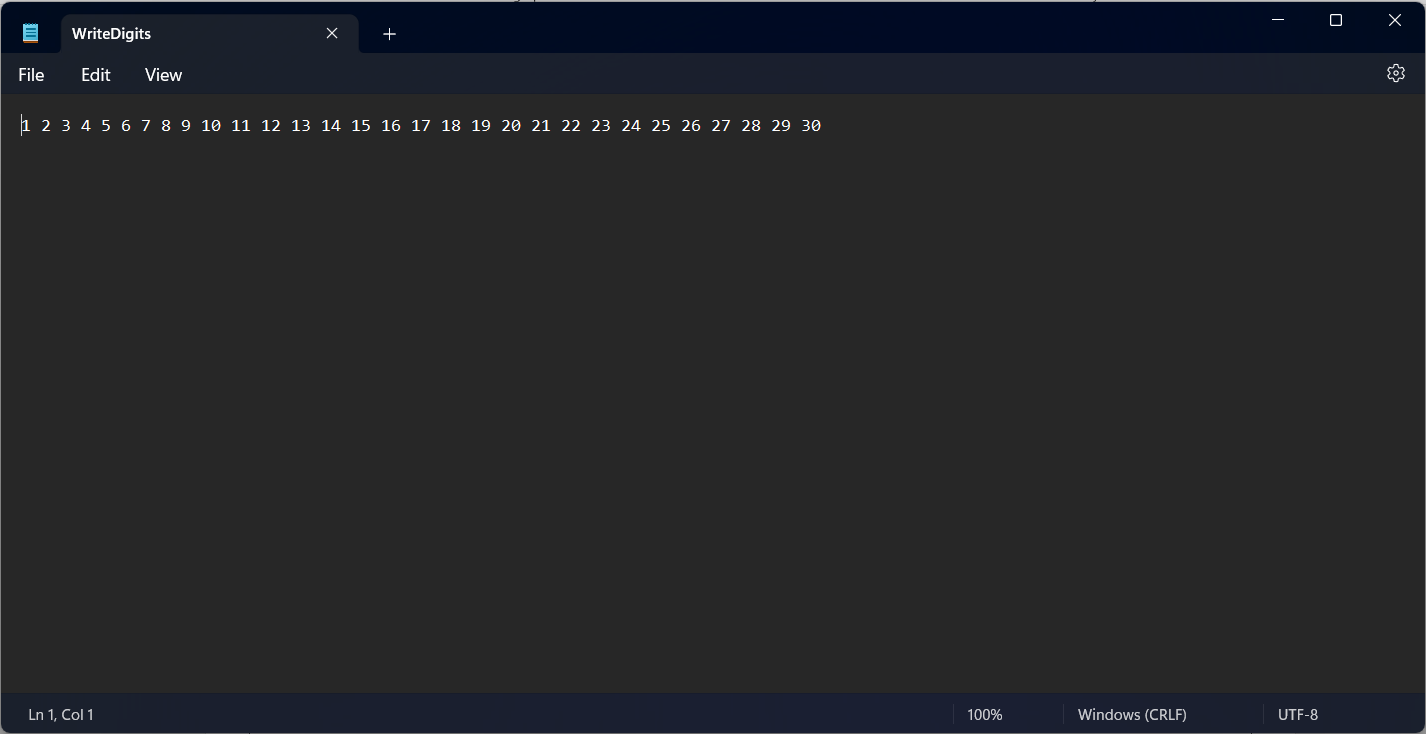


Figure 13.4

**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()

**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()

****

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

**Ans.**

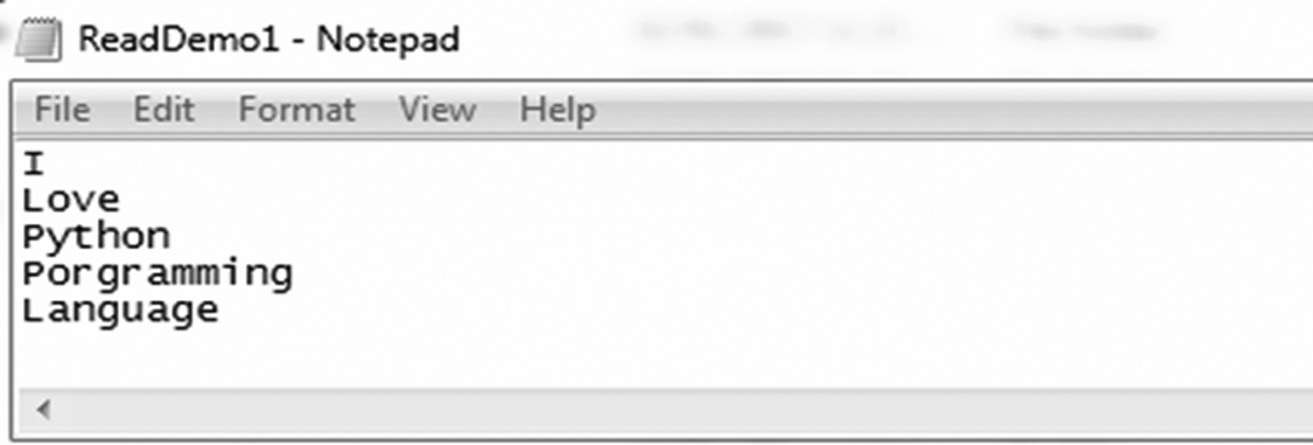
Once a file is opened using the open () function, its content is loaded into the memory. The pointer points to the very first character of the file. To read the content of the file, we open the file in ‘r’ (read) mode. The following code is used to open the file ReadDemo1.txt.

>>> fp1 = open(“ReadDemo1.txt”,”r”)

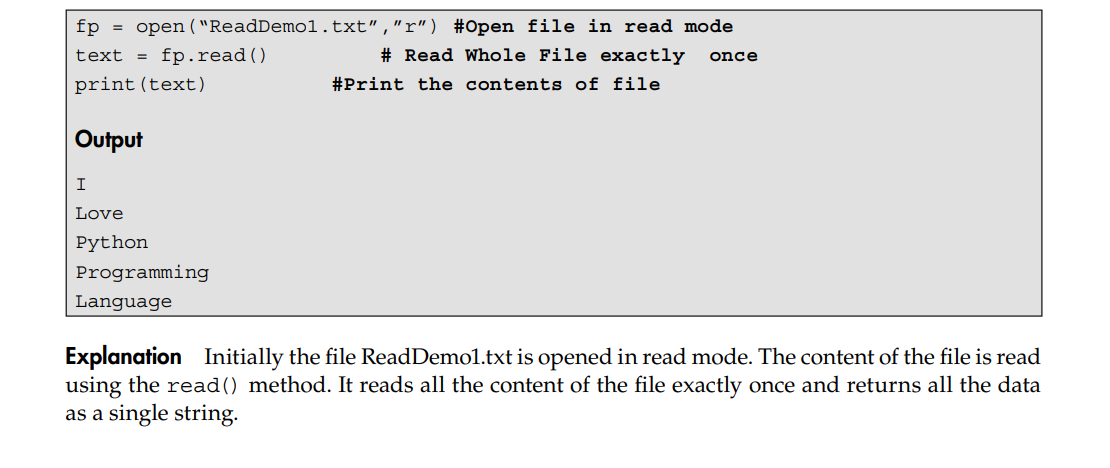
There are several ways to read the content of a file. The two common approaches are:

* + - 1. Use read() method to read all the data from a file and return as one complete string.
      2. Use readlines() method to read all data and return as a list of strings.

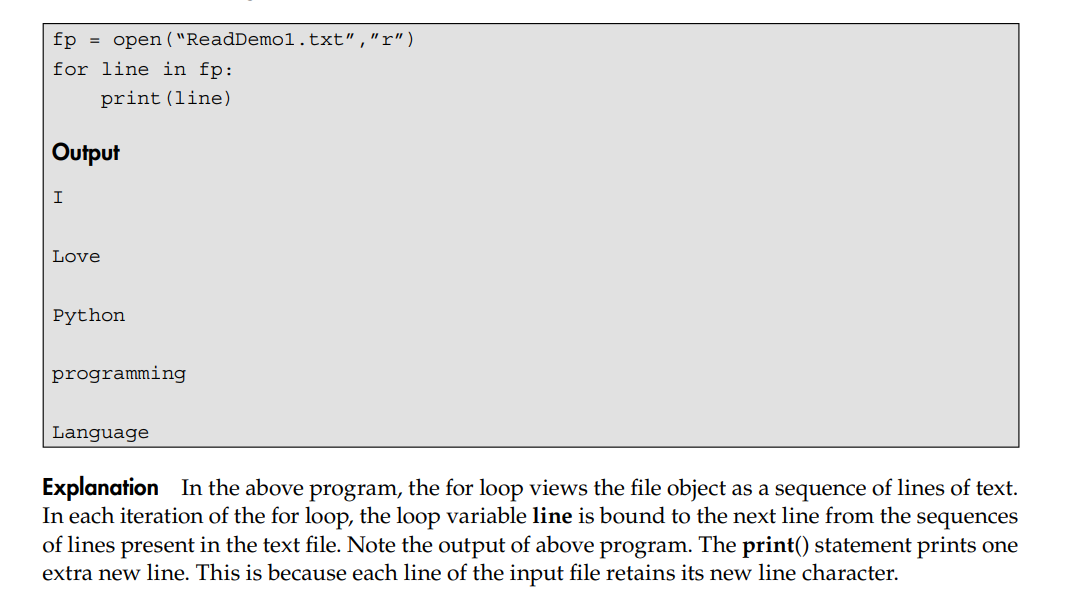
The following program demonstrates the use of the read() method to read the content of the file ReadDemo1.txt. The content of the file is as shown in Figure.



**Program 1:**

****

**Program 2:**

****

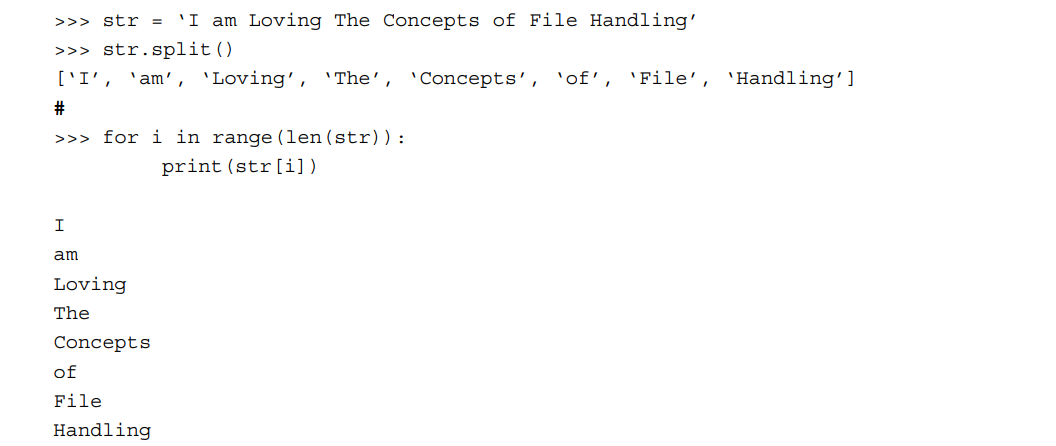
**Q17. Explain reading multiple items on one line.**

**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.

****

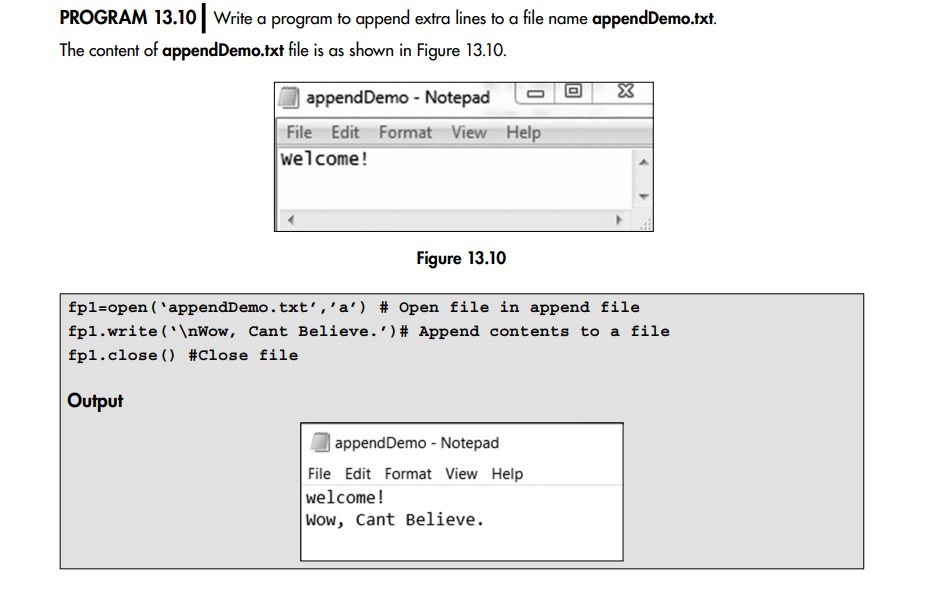
**Explanation** The above example simply splits the string and stores the content to a list. Finally, the for loop is used to access and display each item of the list.

**Q18. Describe the following in short.**

1. **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.

****

1. **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 syntax for 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 value of 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. |

