# 4 Data File Handling

## 4.1 INTRODUCTION

Most of the programs we have seen so far are **transient**, *i.e.*, they run for a short time and produce some output, but when they end, their data disappears. If you run the program again, it starts with a clean slate. This happens because the data entered is executed inside primary memory, RAM, which is volatile (temporary) in nature.

Other programs are **persistent**; they run for a long time (or all the time); they keep at least some of their data in permanent storage (for example, a hard drive); and if they shut down or restart, they execute from the same point. Examples of persistent programs are operating systems, which run whenever a computer is switched on, and web servers, which run all the time, waiting for requests to come in on the network.

One of the simplest ways for programs to maintain their data is by reading and writing text files.

Programs that are used in day-to-day business operations rely extensively on files. Payroll programs keep employee data in files; inventory programs keep data about a company's products in files, accounting systems keep data about a company's financial operations in files, and so on.

Thus, a file is a document or the data stored on a permanent storage device which can be read, written or rewritten according to requirement. In other words, data is packaged up on the storage device as data structures called **Files**. All files are assigned a name that is used for identification purposes by the operating system and the user.

File I/O (Input-Output) means transfer of data from secondary memory (hard disk) to main memory or vice versa. As shown in the figure, when we are working on a computer system, the file or the document stored on the hard disk is brought into RAM (Random Access Memory) for execution and vice versa.

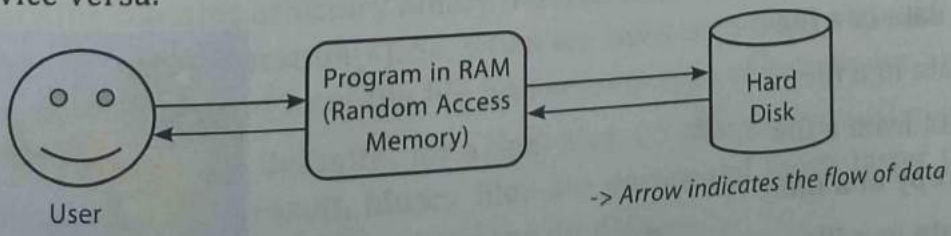



**Fig. 4.1:** Flow of Data

## 4.2 WHY USE FILES

Data used in variables is temporary in nature. Once the application is closed, all the variables and data is lost. So, there comes the concept of files; in order to save our data on some secondary storage device, we have to use files.

Data maintained inside the files is termed as persistent data, *i.e.*, it is permanent in nature. Python allows us to read data from and save data to external text files permanently on secondary storage media, as shown in Fig. 4.2. Apart from this, we permanently store the program data as Python scripts (with .py extension) also.

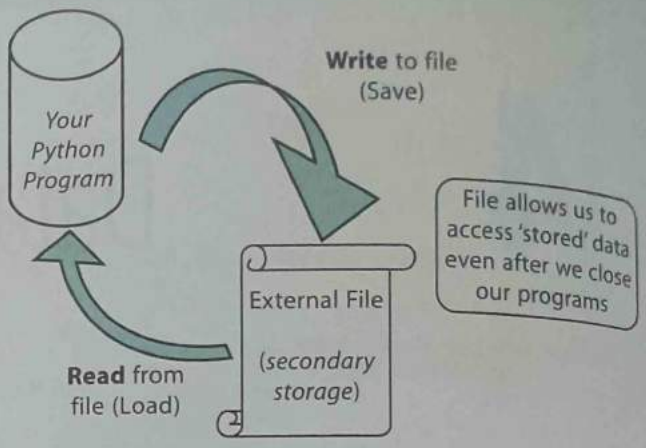



**Fig. 4.2:** Purpose of Using a File

Data is stored using file(s) permanently on secondary storage media. We usually use the files to store data permanently while working and storing data in Word processing applications, spreadsheets, presentation applications, etc. All of these operations require data to be stored in files so that it may be used later.

Thus, files provide a means of communication between the program and the outside world. In a nutshell, a file is a stream of bytes, comprising data of interest.

**CTM:** A file (or data file) is a stream or sequence of characters/data occupying named place on the disk where a sequence of related data is stored. It contains data pertaining to a specific application for later use.

## 4.3 DATA FILE OPERATIONS

Before we start working with a file, first we need to open it. After performing the desirable operation, it needs to be closed so that resources that are tied with the file are freed.

Thus, Python file handling takes place in the following order (Fig. 4.3):

1. Opening a file.
2. Performing operations (read, write, append, insert, delete, etc.) or Processing Data
3. Closing the file.

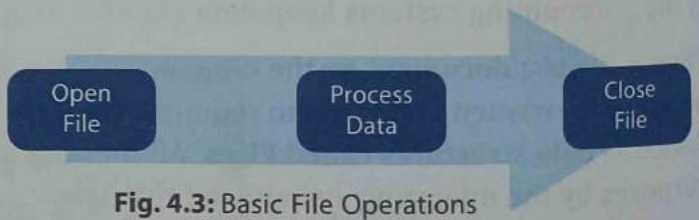
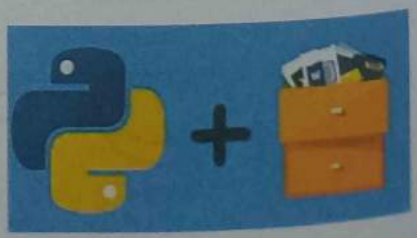



**Fig. 4.3:** Basic File Operations

On the basis of the given basic operations, we can process file in several ways, such as:

- Creating a file
- Traversing a file for displaying data on screen
- Appending data in a file
- Inserting data in a file
- Deleting data from a file
- Creating a copy of a file
- Updating data in a file



In this chapter, we shall be restricting ourselves to three major file operations, *viz.* reading, writing and appending to the file. Rest of the operations are beyond the scope of the syllabus.

### File Types

Before we discuss these file operations, we should be aware of the file types. Computers store every file as a collection of 0s and 1s, *i.e.*, in binary form. Therefore, every file is basically just a series of bytes stored one after the other. Python allows us to create and manage three types of files:

- **Text file**
- **Binary file**
- **CSV file**

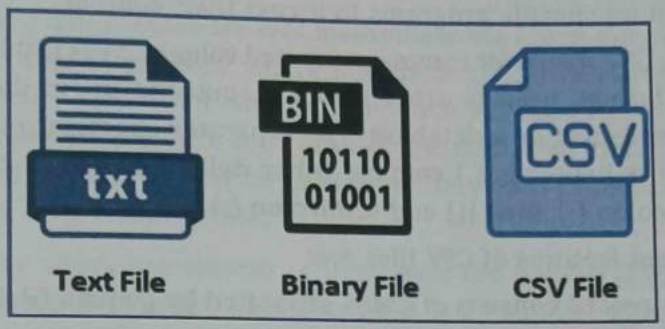



**Fig. 4.4:** Types of Files in Python

1. **Text Files:** A text file consists of a sequence of lines. A line is a sequence of characters (ASCII or UNICODE) which usually comprises alphabets, numbers and other special symbols stored on permanent storage media. Text file stores information in the form of **ASCII** or **UNICODE** character. Although default character coding in Python is ASCII, using the constant 'u' with string, it supports Unicode as well. In a text file, each line is terminated by a special character, known as **End of Line (EOL)**. By default, this EOL character is the newline character ('\n'). So, at the lowest level, text file will be a collection of bytes. In text file, some translation takes place when this EOL character is read or written. Text files are stored as a sequence of bytes in human-readable form and can be created using any text editor.

   We use text files to store character data. For example, test.txt.

   Other examples of text files are:

   - **Document Files:** such as .txt, .rtf
   - **Tabular Files:** such as .csv, .tsv
   - **Source Code Files:** such as .py, .js, .c, .app, .java
   - **Web Standard Files:** such as .html, .xml, .css, .json
   - **Configuration Files:** such as .ini, .cfg, .reg

2. **Binary Files:** Binary files are used to store binary data such as images, video files, audio files, etc. A binary file contains arbitrary binary data, usually numbers stored in the file which can be used for numerical operation(s). So, when we work on a binary file, we have to interpret the raw bit pattern(s) read from the file into correct type of data in our program.

   **In binary file**, there is no delimiter for a line. Also, no character translations can be carried out in a binary file. As a result, binary files are easier and much faster than text files for carrying out reading and writing operations on data.

It is perfectly possible to interpret a stream of bytes originally written as string as numeric value. But that will be an incorrect interpretation of data and we may not get the desired output or might get some garbage value after the file processing activity. So, in the case of a binary file, it is extremely important that we interpret the correct data type while reading the file. Python provides special module(s) for encoding and decoding of data for binary file.

Thus, a text file consists of human-readable characters which can be opened by any text editor. On the other hand, binary files are made up of non-human readable characters and symbols, which require specific programs to access their content.

3. **CSV Files:** CSV stands for comma-separated values. CSV is just like a text file, *i.e.*, it is in human-readable format, used to store data in tabular form (in the form of rows and columns) like a spreadsheet or a database. The separator character of CSV files is called a delimiter. Its default delimiter is (,) comma. Other delimiters which can be used with CSV files are tab ('\t'), colon (:), pipe (|) and semicolon (;).

   Some salient features of CSV files are:

   - Each record consists of fields separated by comma (delimiter).
   - It is the most preferred import and export format for databases and spreadsheets.
   - Each line in a CSV file is treated as a record.

## 4.4 OPENING AND CLOSING FILES

To handle data files in Python, we need to have a *file variable or file object or file handle*. It is the object that gets associated with a particular file to perform read/write operations at a particular time.

Object can be created by using open() function. To work on a file, the first thing we do is open it. This is done by using the built-in function open(). Using this function, a file object is created which is then used for accessing various methods and functions available for file manipulation.

### 4.4.1 open()—Opening a File

When we want to read or write a file (say on the hard drive), we must first open the file. Opening the file communicates with the operating system, which knows where the data for each file is stored. When we open a file, we are asking the operating system to find the file by name and make sure the file exists. In Example 1, we open the file test.txt, which should be stored in the same folder that we are in when we start Python.

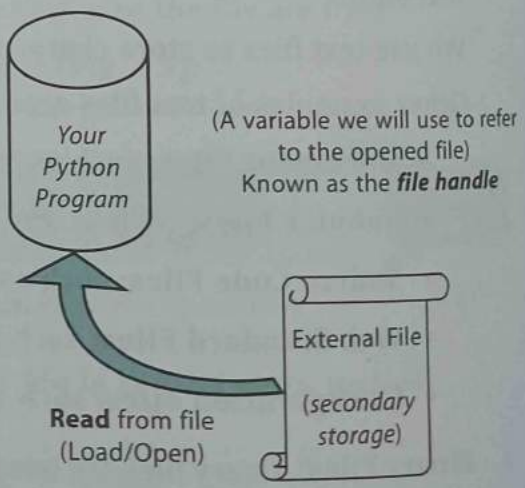



**Fig. 4.5:** Opening a File

**open()** function takes the name of the file as the first argument. The second argument indicates the mode of accessing the file. The syntax for open() is:

***Syntax:***

```
<file variable>/<file object or handle> = open(file_name, access_mode)
```

Here, the first argument with open() is the name of the file to be opened and the second argument describes the mode, *i.e.*, how the file will be used throughout the program. This is an optional parameter as the default mode is the **read** mode (reading).

The file_object establishes a link between the program and the data file stored in the permanent storage, and access modes define the location of the file pointer, *i.e.*, from where the data has to be read and written to the file.

Modes for opening a file:

- read(r): to read the file
- write(w): to write to the file
- append(a): to write at the end of the file

The object of file type is returned using which we will manipulate the file in our program. When we work with file(s), a buffer (area in memory where data is temporarily stored before being written to the file) is automatically associated with the file when we open it. While writing the content to a file, first it goes to buffer, and once the buffer is full, data is written to the file. Also, when the file is closed, any unsaved data is transferred to the file. **flush()** function of file object is used to force transfer of data from buffer to file.

If the opening is successful, the operating system returns us a file handle (Fig. 4.6). The file handle is not the actual data contained in the file; instead, it is a "handle" that we can use to read the data. We are given a handle if the requested file exists and we have proper permission to read the file.

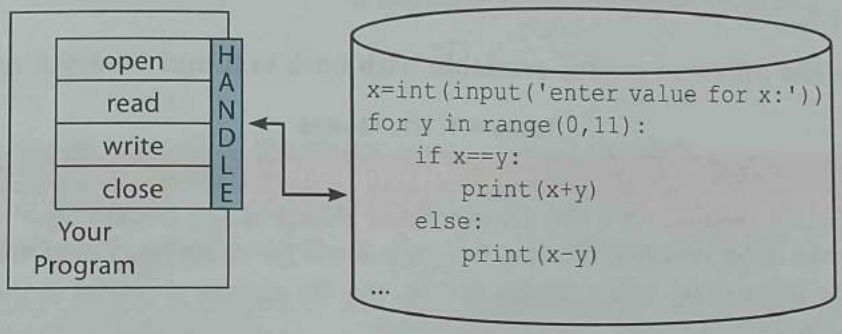



**Fig. 4.6:** A File Handle

**Example 1:**
```
>>> f = open('test.txt')
>>> print(f)
```

As is evident from the above example, when we open a file, it opens in read mode by default and 'f' as the file object or file variable or file handle, whatever we may say, shall be returned as the output by the operating system.

```
========== RESTART: Shell ==========
>>> f =open('test.txt')
>>> print(f)
<_io.TextIOWrapper name='test.txt' mode='r' encoding='cp1252'>
>>>
```
Ln: 25 Col: 4

**POINT TO REMEMBER**

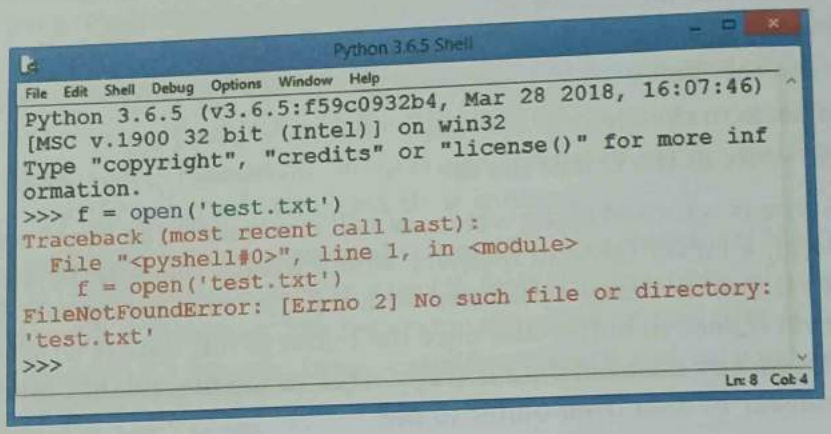
Always create the file in the same default folder where your Python has been installed.

**Note:** When you open a file in read mode, the given file must exist in the folder, otherwise Python will raise **FileNotFoundError**.

We shall be discussing the file paths in detail later in the chapter.

Now, in the other situation in which the file does not exist, open() will fail with a trace back and we will not get a handle to access the contents of the file, as shown in the example given below:

**Example 2:**




### File Modes

The second parameter of the open() function corresponds to a mode which is either read ('r'), write ('w'), or append ('a'). The file mode defines how the file will be accessed. In Python, files are opened in text mode by default. To open file in binary mode, it should be suffixed with 'b' to represent binary files.

Table 4.1 explains the different modes available with both text and binary files.

**Table 4.1: File Modes**

| Mode | Description |
|---|---|
| r | Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode. If the specified file does not exist, it will generate FileNotFoundError. |
| rb | Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode. |
| r+ | Opens a file for both reading and writing. (+) The file pointer will be at the beginning of the file. |
| rb+ | Opens a file for both reading and writing in binary format. The file pointer will be at the beginning of the file. |
| w | Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, it creates a new file for writing. |
| wb | Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing. |
| w+ | Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing. |
| wb+ | Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing. |
| a | Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing. |
| ab | Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing. |
| a+ | Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing. |
| ab+ | Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing. |

**POINT TO REMEMBER**

The default file-open mode is read mode. If you do not provide any file-open mode, Python will open it in read mode ("r").

**Example 3:**

```
>>> file = open("test.txt", "r+")
```

will open the file 'test.txt' for reading and writing purpose. Here, the name (by which it exists on secondary storage media) of the file specified is constant.

We can use a variable instead of a constant as name of the file, test file; if it already exists, then it has to be in the same folder where we are working now, otherwise we have to specify the complete path. It is not mandatory to have file name with extension. In the above example, .txt extension is used for our convenience of identification as it is easy to identify the file as a text file. Similarly, for binary file we will use .dat extension.

### 4.4.2 close()—Closing a File

The close() method of a file object flushes any unwritten information and closes the file object, after which no more writing can be done. Python automatically closes a file when the reference object of a file is reassigned to another file. It is a good practice to use the close() method to close a file.

**Syntax:**

```
fileObject.close()
```

A close() function breaks the link of file object and the file on the disk.

After closing a file (using close()), no tasks can be performed on that file through the file object.

**Example 4:**

```
>>> f =open('test.txt')
>>> print("The name of the file to be closed is:",f.name)
The name of the file to be closed is: test.txt
>>> f.close()
>>>
```

Ln: 29 Col: 4

In the above example, we have used 'name' property of the file object 'f' along with print() statement, which will return the name of the currently used file, which is 'test.txt' in this case.

The above program can be alternatively performed in case the file is not present in the Python root directory.

Let us assume that the text file is present in the D:\ drive. Then, the open() method shall be defined as follows:

```
#Checking for an existing file other than Python root directory
f=open("D:\\test.txt","r") #accessing file with double slash if stored on
                           some other drive
print(f.read()) #read() is used for reading the data present in the file
```

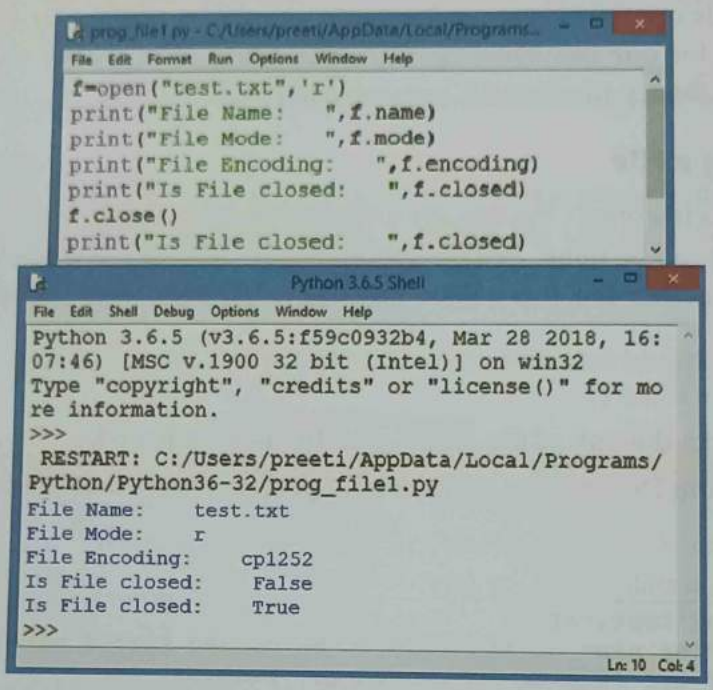
**Note:** A file always gets opened in read mode and as text file by default.

Let us discuss these properties of File Object.

**Various properties of File Object:**

Once open() is successful and file object gets created, we can retrieve various details related to that file using its associated properties.

1. name: Name of the opened file.
2. mode: Mode in which the file gets opened.
3. closed: returns Boolean value, which indicates whether the file is closed or not.
4. encoding: returns the encoding format of the file. (Microsoft Windows OS uses 'cp1252' encoding system by default.)




## 4.5 READING FROM A FILE

Python provides various methods associated with a fileObject for reading data from a file. We can read character data from text file by using the following read methods:

a) read(): To read the entire data from the file; starts reading from the cursor (from the beginning of the file) up to the end of the file.

b) read(n): To read 'n' characters from the file, starting from the cursor; if the file holds fewer than 'n' characters, it will read until the end of the file. Also, if the size is missing or has a negative value, then the entire file is read.

c) readline(): To read only one line from the file; starts reading from the cursor up to, and including, the end of line character.

d) readlines(): To read all lines from the file into a list; starts reading from the cursor up to the end of the file and returns a list of strings, separated by new line character '\n'.

Let us understand these methods with the help of suitable examples using a text file 'test.txt'.

1) **read()/read(size):** read() can be used to read specific-size strings from a file. This function also returns a string read from the file. At the end of the file, again an empty string will be returned.

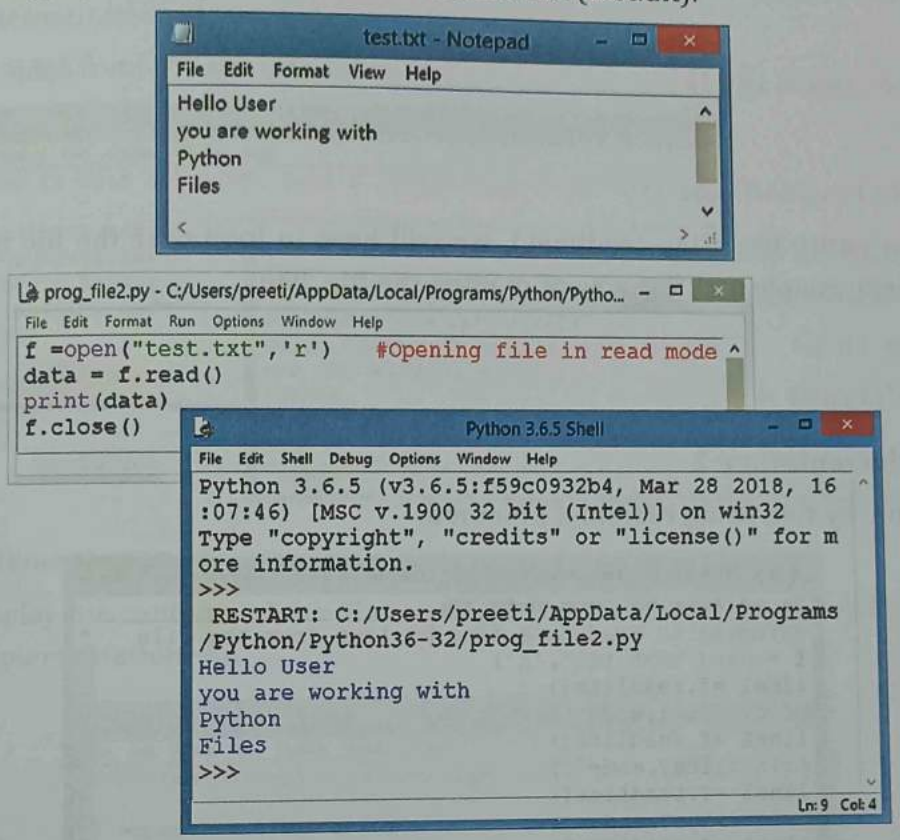
**Syntax** of read() function is:

```
fileObject.read() or
fileObject.read([size])
```

Here, size specifies the number of bytes to be read from the file. So, the function may be used to read a specific number of characters from the file. If the value of size is not provided or a negative value is specified as size, then the entire file will be read. One must take care of the memory size available before reading the entire content from the file.

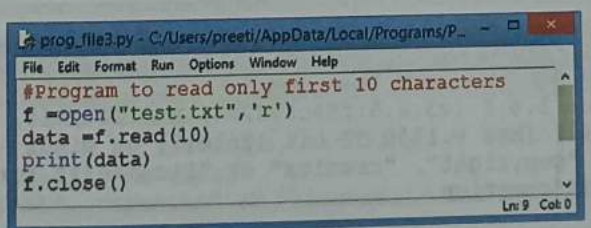
### Practical Implementation–1

To illustrate read() by reading the entire data from a file (test.txt).




### Practical Implementation–2

To illustrate read(n) by reading only the first 10 characters from the file (test.txt).




**Output:**

```
>>>
 RESTART: C:/Users/preeti/AppData/Local/Programs
/Python/Python36-32/prog_file3.py
Hello User
>>>
```

2) **readline():** readline() function will return a line read, as a string from the file. First call to function will return the first line, second call the next line, and so on. We must remember that file object keeps track from where reading/writing of data should happen. For readline(); a line is terminated by '\n' (*i.e.* new line character). The new line character is also read from the file and post-fixed in the string. When end of file is reached, readline() will return an empty string. The syntax for readline() is:

**Syntax:**

```
fileObject.readline()
```

Since the method returns a string, it should be returned and stored using a variable as shown below:

```
>>> x = file.readline()
```

or

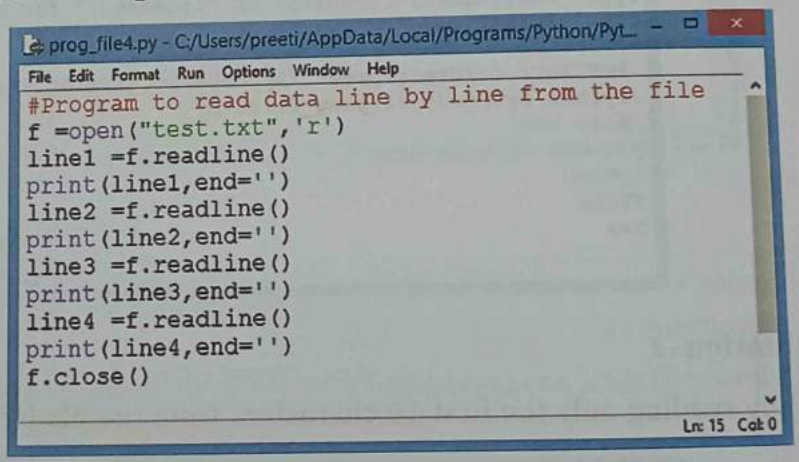
```
>>> print(file.readline())
```

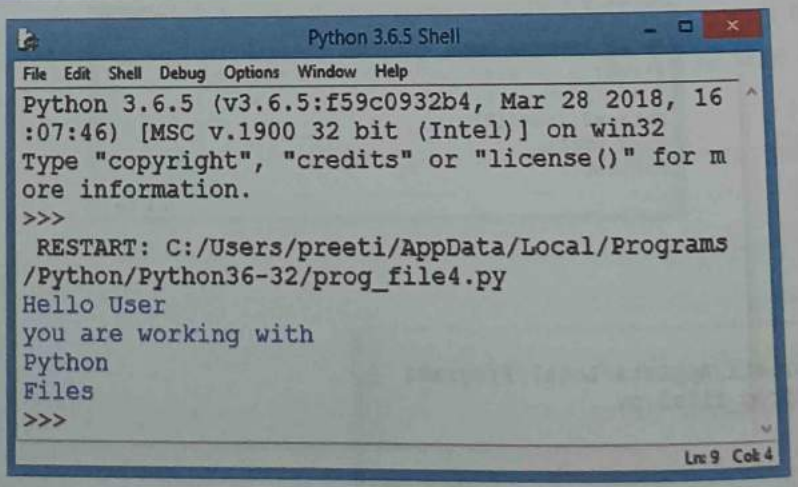
For reading an entire file using readline(), we will have to loop over the file object. This is a memory efficient, simple and fast way of reading the file, like:

```
>>> for line in x:
...     print(line)
```

### Practical Implementation–3

To read data line by line using readline() method.






3) **readlines()**: readlines() function can be used to read the entire content of the file. This method will print all the lines present in the file. It will start reading the data line-wise from the beginning of the file and shall display the contents of the file till the end of file character.

We need to be careful while using it with respect to the size of memory required before using the function. The method will return a list of strings, each separated by new line character '\n'. The syntax is:

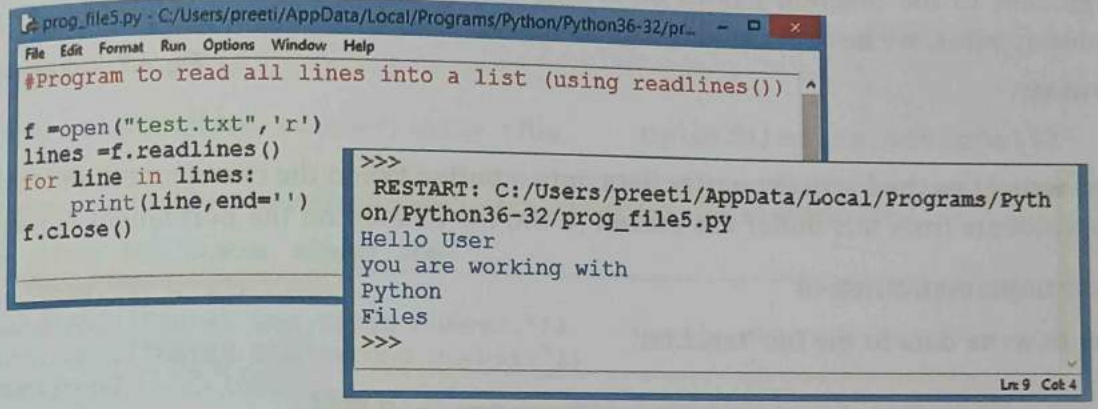
**Syntax:**

```
fileObject.readlines()
```

as it returns a list, which can then be used for manipulation.

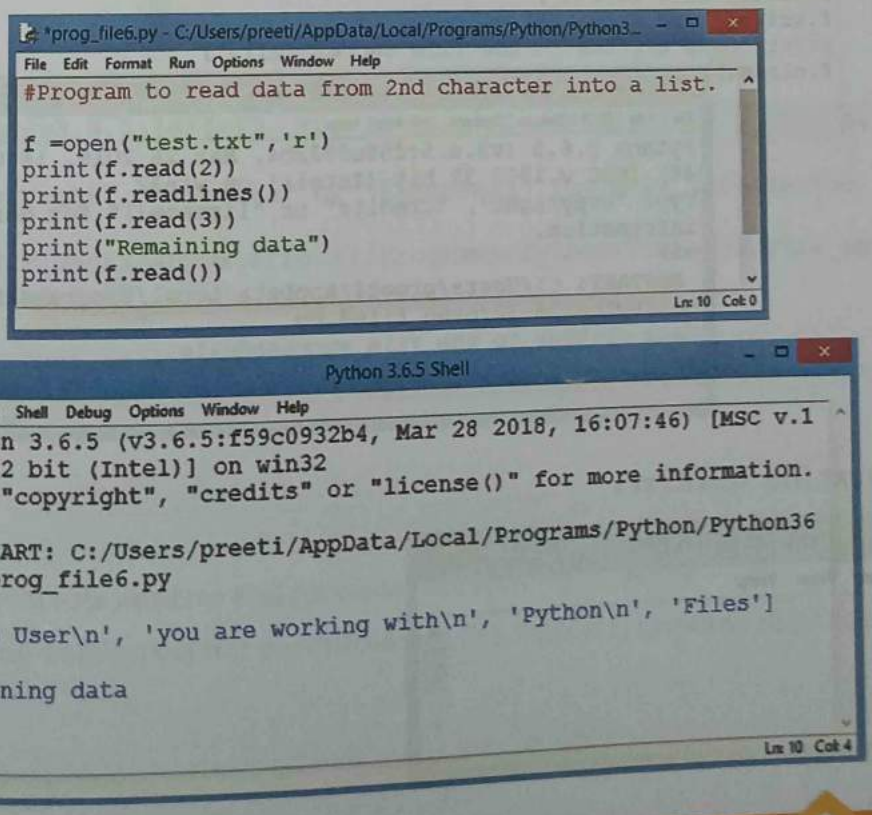
### Practical Implementation–4

To read all the lines from the file into list.




### Practical Implementation–5

Program to display the contents of the file starting from 2nd character into the list. (Modification of Practical Implementation–4).



## 4.6 WRITING TO FILE

We can write character data into a file in Python by using the following two file object methods:

1. write(string)
2. writelines(sequence of lines)

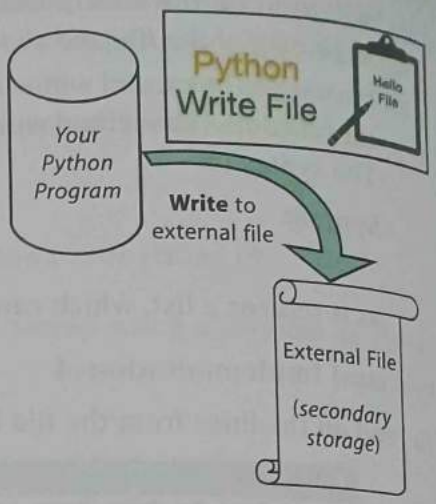



**Fig. 4.7:** Writing a File

1. **write()**: write() method takes a string (as parameter) and writes it in the text file in a single line. For storing data with end of line character, we will have to add '\n' character to the end of the string. Notice the addition of '\n' at the end of every sentence while talking of data. txt ('\n' is treated as special characters of 2 bytes). As argument to the function has to be string, for storing numeric value, we have to convert it to string.

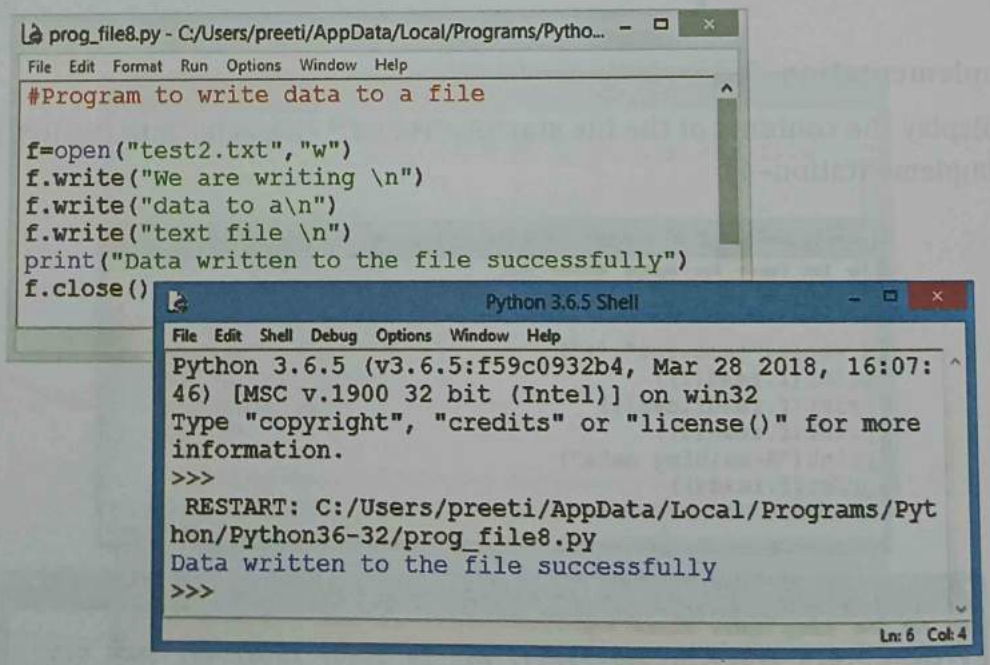
**Syntax:**

```
fileObject.write(string)
```

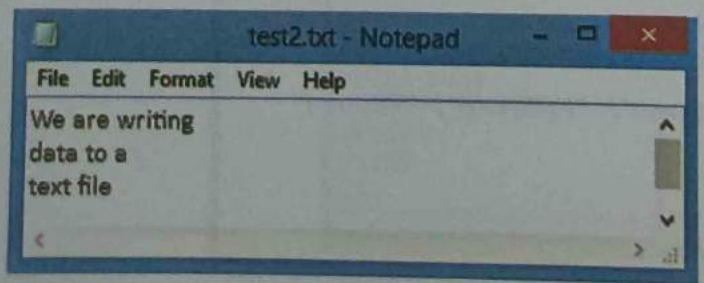
The write() method actually writes data onto a buffer. When the close() method is executed, the contents from this buffer are moved to the file located on the permanent storage.

### Practical Implementation–6

Program to write data to the file 'test2.txt'.




**Contents of text file 'test2.txt':**



In the given program, 'test2.txt' file, does not exist in the current directory, so the new file is created. If the file exists, data present in the file shall be overridden (or overwritten) every time we run this program.

In order to avoid this situation, we should add new text (append) to already-existing text using "a", append mode, which we shall discuss in the next section.

**For storing numeric data value in a text file, conversion to string is required.**

**Example 5:**

```
>>> file=open("num.txt",'w')
>>> x = 52
>>> file.write(str(x))
>>> file.close()
```

**Example 6:**

To perform addition of two numbers using a file.

*File_write.py - C:/Users/User2/AppData/Local/Programs/Python/Python39/File_write.py (3.9....

File Edit Format Run Options Window Help

```
f=open("fnew.txt",'w')
num1=int(input("Enter the first number:"))
num2=int(input("Enter the second number:"))
sum1=num1+num2
f.write(num1)
f.write(num2)
f.write(sum1)
f.close()
```

Ln: 9 Col: 0

IDLE Shell 3.9.7

File Edit Shell Debug Options Window Help

```
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit (AM
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/User2/AppData/Local/Programs/Python/Python39/File_write.py =
Enter the first number:55
Enter the second number:77
Traceback (most recent call last):
  File "C:/Users/User2/AppData/Local/Programs/Python/Python39/File_write.py", li
ne 5, in <module>
    f.write(num1)
TypeError: write() argument must be str, not int
```

Ln: 11 Col: 4

The above code will generate an error and no values are written to the file.

This is because writing numbers is not permitted to a file, thus it is required to be converted into a string first.

**To perform addition of two numbers using a file (Modification of Example 6).**

```
*File_write.py - C:/Users/User2/AppData/Local/Programs/Python/Python39/File_write.py (3.9....
File Edit Format Run Options Window Help
f=open("fnew.txt",'w')
num1=int(input("Enter the first number:"))
num2=int(input("Enter the second number:"))
sum1=num1+num2
f.write("\n Number 1:"+str(num1))
f.write("\n Number 2:"+str(num2))
f.write("\n Sum of two numbers:"+str(sum1))
f.close()                                                    Ln: 9 Col: 0
```

```
IDLE Shell 3.9.7
File Edit Shell Debug Options Window Help
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC
.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more in
rmation.
>>>
= RESTART: C:/Users/User2/AppData/Local/Programs/Python/Python
/File_write.py =
Enter the first number:55
Enter the second number:77
>>>                                                          Ln: 7
```

The content of the file will be:

```
*fnew.txt - C:\Users\User2\AppData\Local\Programs\Python\Pyt...
File Edit Format Run Options Window Help
Number1 :55
Number 2 :77
Sum of two numbers:132
                                                             Ln: 9 Col: 1
```

In the above program '+' operator will concatenate the two strings and str() method is used for type conversion from an integer to a string.

### Practical Implementation–7

Program to write numeric data to a text file.

```
prog_file9.py - C:/Users/preeti/AppData/Local/Programs/Python/Python36-32/pro...
File Edit Format Run Options Window Help
#Adding numeric to a file using conversion function- str()

f =open("newtest.txt","w")
x =100
f.write("Hello World \n")
f.write(str(x))    #Numeric value is converted into string
f.close()
```

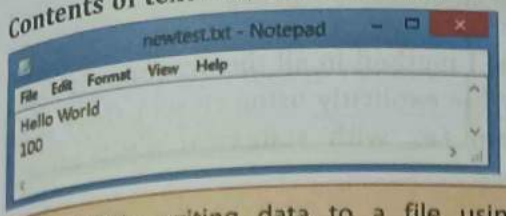
```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MS
C v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more informa
tion.
>>>
 RESTART: C:/Users/preeti/AppData/Local/Programs/Python/Pyt
hon36-32/prog_file9.py
>>>
```

Blinking Cursor indicates the successful write operation in the File

Ln: 5 Col: 4

In the above example, after the program gets executed and the data (both string as well as numeric) gets written to the file, the blinking cursor in the Python shell indicates that the data has been successfully written onto the file 'newtest.txt'.

**Contents of text file, 'newtest.txt':**




**CTM:** While writing data to a file using the write() method, we must provide line separator ('\n'-new line feed), otherwise the entire data to be written shall be written to a single line.

2. **writelines():** For writing a string at a time, we use write() method; it can't be used for writing a list, tuple, etc., into a file. Sequence data type including multiple strings at the same time can be written using writelines() method in the text file.

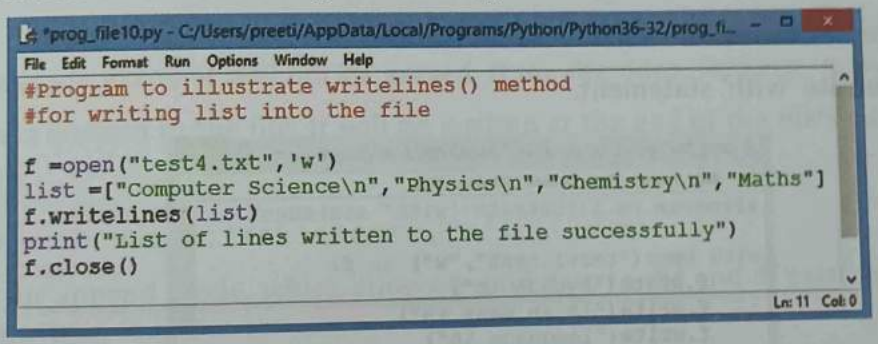
   **Syntax:**

   ```
   fileObject.writelines(sequence)
   ```

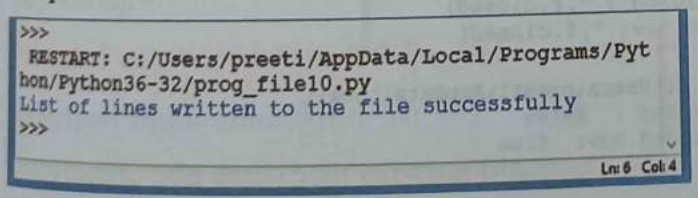
   So, whenever we have to write a sequence of string/data type, we must use writelines() instead of write().

### Practical Implementation–8

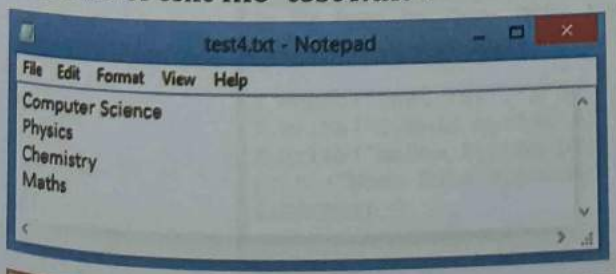
Program to add list items to a file using writelines() method.




**Output:**




**Contents of text file 'test4.txt':**




**CTM:** While reading from or writing to the file, the cursor always starts from the beginning of the file.

Also to be noted here is that writelines() method does not add any EOL character to the end of string. We have to do it ourselves. So, to resolve this problem, we have used '\n' new line character (in the program) after the end of each list item or string.

**CTM:** writelines() method accepts any type of sequence as an argument.

You must have noticed that till now we have used close() method in all the programs to close the file in the end. In case you don't want to close your file explicitly using close(), there is an alternative statement which can be used in the program, *i.e.*, 'with' statement which we will discuss now.

### with Statement

Apart from using open() function for creation of file, **with statement** can also be used for the same purpose. We can use this statement to group file operation statements within block to make the code compact and much more readable. This method is quite handy when you have two related operations which you would like to execute as a pair, with blocks of codes in between.

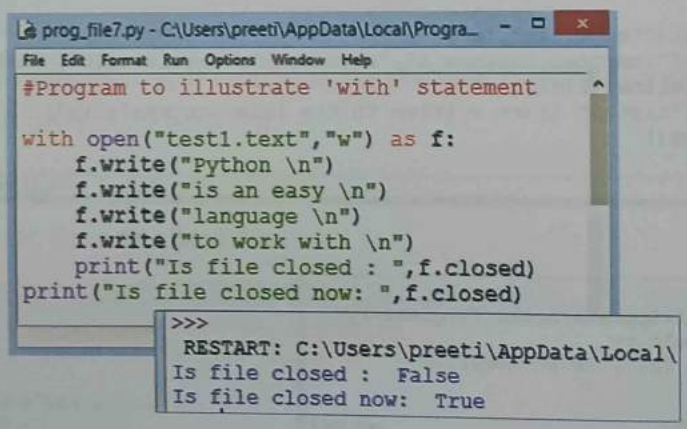
Using **with** ensures that all the resources allocated to the file objects get deallocated automatically once we stop using the file. In the case of exceptions also, we are not required to close the file explicitly using **with** statement. Its syntax is:

**Syntax:**

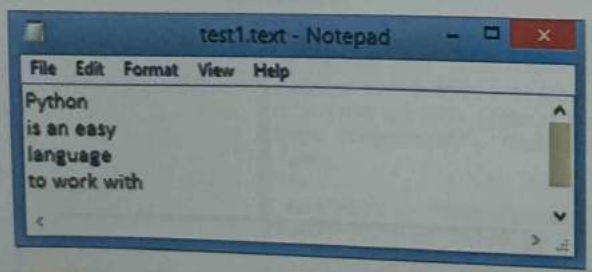
```
with open("filename","mode") as fileObject:
    file manipulation statements
```

### Practical Implementation–9

Program to illustrate 'with' statement.




**Contents of text file "test1.txt":**



Thus, the advantages of using with statement are:

1. It handles all the exceptions occurring before the end of the block (which generally are: End of the file error and stack underflow).
2. It automatically closes the file and releases the resources allocated to the file.

Therefore, based on the concepts learnt so far, certain important points should be kept in mind while working with files in Python.

**POINTS TO REMEMBER**

1. For storing data with EOL character, we have to add the character to the end of the string.
2. In case of storing numeric value, we have to either convert it into a string using str() method or write it in quotes.
3. We cannot write numeric data directly into the file.
4. If we don't use the close() method, it will not write anything into the file as without closing the file, the resources are not released.
5. If we write more data to an existing file, it will overwrite the previous data. Thus, the entire file will be overwritten.

## 4.7 APPENDING TO FILE

Append means 'to add to'; so if we want to add more data to a file which already has some data in it, we will be appending data. In such a case, use the access mode '**a**', which means:

***'Open for writing, and if it exists, then append data to the end of the file'.***

In Python, we can use the 'a' mode to open an output file in append mode. This means that:

- If the file already exists, it will not be erased. If the file does not exist, it will be created.
- When data is written to the file, it will be written at the end of the file's current contents.

**Syntax:**

```
<file_object> = open(<filename>,'a')
```

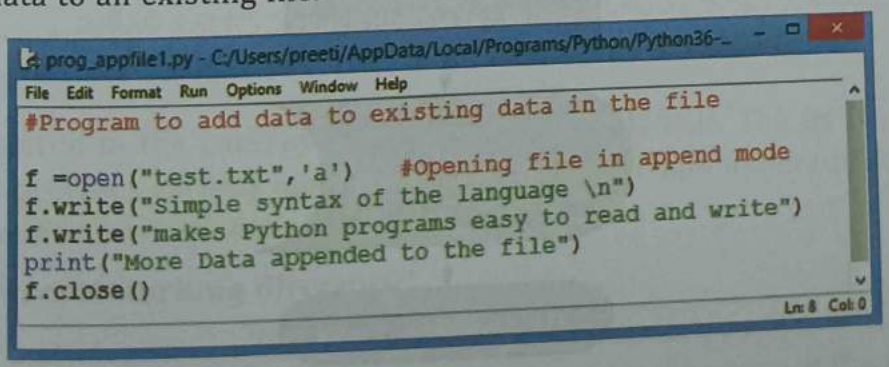
Here, 'a' stands for append mode, which allows to add data to the end of existing data instead of overwriting in the file.

For example,

```
>>> f = open("test1.txt", "a")
```

### Practical Implementation–10

Program to add data to an existing file.

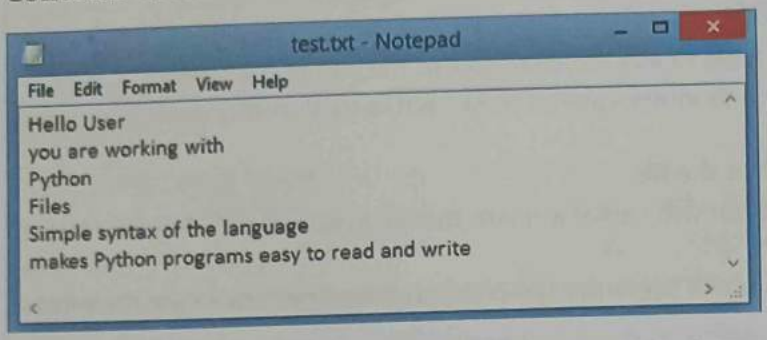


**Output:**

```
>>>
 RESTART: C:/Users/preeti/AppData/Local/Programs/Python/Python3
6-32/prog_appfile1.py
More Data appended to the file
>>>
```

Ln: 9 Col: 4

**Contents of text file "test.txt":**

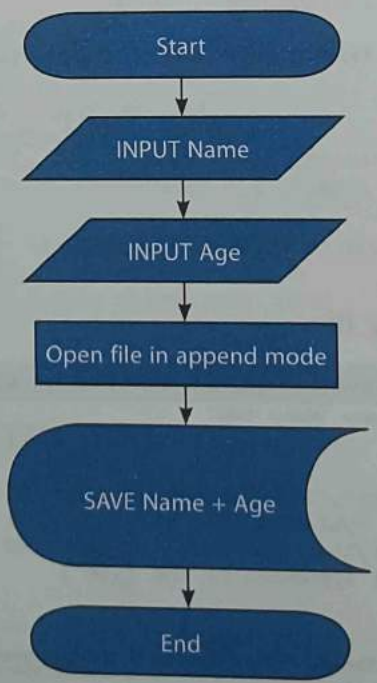


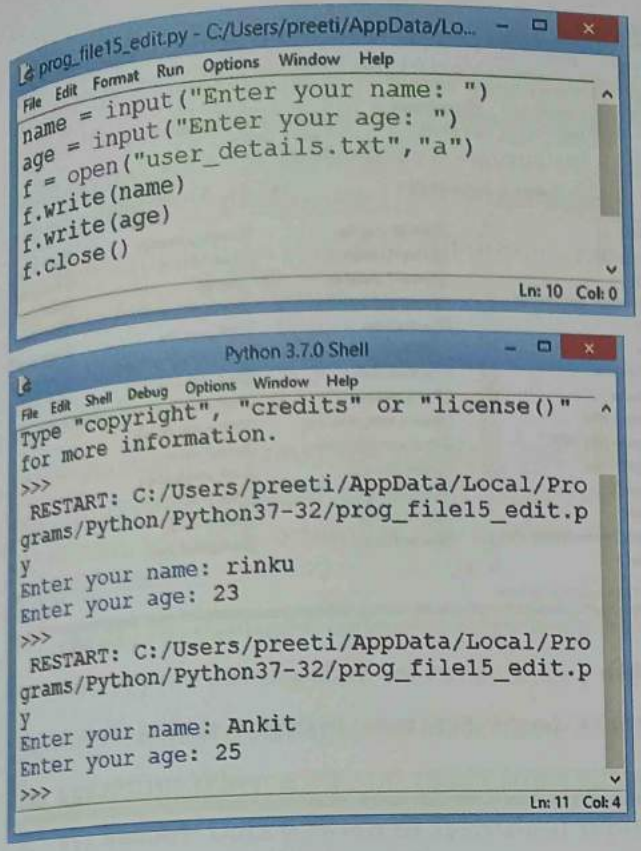

While writing a program with the help of a flow chart, you can store data inside a file using a special symbol:

This symbol is used to represent the storing of data in a flow chart. Let's take an example to understand it.

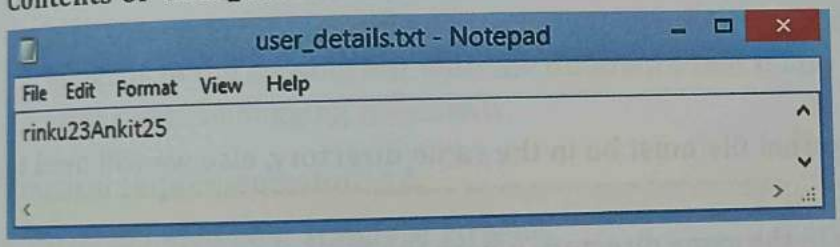
### Practical Implementation–11

Design and code a program that asks the user to input their name along with age and store the data in a text file.






**Contents of 'user_details.txt':**




## 4.8 RELATIVE AND ABSOLUTE PATHS

Files are organized into directories (also called "folders"). Every running program has a "current directory," which is the default directory for most operations. For example, while opening a file for reading, Python looks for it in the current directory.

The os module provides functions for working with files and directories ("os" stands for "operating system"). **os.getcwd** returns the name of the current directory:

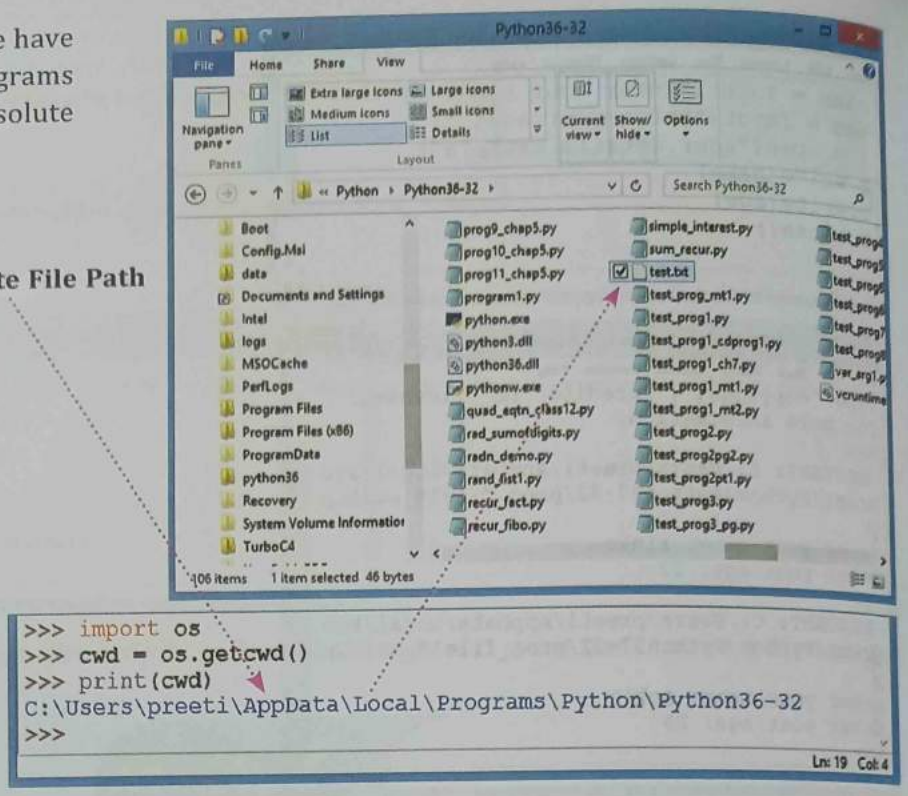
```
>>> import os
>>> cwd = os.getcwd()
>>> print(cwd)
```

Files are always stored in the current folder/directory by default. The **os** (Operating System) module of Python provides various methods to work with file and folder/directories. For using these functions, we have to import os module in our program.

**cwd** stands for **"current working directory"**.

A string like cwd that identifies a file is called a **path**. A **relative path** starts from the current directory, whereas an **absolute path** starts from the topmost directory in the file system.

For example, the text file we have created in the previous programs was opened through the absolute path.




Alternatively,

```
>>> f = open("test.txt")
```

\\test.txt is the Relative File Path

**Note:** The Python program and external file must be in the same directory, else we will need to enter the entire file path.

To create a text file manually in the same directory where the Python resides, follow these steps:

1. Select File tab from the Python Shell -> New File.
2. Type the data -> select Save.
3. Type file name -> select Save as type option and select the text files (.txt).
4. Click Save button.

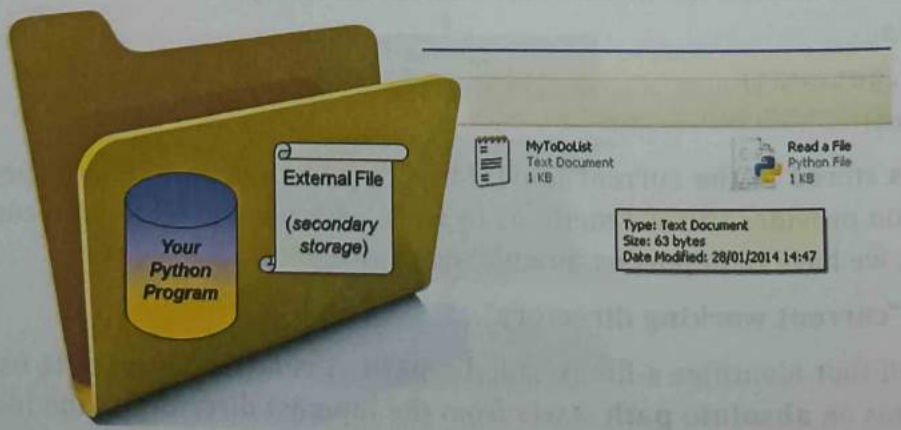



**Fig. 4.8:** Relative File Path

## 4.9 STANDARD FILE STREAMS

We use file object(s) to work with data file; similarly input/output from standard I/O devices is also performed using standard I/O stream object. Since we use high-level functions for performing input/output through keyboard and monitor, such as eval(), input() and print statement, we are not required to explicitly use I/O stream object.

The standard streams available in Python are:

- Standard input stream,
- Standard output stream, and
- Standard error stream.

These standard streams are nothing but file objects, which get automatically connected to your program's standard device(s) when we start Python. In order to work with standard I/O stream, we need to **import sys** module. The methods which are available for I/O operations in it are:

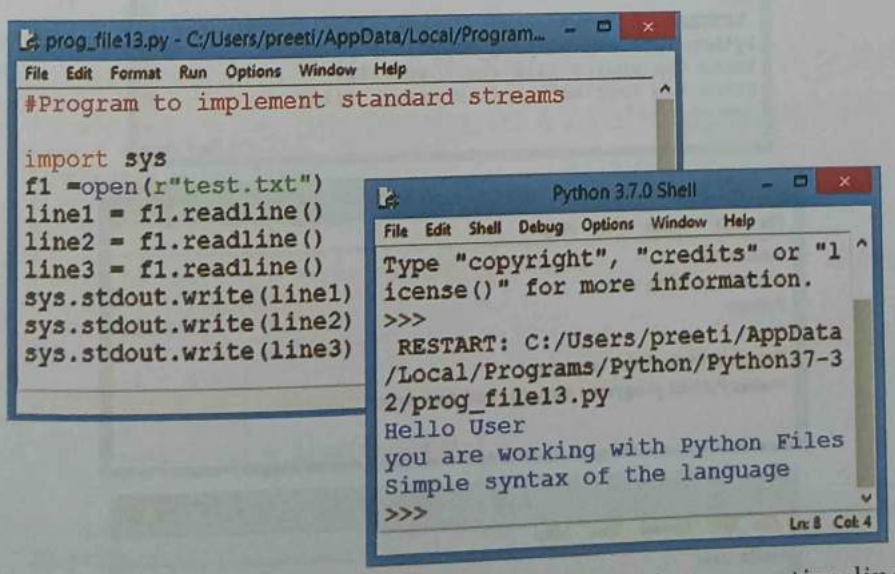
- **read()** for reading a byte at a time from keyboard.
- **write()** for writing data on console, *i.e.*, monitor.

The three standard streams are described as follows:

1. **sys.stdin:** When a stream reads from standard input.
2. **sys.stdout:** Data written to sys.stdout typically appears on your screen, but can be linked to the standard input of another program with a pipe.
3. **sys.stderr:** Error messages are written to sys.stderr. It is similar to sys.stdout as it also prints directly to the console but with the difference that it also prints exceptions, error messages along with debugging comments.

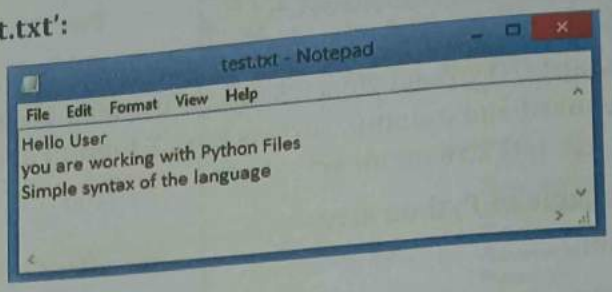
### Practical Implementation–12

Program to implement standard streams.



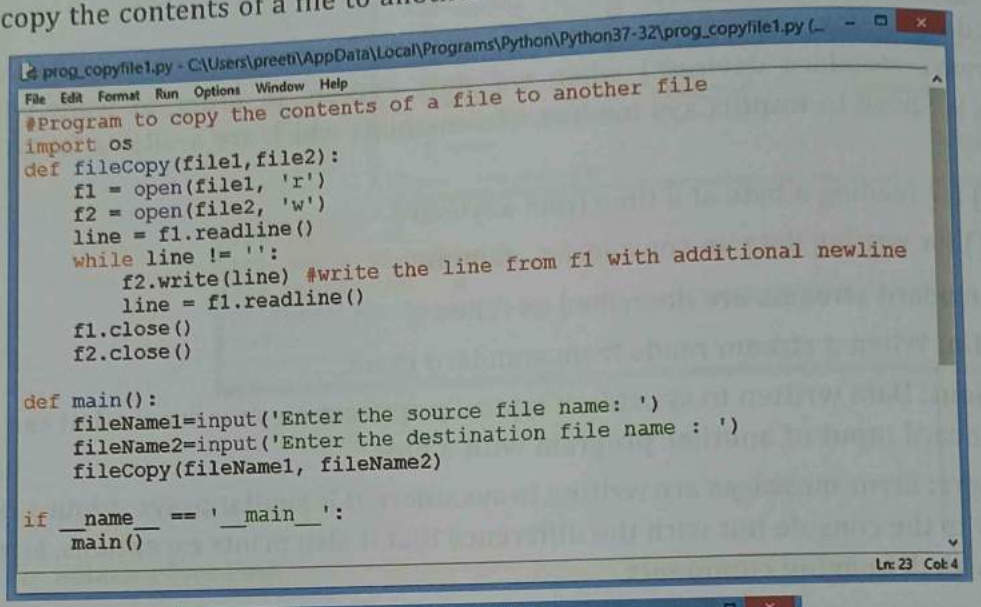

The lines containing the method stdout.write() shall write the respective lines (from the file 'test.txt') on device/file associated with sys.stdout, which is the monitor.

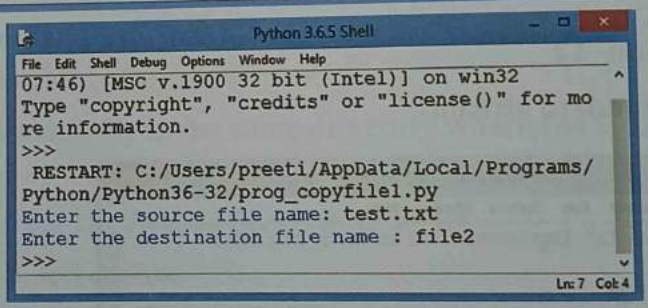
**Contents of the file 'test.txt':**

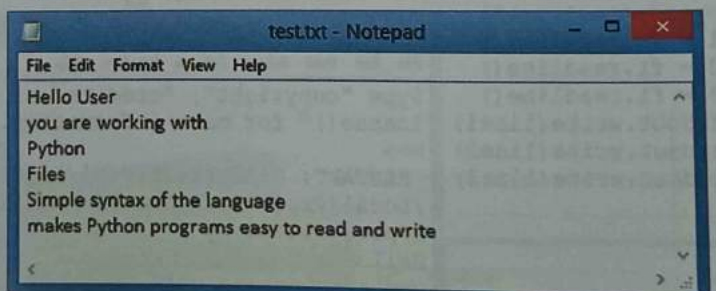


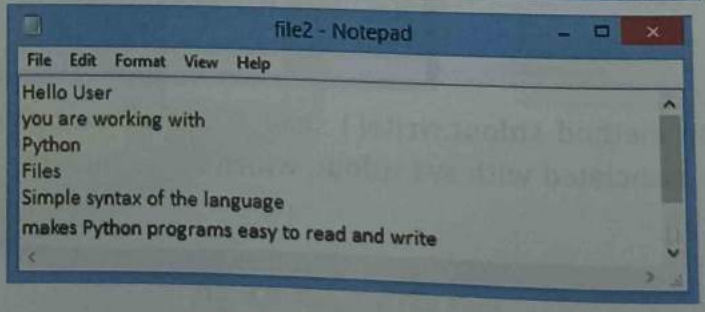

## Practical Implementation–13

Program to copy the contents of a file to another file.



**Explanation:**

In the given program, filecopy() function takes two file names as parameters. file1 is opened in read mode and file 2 is opened in write mode. The lines from the file "file1" are read using readline() and while loop is used to read all the contents of the file until blank space ' ' is found and finally both the files are closed explicitly.

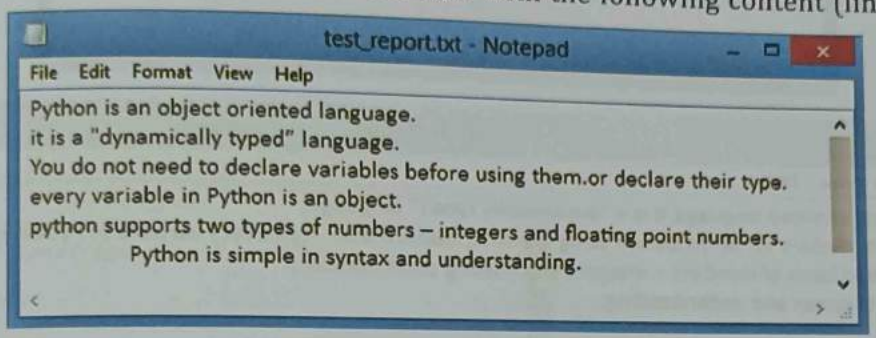
The main() function asks the user to input the source file and the destination file and passes these files as arguments to the filecopy() function.

As is evident from the given output, the contents of both the files are same.

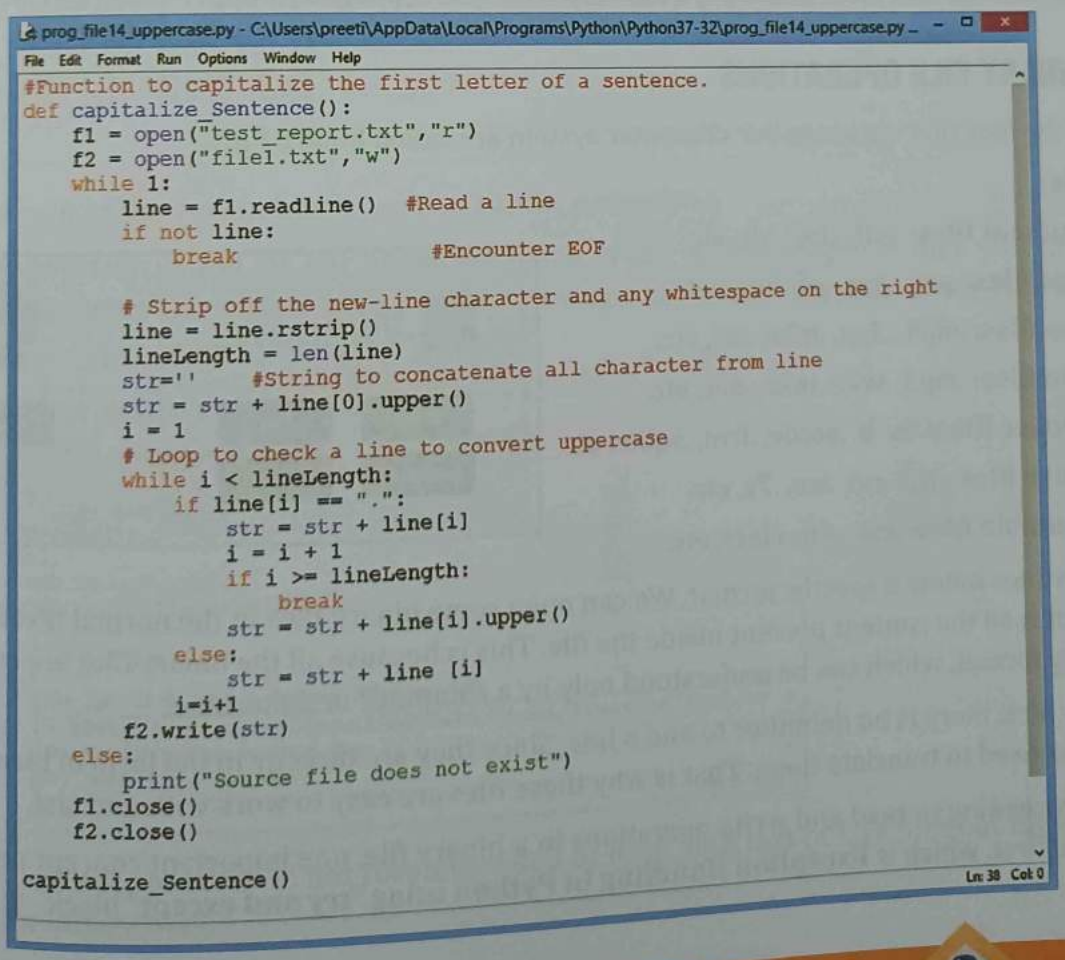
### Practical Implementation–14

**[HOTS]**

A text file named "test_report.txt" exists on a disk with the following content (lines):

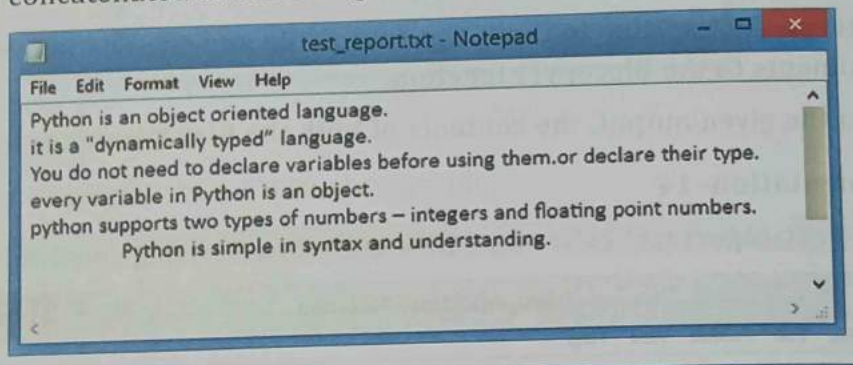


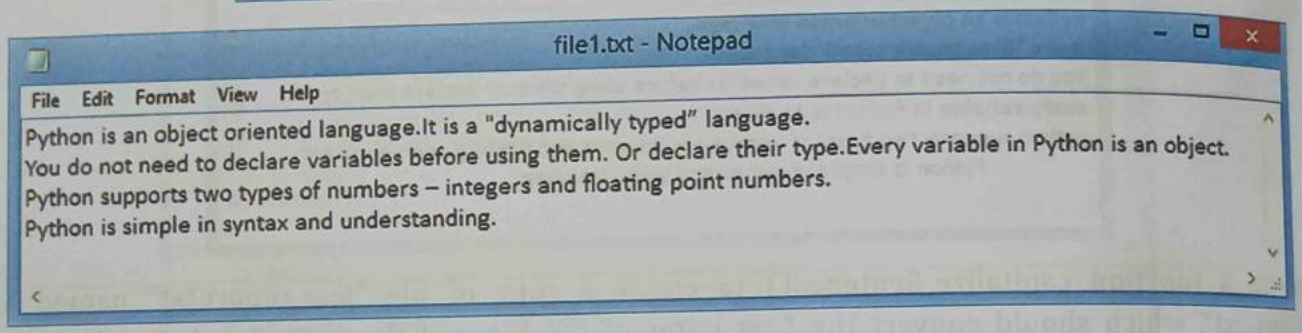

Write a function capitalize_Sentence() to create a copy of file "test_report.txt" named as "file1.txt", which should convert the first letter of the file and the first alphabetic character following a full stop into upper case.



**Explanation:**

In the given program, the lines from the file "file1" are read using readline() method. In order to calculate the exact length of the string, the white space or new-line character are removed using rstrip() method of string library. The first character of each sentence from the file is converted to uppercase and concatenated to the string 'str' wherever full stop is encountered.



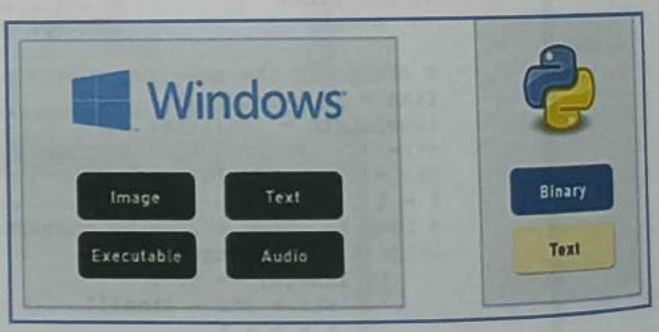

As is evident from the above output, the contents of the target file show that the first character of each line and the character following a full stop have been changed to upper case (if in lower case).

## 4.10 BINARY FILE OPERATIONS

Most of the files that we see in our computer system are called binary files.

**Example:**

- **Document files:** .pdf, .doc, .xls, etc.
- **Image files:** .png, .jpg, .gif, .bmp, etc.
- **Video files:** .mp4, .3gp, .mkv, .avi, etc.
- **Audio files:** .mp3, .wav, .mka, .aac, etc.
- **Database files:** .mdb, .accde, .frm, .sqlite, etc.
- **Archive files:** .zip, .rar, .iso, .7z, etc.
- **Executable files:** .exe, .dll, .class, etc.




All binary files follow a specific format. We can open some binary files in the normal text editor but we cannot read the content present inside the file. This is because all the binary files are encoded in the binary format, which can be understood only by a computer or a machine.

In binary files, there is no delimiter to end a line. Since they are directly in the form of binary, hence there is no need to translate them. That is why these files are easy to work with and fast.

Before proceeding to read and write operations in a binary file, one important concept needs to be addressed first, which is **Exception Handling in Python** using **"try and except"** block.

## 4.11 EXCEPTION HANDLING

An exception is an error that occurs during the execution of a program. The most common errors that are encountered in Python programs are classified as:

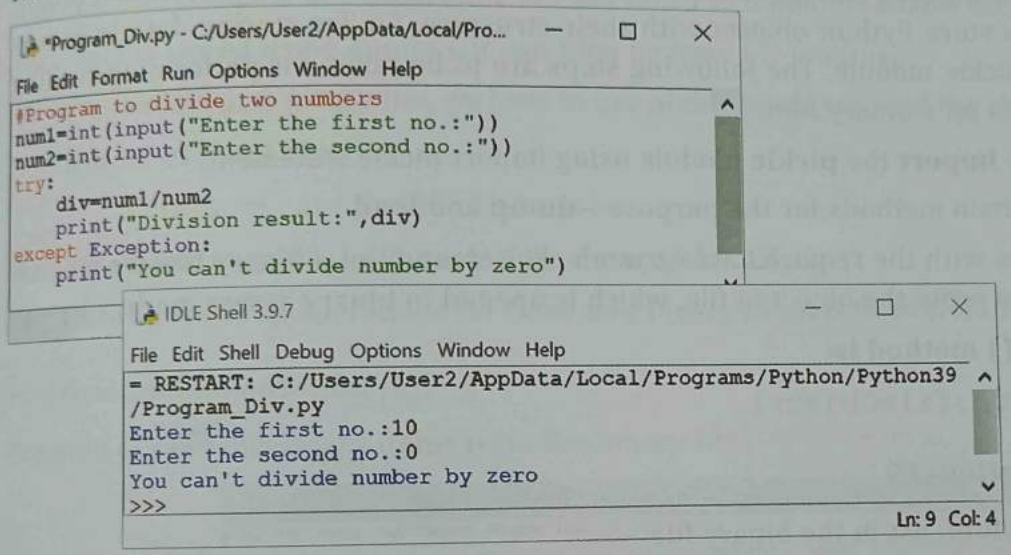
- Compile time error
- Syntax error
- Run-time error (Logical error)

How to handle these exceptions is termed as exception handling. try...except construct is used for exception handling in Python.

**try block:** try block signifies to run the code, which includes the statements which may generate some error or an exception.

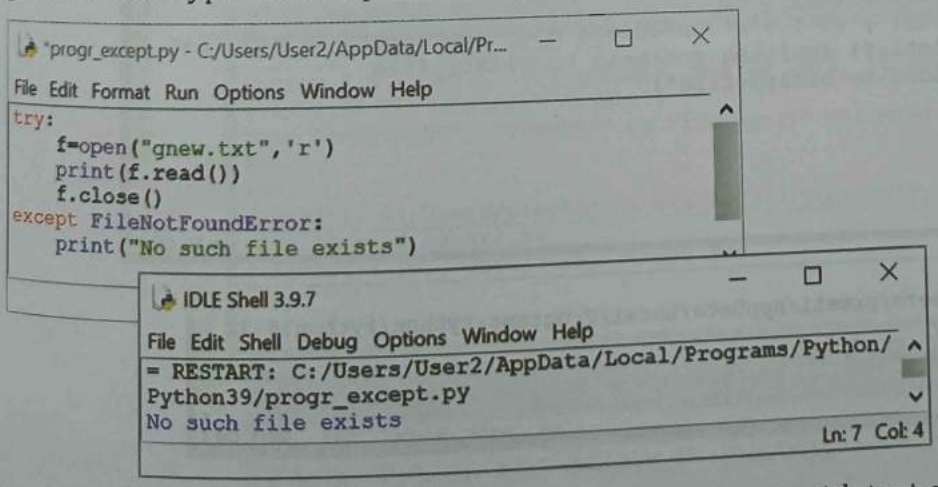
**except:** except block runs its code if an exception occurs.

For example,




With the help of the except block, we can handle exceptions.

**except Exception:** If you do not know the name of the error/exception, then you can simply take the Exception class name along with the **except** keyword. Exception is my base class and possesses all types of exceptions in Python.




**Note:** try...except construct is not recommended or used with text or CSV files but these are used with binary files.

## 4.12 READING AND WRITING DATA FROM/IN BINARY FILE

If we wish to write a structure such as a list or dictionary to a binary file and read it subsequently, we need to use the Python module **pickle**.

The module Pickle is used for serializing and de-serializing any Python object structure.

**Serialization** is the process of transforming data or an object in memory (RAM) to a stream of bytes called byte streams. These byte streams in a binary file can then be stored in a disk or in a database or sent through a network. Serialization process is also called pickling. **Pickling** refers to the process of converting the structure to a byte stream before writing to the file. While reading the contents of the file, a reverse process called **Unpickling** is used to convert the byte stream back to the original structure.

We know that the methods provided in Python for writing/reading a file work with string parameters. So, when we want to work on a binary file, conversion of data at the time of reading as well as writing is required. **Pickle** module can be used to store any kind of object in a binary file as it allows us to store Python objects with their structure. So, for storing data in binary format, we will use pickle module. The following steps are to be taken for performing reading and writing operations on a binary file:

1. First, we need to **import** the **pickle** module using import pickle statement.

   It provides two main methods for the purpose—**dump** and **load.**

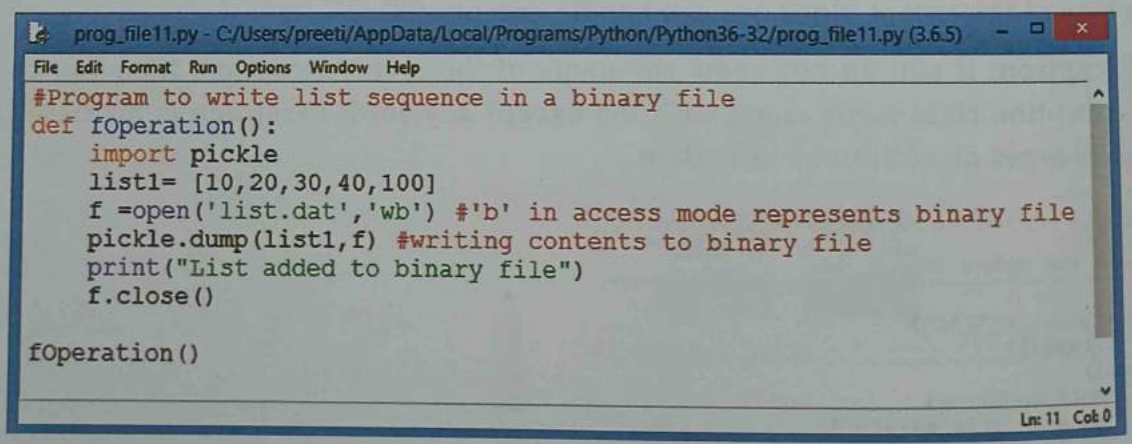
2. Open a binary file with the required access mode. For creation of a binary file, we will use **pickle.dump()** to write the object in file, which is opened in binary access mode.

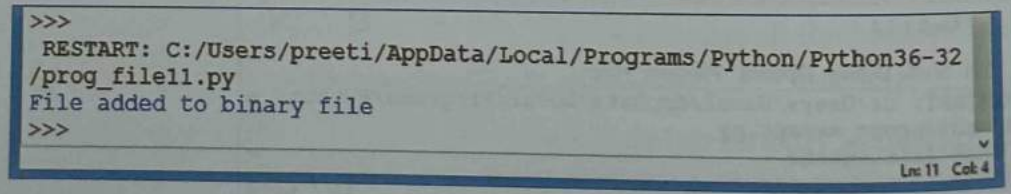
   **Syntax of dump() method is:**

   ```
   dump(object, fileObject)
   ```

### Practical Implementation-15

Program to write structure, list in the binary file.

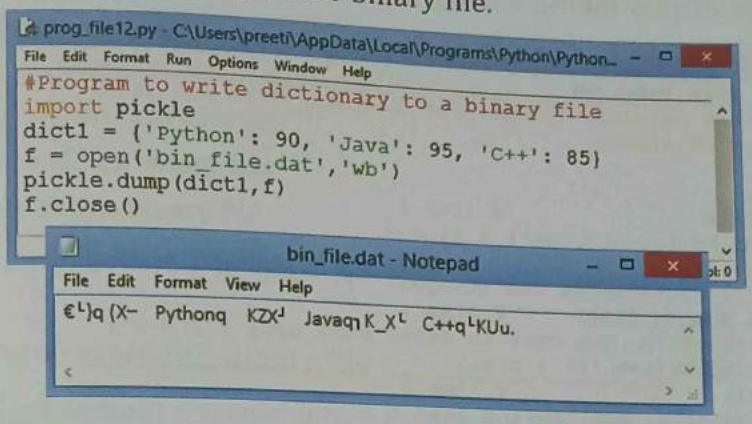





## Practical Implementation–16

Program to write structure, dictionary to the binary file.




As shown in the above program, the dictionary items are added to the binary file "bin_file.dat". However, the contents of a binary file are not much in human-readable form.

Once data is stored using dump(), it can then be used for reading.

3. For reading data from a file, we have to use **pickle.load()** to read the object from pickled file.

   **Syntax of load() is:**

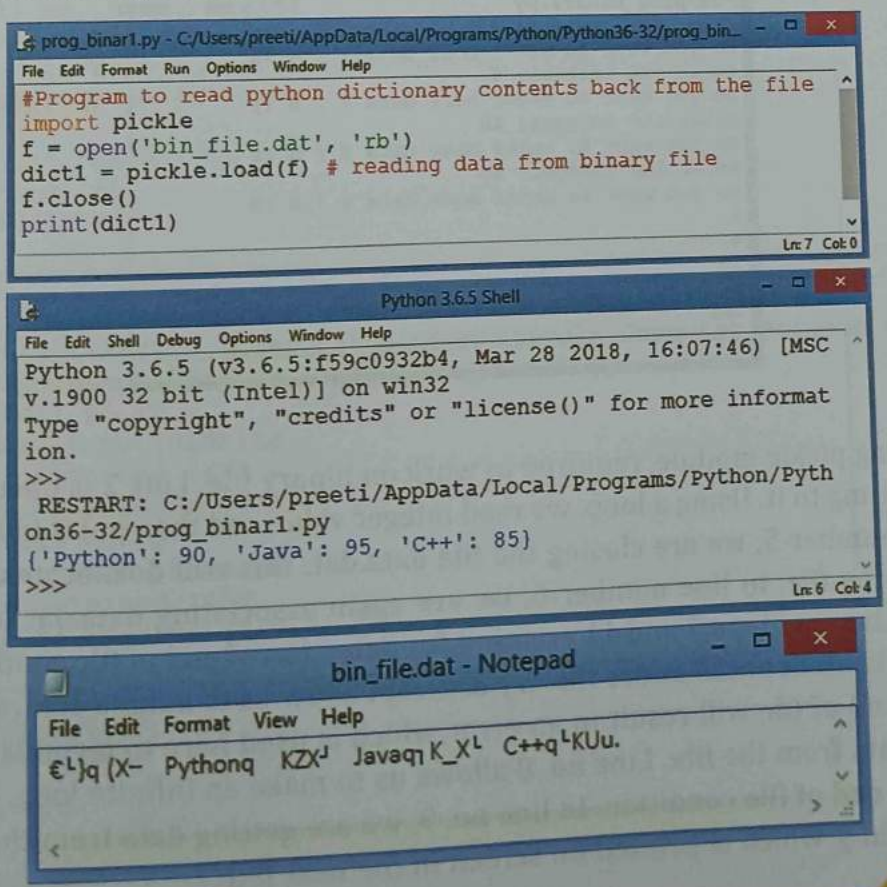
   ```
   object = load(fileObject)
   ```

**Note: We need to call load() each time dump() is called.**

4. Once all required operations are done on a binary file, it is to be closed using close() method.

## Practical Implementation–17

Program to read dictionary items from the binary file.



### Practical Implementation–18

Program to store and display multiple integers in and from a binary file.

prog_binar2.py - C:/Users/preeti/AppData/Local/Programs/Python/Python36-32/prog_binar2.py (3.6.5)

File Edit Format Run Options Window Help

```
#Program to store multiple integers in a binary file and read
#and display it on screen

def binfile():
    import pickle                # line 1
    file = open('data.dat','wb') # line 2
    while True:
        x = int(input("Enter the integer: "))  # line 3
        pickle.dump(x,file)      # line 4
        ans = input('Do you want to enter more data Y / N :')
        if ans.upper()== 'N' : break
    file.close()                 # line 5
    file = open('data.dat','rb') # line 6
    try :                        # line 7
        while True :             # line 8
            y = pickle.load(file) # line 9
            print(y)             # line 10
    except EOFError :            # line 11
        pass
    file.close()                 # line 12

binfile()
```

Ln: 22 Col: 9

**Output:**

Python 3.6.5 Shell

File Edit Shell Debug Options Window Help

```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16
900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for m
>>>
 RESTART: C:/Users/preeti/AppData/Local/Programs
-32/prog_binar2.py
Enter the integer: 2
Do you want to enter more data Y / N :Y
Enter the integer: 6
Do you want to enter more data Y / N :Y
Enter the integer: 10
Do you want to enter more data Y / N :Y
Enter the integer: 66
Do you want to enter more data Y / N :N
2
6
10
66
>>>
```

**Explanation:**

Line 1 is importing pickle module, required to work on binary file. Line 2 is opening a binary file (data.dat) for writing to it. Using a loop, we read integer value and then put it in file using line no. 3 and 4. In line number 5, we are closing the file data.dat; this will deallocate all the resources being used with the file. In line number 6, we are again associating data.dat to file stream for reading from it. Line numbers 7 and 11 are used for detection of end of file condition. This helps us in reading the content of the entire file. try & except allow us to handle errors in the program. Also, reading at end of file will result in an error which is used here to terminate the loop used for reading the data from the file. Line no. 8 allows us to make an infinite loop, as the same will be handled using end of file condition. In line no. 9, we are getting data from the binary file and storing the same in y, which is printed on screen in the next line.

This is to remember that one load function will get data for one dump(). Last statement will again close the file.

We will now discuss the four major operations performed using a binary file such as—

1. Inserting/Appending a record in a binary file
2. Reading a record from a binary file
3. Searching a record in a binary file
4. Updating a record in a binary file

We will now discuss each of them. The file in consideration is "student" file with the fields Roll_no, name and marks.

### Inserting/Appending a record in a binary file

Inserting or adding (appending) a record into a binary file requires importing pickle module into your program followed by dump() method to write onto the file.

### Practical Implementation–19

Program to insert/append a record in the binary file "student.dat".

*prog_bin_insert.py - C:\Users\Ashish Aggarwal\Documents\Books\Computer Science with P...

File Edit Format Run Options Window Help

```
# Program for inserting/appending a record in a binary file- student

import pickle
record = []
while True:
    roll_no = int(input("Enter student Roll no :"))
    name = input("Enter the student Name :")
    marks = int(input("Enter the marks obtained :"))
    data = [roll_no, name, marks]
    record.append(data)
    choice = input("Wish to enter more records (Y/N)?: ")
    if choice.upper() == 'N':
        break
f = open("student","wb")
pickle.dump(record, f)
print("Record Added")
f.close()
```

```
>>>
= RESTART: C:\Users\Ashish Aggarwal\Documents\Books\Computer Science with Python
\CS Class 12\Supplement\Chapter 4\prog_bin_insert.py
Enter student Roll no :2
Enter the student Name :Radhika
Enter the marks obtained :490
Wish to enter more records (Y/N)?: y
Enter student Roll no :3
Enter the student Name :Shaurya
Enter the marks obtained :450
Wish to enter more records (Y/N)?: n
Record Added
```

student - Notepad

File Edit Format View Help

```
€l]q (]q (K1X•
Radhikaq1Mê e]ql(KlX•
Shauryaq1MÅ ee.
```

As shown in the given program, inserting and adding a record in student file begins with importing pickle module. Then, iterating for the number of records to be added is done using while loop. Input is accepted from the user for roll number, name and marks. These fetched values are saved as a list to be appended. Once the record is created, it is appended to the binary file "student" using the dump() method, which writes the object onto the opened file.

Finally the file is closed explicitly and the record is added into the student.dat file as shown in the output. Since binary file contains unreadable characters, it becomes difficult to read and understand, but it is directly used and understood by the computer.

### Reading a record from a binary file

The following practical implementation illustrates how a record is read from a binary file.

### Practical Implementation–20

Program to read a record from the binary file "student.dat"

*prog_bin_read.py - C:\Users\Ashish Aggarwal\Documents\Books\Computer Science with Py...

File Edit Format Run Options Window Help

```
# Program to read a record from the binary file- "student.dat"

import pickle
f = open("student","rb")
stud_rec = pickle.load(f) # To read the object from the opened file
print("Contents of student file are:")
# reading the fields from the file
for R in stud_rec:
    roll_no = R[0]
    name = R[1]
    marks = R[2]
    print(roll_no, name, marks)
f.close()
```

```
= RESTART: C:\Users\Ashish Aggarwal\Documents\Books\Computer Science with Python
\CS Class 12\Supplement\Chapter 4\prog_bin_read.py
Contents of student file are:
2 Radhika 490
3 Shaurya 450
>>>
```

The above program deals with reading the contents from binary file student using load() method of pickle module. It is used to read the object from the opened file. The syntax for this is given by the statement—

```
object = pickle.load(file)
```

Once the contents are read, it gets stored inside the object, stud_rec. All the data of the respective fields from this object is held in the respective variables—roll_no, name and marks—and are finally displayed as the output.

### Searching a record in a binary file

Searching the binary file "student" is carried out on the basis of the roll number entered by the user. The file is opened in the read-binary mode and gets stored in the file object, f. load() method is used to read the object from the opened file. A variable 'found' is used which will tell the status of the search operation being successful or unsuccessful. Each record from the file is read and the content of the field, roll no, is compared with the roll number to be searched. Upon the search being successful, appropriate message is displayed to the user as shown in the practical implementation that follows.

### Practical Implementation–21

Program to search a record from the binary file "student.dat" on the basis of roll number.

*prog_bin_search.py - C:\Users\Ashish Aggarwal\Documents\Books\Computer Science with ...

File Edit Format Run Options Window Help

```
# Program to search a record from the binary file- "student.dat"

import pickle
f = open("student","rb")
stud_rec = pickle.load(f) # To read the object from the opened file
found = 0
rno = int(input("Enter the roll number to search:"))

for R in stud_rec:
    if R[0] == rno:
        print("Successful Search", R[1], "Found!")
        found = 1
        break
if found == 0:
    print("Sorry, record not found")
f.close()
```

```
= RESTART: C:\Users\Ashish Aggarwal\Documents\Books\Computer Science with Python
\CS Class 12\Supplement\Chapter 4\prog_bin_search.py
Enter the roll number to search:2
Successful Search Radhika Found!
```

### Updating a record in a binary file

This program requires roll number to be fetched from the user whose name is to be updated.

*prog_bin_update.py - C:/Users/Ashish Aggarwal/Documents/Books/Computer Science with ...

File Edit Format Run Options Window Help

```
# Program to update the name of the student from the binary file

import pickle
f = open("student", "rb+")
stud_rec = pickle.load(f) # To read the object from the opened file
found = 0
rollno = int(input("Enter the roll number to search:"))
for R in stud_rec:
    rno = R[0]
    if rno == rollno:
        print("Current name is:", R[1])
        R[1] = input("New Name:")
        found = 1
        break

if found == 1:
    f.seek(0) # Taking the file pointer to the beginning of the file
    pickle.dump(stud_rec, f)
    print("Name Updated!!!")
f.close()
```

```
= RESTART: C:/Users/Ashish Aggarwal/Documents/Books/Computer Science with Python
/CS Class 12/Supplement/Chapter 4/prog_bin_update.py
Enter the roll number to search:2
Current name is: Radhika
New Name:Sonia
Name Updated!!!
```

Once the record is found, the file pointer is moved to the beginning of the file using seek(0) statement, and then the changed name is written to the file and the record is updated. seek() method is used for random access to the file.

We will be learning more about it in the next section.

### Practical Implementation–22

Write a menu-driven program to perform all the basic operations using dictionary on student binary file such as inserting, reading, updating, searching and deleting a record.

File Edit Format Run Options Window Help

```
import os
import pickle

# Accepting data for Dictionary
def insertRec():
    rollno = int(input("Enter roll number:"))
    name = input("Enter Name:")
    marks = int(input("Enter Marks:"))
    # Creating the Dictionary
    rec = {"Rollno": rollno, "Name": name, "Marks": marks}
    # Writing the Dictionary
    f = open("student.dat", "ab")
    pickle.dump(rec, f)
    f.close()

# Reading the records
def readRec():
    f = open("student.dat", "rb")
    while True:
        try:
            rec = pickle.load(f)
            print("Roll Num:", rec['Rollno'])
            print("Name:", rec["Name"])
            print("Marks:", rec["Marks"])
        except EOFError:
            break
    f.close()

# Searching a record  based on Rollno
def searchRollNo(r):
    f = open("student.dat", "rb")
    flag = False
    while True:
        try:
            rec = pickle.load(f)
            if rec['Rollno'] == r:
                print("Roll Num:", rec['Rollno'])
                print("Name:", rec["Name"])
                print("Marks:", rec["Marks"])
                flag = True
        except EOFError:
            break
    if flag == False:
        print("No Record Found")
    f.close()

# Marks Modification for a RollNo
def updateMarks(r, m):
    f = open("student.dat", "rb")
    reclst = []
    while True:
        try:
            rec = pickle.load(f)
            reclst.append(rec)
        except EOFError:
            break
    f.close()
    for i in range(len(reclst)):
        if reclst[i]['Rollno'] == r:
            reclst[i]['Marks'] = m
    f = open("student.dat", 'wb')
    for x in reclst:
        pickle.dump(x, f)
    f.close()
```

```
# Deleting a record based on RollNo
def deleteRec(r):
    f = open("student.dat", "rb")
    reclst = []
    while True:
        try:
            rec = pickle.load(f)
            reclst.append(rec)
        except EOFError:
            break
    f.close()
    f = open("student.dat", 'wb')
    for x in reclst:
        if x['Rollno'] == r:
            continue
        pickle.dump(x, f)
    f.close()

while True:
    print('Type 1 to insert rec.')
    print('Type 2 to display rec.')
    print('Type 3 to search rec.')
    print('Type 4 to update rec.')
    print('Type 5 to delete rec.')
    print('Enter your choice 0 to exit')
    choice = int(input("Enter your choice:"))
    if choice == 0:
        break
    elif choice == 1:
        insertRec()
    elif choice == 2:
        readRec()
    elif choice == 3:
        r = int(input("Enter a rollno to search:"))
        searchRollNo(r)
    elif choice == 4:
        r = int(input("Enter a rollno:"))
        m = int(input("Enter new Marks:"))
        updateMarks(r, m)
    elif choice == 5:
        r = int(input("Enter a rollno:"))
        deleteRec(r)
```

```
>>>
= RESTART: C:\Users\Ashish Aggarwal\Documents\Books\Computer Science with Pyt
hon\CS Class 12\Supplement\Chapter 4\prog_bin_allopertns.py
Type 1 to insert rec.
Type 2 to display rec.
Type 3 to search rec.
Type 4 to update rec.
Type 5 to delete rec.
Enter your choice 0 to exit
Enter your choice:1
Enter roll number:20
Enter Name:Ankur
Enter Marks:400
Type 1 to insert rec.
Type 2 to display rec.
Type 3 to search rec.
Type 4 to update rec.
Type 5 to delete rec.
Enter your choice 0 to exit
Enter your choice:2
Roll Num: 20
Name: Ankur
Marks: 400
Type 1 to insert rec.
Type 2 to display rec.
Type 3 to search rec.
Type 4 to update rec.
Type 5 to delete rec.
Enter your choice 0 to exit
Enter your choice:3
Enter a rollno to search:20
Roll Num: 20
Name: Ankur
Marks: 400
Type 1 to insert rec.
Type 2 to display rec.
Type 3 to search rec.
Type 4 to update rec.
Type 5 to delete rec.
Enter your choice 0 to exit
Enter your choice:0
>>>
```

## 4.13 RANDOM ACCESS IN FILES USING TELL() AND SEEK()

Till now, in all our programs we laid stress on the sequential processing of data in a text and binary file. But files in Python allow random access of the data as well using built-in methods seek() and tell().

**seek()**—seek() function is used to change the position of the file handle (file pointer) to a given specific position. File pointer is like a cursor, which defines from where the data has to be read or written in the file.

Python file method seek() sets the file's current position at the offset. This argument is optional and defaults to 0, which means absolute file positioning. Other values are: 1, which signifies seek is relative (may change) to the current position, and 2, which means seek is relative to the end of file. There is no return value.

The reference point is defined by the "from_what" argument. It can have any of the three values:

0: sets the reference point at the beginning of the file, which is by default.

1: sets the reference point at the current file position.

2: sets the reference point at the end of the file position.

seek() can be done in two ways:

- Absolute Positioning
- Relative Positioning

Absolute referencing using seek() gives the file number on which the file pointer has to position itself. The syntax for seek() is—

```
f.seek(file_location)     #where f is the file pointer
```

For example, f.seek(20) will give the position or file number where the file pointer has been placed. This statement shall move the file pointer to 20th byte in the file no matter where you are.

Relative referencing/positioning has two arguments, offset and the position from which it has to traverse. The syntax for relative referencing is:

```
f.seek(offset, from_what)  #where f is file pointer
```

For example,

f.seek(–10,1) from current position, move 10 bytes backward

f.seek(10,1) from current position, move 10 bytes forward

f.seek(–20,1) from current position, move 20 bytes backward

f.seek(10,0) from beginning of file, move 10 bytes forward

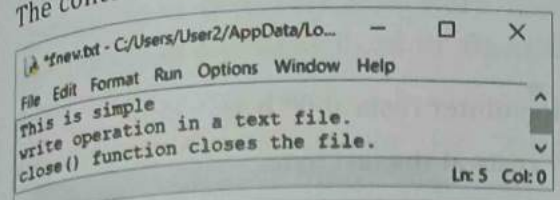
**POINT TO REMEMBER**

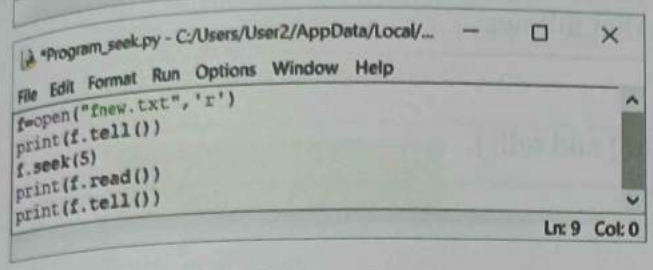
It must be remembered that Python 3.x only supports text file seeks from the beginning of the file. seek() with negative offset only works when the file is opened in binary mode.

Example:

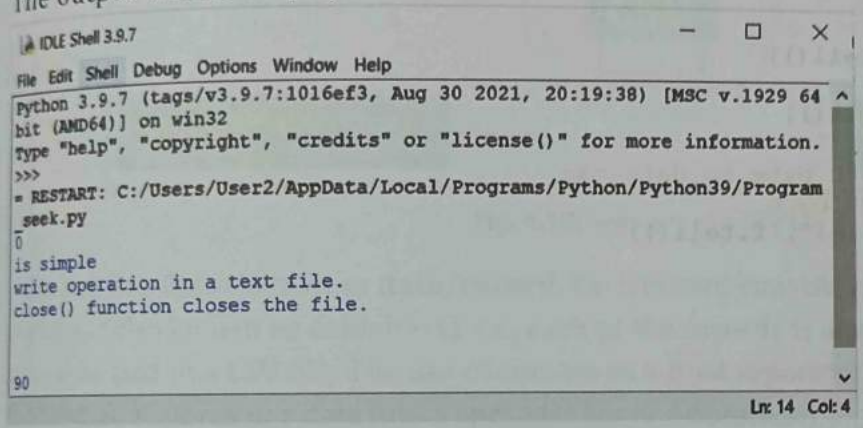
**To illustrate seek() and tell()**

The contents of text file 'fnew.txt' is:

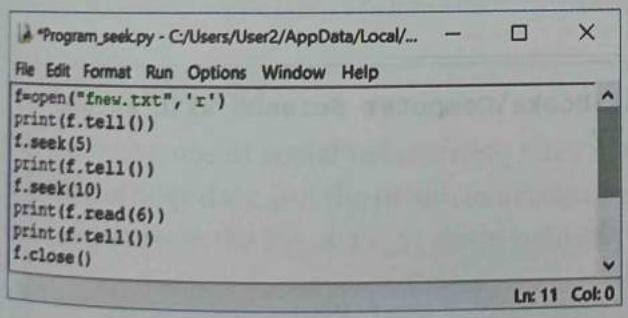


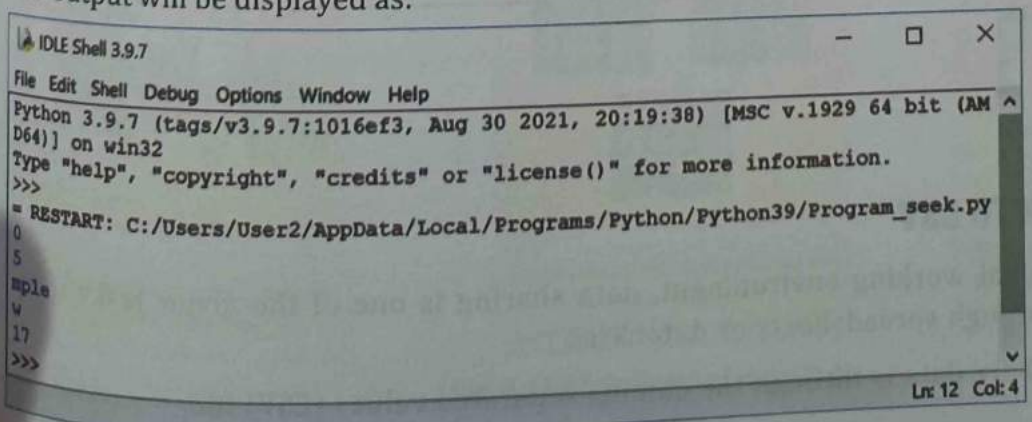

The output will be displayed as:




**To illustrate seek()and tell() functions (modification of above example)**




The output will be displayed as:



**tell()**—tell() returns the current position of the file read/write pointer within the file. If we have moved our file pointer through seek(), tell() is used to tell the position where file pointer is currently pointing to.

Its syntax is:

```
f.tell()      #where f is file pointer
```

- When you open a file in reading/writing mode, the file pointer rests at $0^{th}$ byte.
- When you open a file in append mode, the file pointer rests at the last byte.

This is illustrated in the practical implementation that follows:

### Practical Implementation–23

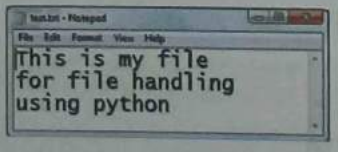
Program to read byte by byte from a file using seek() and tell().

File Edit Format Run Options Window Help

```
#Illustrating seek() and tell()
#Reading byte by byte

f = open("test.txt")
print("Before reading: ", f.tell())
s = f.read()
print("After reading: ", f.tell())
f.seek(0)
#Brings file pointer to the 0th byte so data can
#be written/read from the next byte
print("From the beginning again:", f.tell())
s = f.read(4)
print("First 4 bytes are:", s)
print(f.tell())
s = f.read(3)
print("next 3 bytes:", s)
print(f.tell())
f.close()
```

**Contents of "test.txt":**

```
>>>
= RESTART: C:\Users\Ashish Aggarwal\Documents\Books\Computer Science with Python
\CS Class 12\Supplement\Chapter 4\PracImp23.py
Before reading:  0
After reading:  48
From the beginning again: 0
First 4 bytes are: This
4
next 3 bytes:  is
7
```




## 4.14 INTRODUCTION TO CSV

In today's organizational working environment, data sharing is one of the major tasks to be carried out, largely through spreadsheets or databases.

A basic approach to share data is through the comma separated values (CSV) file.

CSV is a simple flat file in a human readable format which is extensively used to store tabular data, in a spreadsheet or database. A CSV file stores tabular data (numbers and text) in plain text.

Files in the CSV format can be imported to and exported from programs that store data in tables, such as Microsoft Excel or OpenOffice Calc.

Already defined, CSV stands for "**comma separated values**". Thus, we can say that a comma separated file is a delimited text file that uses a comma to separate values.

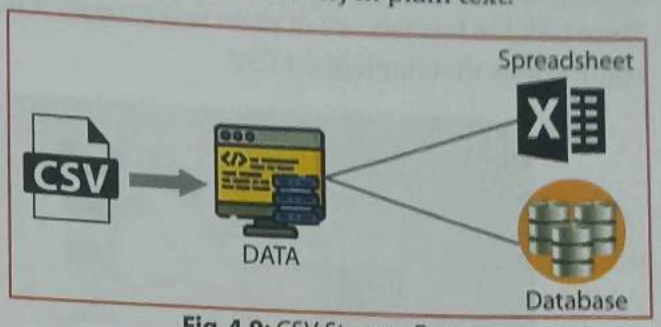



**Fig. 4.9:** CSV Storage Format

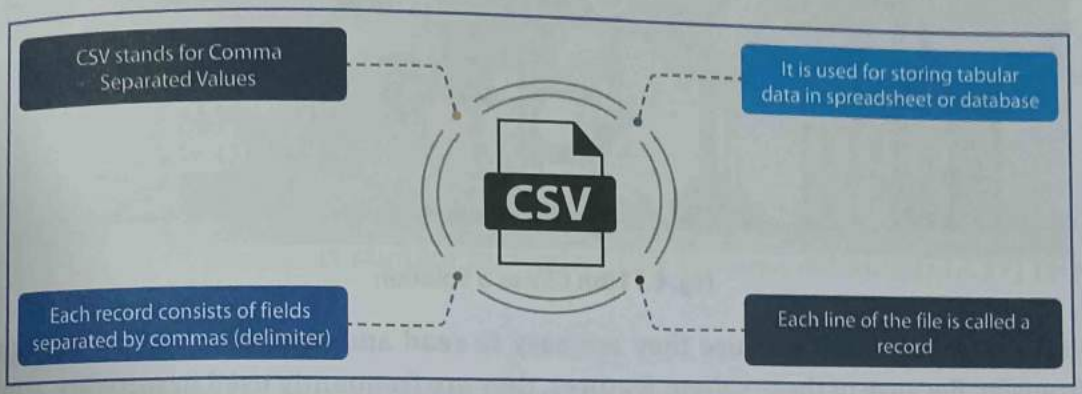



**Fig. 4.10:** Features of a CSV file

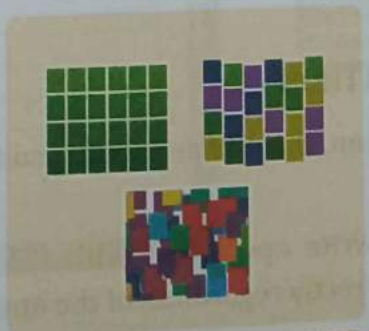
Each line in a file is known as **data/record**. Each record consists of one or more fields, separated by commas (also known as delimiters), *i.e.*, each of the records is also a part of this file. Tabular data is stored as **text** in a CSV file. The use of comma as a field separator is the source of the name for this file format. It stores our data into a spreadsheet or a database.

**CTM:** The most commonly used delimiter in a CSV file is usually a **comma**.

## 4.15 WHY USE CSV

The extensive use of social networking sites and their various associated applications requires the handling of huge data. But the problem arises as to how to handle and organize this large unstructured data, as shown in the Fig. 4.11(a) given below.

Storing huge and exponentially growing data sets



Processing data having complex structure (structured, un-structured, semi-structured)

Bringing huge amount of data to computation unit becomes a bottleneck

**Fig. 4.11(a):** Problem of Huge Data

The solution to the above problem is CSV (Fig. 4.11(b)). Thus, CSV organizes data into a structured form and, hence, the proper and systematic organization of this large amount of data is done by CSV. Since CSV file formats are of plain text format, it makes it very easy for website developers to create applications that implement CSV.


**Fig. 4.11(b):** CSV as a Solution

CSV files are commonly used because they are easy to read and manage, small in size, and fast to process/transfer. Because of these salient features, they are frequently used in software applications, ranging anywhere from online e-commerce stores to mobile apps to desktop tools. For example, Magento, an e-commerce platform, is known for its support of CSV.

Thus, in a nutshell, the several advantages that are offered by CSV files are as follows:

- CSV is faster to handle.
- CSV is smaller in size.
- CSV is easy to generate and import onto a spreadsheet or database.
- CSV is human readable and easy to edit manually.
- CSV is simple to implement and parse.
- CSV is processed by almost all existing applications.

After understanding the concept and importance of using CSV files, we will now discuss the read and write operations on CSV files.

## 4.16 CSV FILE HANDLING IN PYTHON

For working with CSV files in Python, there is an inbuilt module called **csv**. It is used to read and write tabular data in CSV format.

Therefore, to perform read and write operations with CSV file, we must import **csv module**. csv module can handle CSV files correctly regardless of the operating system on which the files were created.

Along with this module, open() function is used to open a CSV file and return file object. We load the module in the usual way using import:

```
>>> import csv
```

Like other files (text and binary) in Python, there are two basic operations that can be carried out on a CSV file:

1. Reading from a CSV file
2. Writing to a CSV file

Let us discuss these CSV operations.

### 4.16.1 Reading from a CSV File

Reading from a CSV file is done using the **reader** object. The CSV file is opened as a text file with Python's built-in open() function, which returns a file object. This creates a special type of object to access the CSV file (reader object), using the reader() function.

The reader object is an iterable that gives us access to each line of the CSV file as a list of fields. You can also use next() directly on it to read the next line of the CSV file, or you can treat it like a list in a for loop to read all the lines of the file (as lists of the file's fields).

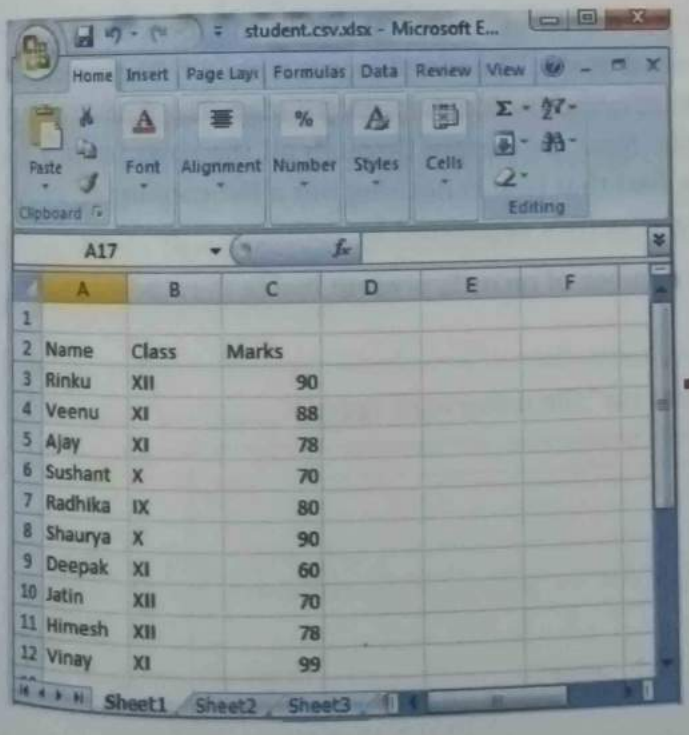
This is shown in Practical Implementation–24.

(i) Before this, enter the student details in spreadsheet and select File → Save As option.

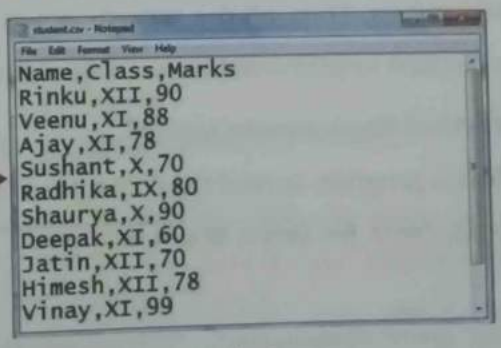
(ii) Type the name of the file as student and select CSV (comma delimited) (.CSV) format from Save As type option.

(iii) Click Save button.

(iv) Next step is to open the Notepad and open student.csv file that contains equivalent data for student.xls in comma separated values.




student.xls




student.csv

In student.csv (notepad) file, the first line is the header and remaining lines are the data/records. The fields are separated by comma, or we may say the separator character. In general, the separator character is called a delimiter, and the comma is not the only one used. Other popular delimiters include the tab (\t), colon (:) and semi-colon (;) characters.

### Practical Implementation-24

Write a program to read the contents of "student.csv" file.

```
File Edit Format Run Options Window Help
#Python program to read "student.csv" file contents.

import csv
f = open("student.csv", 'r') #opens file in read mode
csv_reader = csv.reader(f) #reader() function reads the file object
#csv_reader is the csv reader object
#Reading the student file record by record
for row in csv_reader:
    print(row)

f.close() #Explicitly closing the file
```

```
>>>
= RESTART: C:/Users/Ashish Aggarwal/Documents/Books/Computer Science with Python
/CS Class 12/Supplement/Chapter 4/PracImp24.py
['Name', 'Class', 'Marks']
['Rinku', 'XII', '90']
['Veenu', 'XI', '88']
['Ajay', 'XI', '78']
['Sushant', 'X', '70']
['Radhika', 'IX', '80']
['Shaurya', 'X', '90']
['Deepak', 'XI', '60']
['Jatin', 'XII', '70']
['Himesh', 'XII', '78']
['Vinay', 'XI', '99']
>>>
```

**Explanation:**

As seen from the above output, every record is stored in reader object in the form of a List. In the above code, we first open the CSV file in read mode. The file object is named **f**. The file object is converted to csv.reader object. We save the csv.reader object as **csv_reader**. The reader object is used to read records as lists from a csv file. Now, we iterate through all the rows using a for loop. When we try to print each row, one can find that row is nothing but a list containing all the field values. Thus, all the records are displayed as lists separated by comma.

In the next implementation, we will count the number of records present inside the student.csv file.

### Practical Implementation-25

Write a program to read the contents of "student.csv" file using with open().

```
File Edit Format Run Options Window Help
#Demonstrate use of with open()

import csv
with open("student.csv", 'r') as csv_file:
    reader = csv.reader(csv_file)
    rows = [] #list to store the file data

    for rec in reader: #copy data into the list 'rows'
        rows.append(rec)
    print(rows)
```

```
>>>
= RESTART: C:/Users/Ashish Aggarwal/Documents/Books/Computer Science with Python
/CS Class 12/Supplement/Chapter 4/PracImp25.py
[['Name', 'Class', 'Marks'], ['Rinku', 'XII', '90'], ['Veenu', 'XI', '88'], ['Aj
ay', 'XI', '78'], ['Sushant', 'X', '70'], ['Radhika', 'IX', '80'], ['Shaurya', '
X', '90'], ['Deepak', 'XI', '60'], ['Jatin', 'XII', '70'], ['Himesh', 'XII', '78
'], ['Vinay', 'XI', '99']]
```



The above modified code uses "with open()" function, the only difference being that the file being opened using with open() gets automatically closed after the program execution gets over, unlike open() where we need to give close() statement explicitly.

### Practical Implementation–26

Write a program to count the number of records present in "student.csv" file.

```
File Edit Format Run Options Window Help
#Python program to count the number of records present in "student.csv" file

import csv
f = open("student.csv", 'r')
csv_reader = csv.reader(f) #csv_reader is the csv reader object
c = 0
#Reading the student file record by record
for row in csv_reader:
    c = c + 1
print("No. of records are:", c)

f.close() #Explicitly closing the file
```

```
>>>
= RESTART: C:/Users/Ashish Aggarwal/Documents/Books/Computer Science with Python
/CS Class 12/Supplement/Chapter 4/PracImp26.py
No. of records are: 11
```

**Explanation:**

In the above program, a special type of object is created to access the CSV file (reader object), which is csv_reader using the reader() function. The reader object is an iterable that gives us access to each line of the CSV file as a list of fields.

The variable 'c' is used as a counter variable to count the number of rows/records present in this file, which is finally printed and thus the output is so obtained.

One of the important observations from the output is the number of records which is being displayed as 11 instead of 10. This is so because the header (first line) in the student csv file is also treated as a record only. This limitation is overcome in the next implementation.

### Practical Implementation–27

Program to count the exact number of records present in the csv file excluding the header.

```
File Edit Format Run Options Window Help
#Python program to count the no. of records present in
# "student.csv" file

import csv
f = open("student.csv", 'r')
csv_reader = csv.reader(f) #csv_reader is the csv reader object
csvRows = []
value = 0
for row in csv_reader:
    if csv_reader.line_num == 0: #skip the first header row
        continue
    csvRows.append(row)
    value = len(list(csv_reader))
print("No. of records are:", value)
f.close()
```

```
>>>
= RESTART: C:/Users/Ashish Aggarwal/Documents/Books/Computer Science with Python
/CS Class 12/Supplement/Chapter 4/PracImp27.py
No. of records are: 10
```

In the above program we have used line_num object of CSV file. Our csv_reader_object has a method called line_num that returns the number of lines in our CSV.

Then, if statement checks if the line is the first line or not. If the condition is true, *i.e.*, if it is the header line, then it is ignored using continue statement and the counting of records is resumed from the second line onwards. Also, line_num object always stores the current line in consideration and, hence, the correct output for 10 records is so obtained.

**CTM: line_num** is nothing but a counter which returns the number of rows which have been iterated.

### Practical Implementation–28

Program to print the records in the form of comma separated values, instead of lists.

```
File Edit Format Run Options Window Help
#Python program to print records in the form of
#comma separated values
import csv
f = open("student.csv", "r")
csv_reader = csv.reader(f) #csv_reader is the csv reader object
for row in csv_reader:
    print(','.join(row))

f.close()
```

```
= RESTART: C:\Users\Ashish Aggarwal\Documents\Books\Computer Science with Python
\CS Class 12\Supplement\Chapter 4\PracImp5.py
Name,Class,Marks
Rinku,XII,90
Veenu,XI,88
Ajay,XI,78
Sushant,X,70
Radhika,IX,80
Shaurya,X,90
Deepak,XI,60
Jatin,XII,70
Himesh,XII,78
Vinay,XI,99
```

In the above program, we have used a new function join(). join() is a string method that joins the string by a string separator; here the separator is ‘,’. Thus, all the records are displayed as a string separated by a comma separator and not as a list and hence the output is so obtained.

### Practical Implementation–29

Program to search the record of a particular student from CSV file on the basis of inputted name.

```
File Edit Format Run Options Window Help
#Python program to search record for given students
#name from csv file

import csv
f = open("student.csv","r")
csv_reader = csv.reader(f) #csv_reader is the csv reader object
name = input("Enter the name to be searched: ")
for row in csv_reader:
    if(row[0] == name):
        print(row)
```

```
>>>
= RESTART: C:\Users\Ashish Aggarwal\Documents\Books\Computer Science with Python
\CS Class 12\Supplement\Chapter 4\PracImp6.py
Enter the name to be searched: Jatin
['Jatin', 'XII', '70']
```

Till now we have covered the basics of how to use the csv module to read the contents of a CSV file. Now, we will discuss how to write to a CSV file in Python.

### 4.16.2 Writing to a CSV File

To write to a CSV file in Python, we can use the **csv.writer()** function. The csv.writer() function returns a writer object that converts the user's data into a delimited string. This string can later be used to write into CSV files using the writerow() function.

In order to write to a CSV file, we create a special type of object to write to the CSV file "**writer object**", which is defined in the CSV module, and which we create using the writer() function.

The **writerow()** method allows us to write a list of fields to the file. The fields can be strings or numbers or both. Also, while using writerow(), you do not need to add a new line character (or other EOL indicator) to indicate the end of the line; writerow() does it for you as necessary.

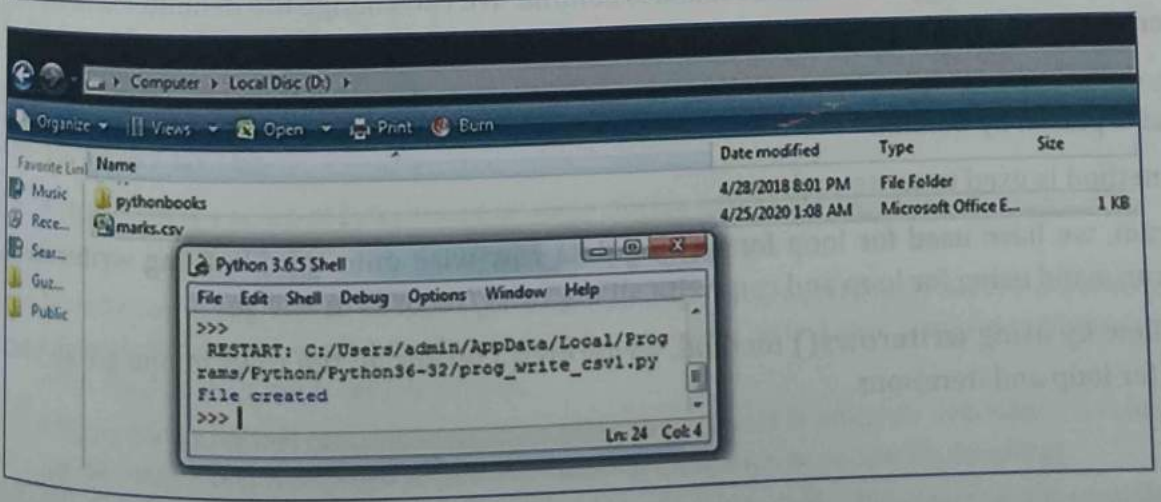
Let us write data onto a CSV file using writerow() method.

#### Practical Implementation–30

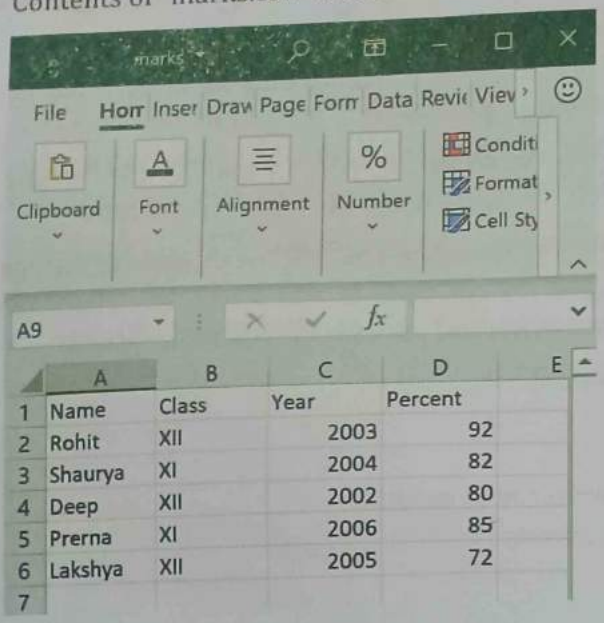
Program to write data onto "student" CSV file using writerow() method.

File Edit Format Run Options Window Help

```
#Program to write student data onto a csv file

#importing the csv module
import csv

#Field names
fields = ['Name','Class','Year','Percent']

#Data rows of csv file
rows = [['Rohit','XII','2003','92'],
        ['Shaurya','XI','2004','82'],
        ['Deep','XII','2002','80'],
        ['Prerna','XI','2006','85'],
        ['Lakshya','XII','2005','72']]

#Name of the csv file
filename = 'marks.csv'

#Writing to csv file
with open(filename,'w',newline='') as f:
    #By default, newline is '\r\n'
    #creating a csv writer object
    csv_w = csv.writer(f, delimiter = ',')
    #writing the fields (the column heading) once
    csv_w.writerow(fields)
    for i in rows:
        #writing the data row-wise
        csv_w.writerow(i)
    print("File Created")
```



Contents of "marks.csv" created:



| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | Name | Class | Year | Percent | |
| 2 | Rohit | XII | 2003 | 92 | |
| 3 | Shaurya | XI | 2004 | 82 | |
| 4 | Deep | XII | 2002 | 80 | |
| 5 | Prerna | XI | 2006 | 85 | |
| 6 | Lakshya | XII | 2005 | 72 | |
| 7 | | | | | |

**Explanation:**

In the given program, the very first line is for importing csv file into your program. Next, whatever are the column headings for our data are mentioned as a list in the variable called **fields**. All the data stored inside these fields is placed inside the variable called **rows**.

Now give the name of your file, let us say, **student.csv**. This will be created and stored inside your current working directory or the path that you mentioned (as we have given for D:/) for the attribute **"filename"**.

**'w'** stands for write mode and we are using the file by opening it using "**with open**", since using with open does not require the file to be closed explicitly. The next statement comprises the most important function used for writing onto csv file, viz. csv.writer(), to obtain a writer object and store it in the variable csv_w as the name of the variable, and this is the CSV object. writer() takes the name of file object 'f' as the argument. By default, the delimiter is comma (,).

writerow(fields) is going to write the fields which are the column headings into the file and have to be written only once. Using for loop, rows are traversed from the list of rows from the file. writerow(i) is writing the data row-wise in the for loop and in the end the file is automatically closed.

Also, while giving csv.writer(), the delimiter taken is comma. We can change the delimiter whenever and wherever required by changing the argument passed to delimiter attribute.

For example, delimiter = "|" (pipe symbol). You can put any character as delimiter and if nothing is given, comma is placed by default.

writerow() method is used to write each row.

In this program, we have used for loop for writing data row-wise onto the file using writerow() method. We can avoid using for loop and can write all the rows/records in one go.

This can be done by using **writerows()** method. writerows() writes all the rows in one go, so you need not use for loop and iterations.

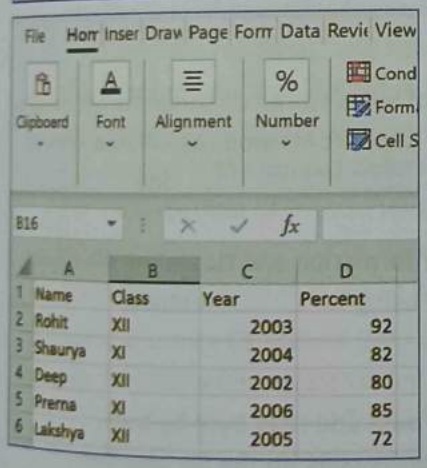


## Practical Implementation–31

Program to write data onto "student" csv file using writerows() method (modification of Practical Implementation–30).

```
File Edit Format Run Options Window Help
#Program to write student data onto a csv file

#importing CSV module
import csv

#Field Names
fields = ['Name', 'Class', 'Year', 'Percent']

#Data rows of csv file
rows = [
        ['Rohit','XII','2003','92'],
        ['Shaurya','XI','2004','82'],
        ['Deep','XII','2002','80'],
        ['Prerna','XI','2006','85'],
        ['Lakshya','XII','2005','72']
        ]

#Name of csv file
filename = "newmarks.csv"

#Writing to csv file
with open(filename, 'w', newline='') as f:
    #By default, newline is '\r\n'
    #creating a csv writer object
    csv_w = csv.writer(f, delimiter=',')
    #writing the fields (the column heading) once
    csv_w.writerow(fields)
    #writing the rows all at once
    csv_w.writerows(rows)
    print("All rows written in one go")
```

```
>>>
= RESTART: C:/Users/Ashish Aggarwal/Documents/Books/Computer Science with Python
/CS Class 12/Supplement/Chapter 4/PracImp31.py
All rows written in one go
>>>
```



| | A | B | C | D |
|---|---|---|---|---|
| 1 | Name | Class | Year | Percent |
| 2 | Rohit | XII | 2003 | 92 |
| 3 | Shaurya | XI | 2004 | 82 |
| 4 | Deep | XII | 2002 | 80 |
| 5 | Prerna | XI | 2006 | 85 |
| 6 | Lakshya | XII | 2005 | 72 |

## MEMORY BYTES

- A file in itself is a bunch of bytes stored on some storage devices like hard-disk, pen-drive, etc.
- Data files can be stored in two ways: (i) Text files (ii) Binary files.
- A text file stores information in ASCII or Unicode characters, where each line of text is terminated (delimited) with a special character known as EOL (End of Line) '\n' character. In text files, some internal translations take place when this EOL character is read or written.
- A binary file is a file that contains information in the same format in which the information is held in memory, *i.e.*, the file content that is returned to us is raw (with no translation or no specific encoding).
- The open() function is used to open a data file in a program through a file-object (or a file handle).

- A file mode governs the type of operations (e.g., read/write/append) possible in the open file, i.e., it refers to how the file will be used once it has opened.
- A close() function breaks the link of the file-object and the file on the disk.
- A text file can be opened in these file modes: 'r', 'w', 'a', 'r+', 'w+', 'a+'.
- A binary file can be opened in these file modes: 'rb', 'wb', 'ab', 'r+b', ('rb+') 'w+b'('wb+'),'a+b'('ab+').
- The four file reading functions in Python are: read(), read(n), readline(), readlines().
- While read(n) reads some bytes from the file and returns it as a string, readline() reads a line at a time and returns it in the form of a list.
- The two writing functions for Python data files are write() and writelines().
- While write() writes a string in file, writelines() writes a list in a file.
- Every open file maintains a file-pointer to determine and control the position of read or write operation in file.
- Serialization is the process of converting a data structure/object that can be stored in non-string format and can be resurrected later. Serialization can also be called deflating the data while resurrecting can be called inflating it.
- Pickle module is used in serialization of data. This allows us to store data in binary form in the file. dump() and load() functions are used to write data and read data from the file.
- os module provides us various functions and attributes to work on files.
- Files in Python are interpreted as a sequence or stream of bytes stored on some storage media.
- The close() method of a file object flushes any unwritten information and closes the file object.
- The rename() method is used to rename the file or folder.
- seek() function is used to change the position of the file handle (file pointer) to a given specific location.
- File pointer is like a cursor which defines from where the data has to be read or written in a file.
- tell() returns the location of where the current file pointer is placed in the file.

## OBJECTIVE TYPE QUESTIONS

**1.** Fill in the blanks.

(a) ..... in Python are interpreted as a sequence or stream of bytes stored on some storage media.
(b) The ..... function creates a file object used to call other support methods associated with it.
(c) Files in Python can be opened in one of the three modes—....., ..... and ..... .
(d) The ..... method writes a list of strings to a file.
(e) The ..... method of a file object flushes any unwritten information and closes the file object.
(f) The name of the current working directory can be determined using ..... method.
(g) The ..... method is used to rename the file or folder.
(h) The ..... method is used to remove/delete a file.
(i) A ..... file is a series of 1's and 0's, treated as raw data and read byte-by-byte.
(j) The ..... statement automatically closes the file after the processing on the file gets over.
(k) The read() function reads data from the ..... of a file.
(l) The pickle module produces two main methods called ..... and ..... for writing and reading operations.
(m) The readlines() returns a list of strings from the file till ..... .
(n) The ..... method reads 'n' characters from the file.
(o) ..... function is used to force transfer of data from buffer to file.
(p) ..... format is a text format accessible to all applications across several platforms.
(q) ..... method is used for random access of data in a CSV file.
(r) ..... method of pickle module is used to write an object into binary file.
(s) ..... method of pickle module is used to read data from a binary file.
(t) ..... statement is given for importing csv module into your program.

(u) ..... is a string method that joins the string with a string separator.
(v) ..... object contains the number of the current line in a CSV file.
(w) To read all the file contents in the form of a list, ..... method is used.
(x) To write contents of list, tuple and sequence data type, ..... method may be used.
(y) To force Python to write the contents of file buffer on to storage file, ..... method may be used.
(z) When we work with file, ..... area is automatically associated with the file when we open it.

2. State whether the following statements are True or False.
(a) An absolute path always begins with the root folder.
(b) It is not necessary to always create the file in the same default folder where Python has been installed.
(c) In binary file, there is no delimiter for a line.
(d) A binary file stores information in ASCII or Unicode characters.
(e) readline() reads the entire file at a time.
(f) A close() function breaks the link of the file object and the file on the disk.
(g) When you open a file in read mode, the given file must exist in the folder; otherwise Python will raise FileNotFound error.
(h) The default file open mode is write mode.
(i) Opening a file in append mode erases the previous data.
(j) A file mode defines the type of operations that is to be performed on the file.
(k) A stream is defined as a sequence of characters.
(l) readlines() function returns a list of strings, each separated by "\n".
(m) Data maintained inside the file is termed as "persistent" (permanent in nature) data.
(n) with statement ensures that all the resources allocated to the file objects get deallocated automatically.
(o) CSV module can handle CSV files correctly regardless of the operating system on which the files were created.
(p) csv module gets automatically imported into your program for reading a CSV file.
(q) The type of operation that can be performed on a file depends upon the file mode in which it is opened.
(r) Functions readline() and readlines() are essentially the same.
(s) Every record in a CSV file is stored in reader object in the form of a list.
(t) writerow() method allows us to write a list of fields to the CSV file.
(u) Comma is the default delimiter for a CSV file.
(v) tell() method of Python tells us the current position within the file.
(w) The offset argument to seek() method indicates the number of bytes to be moved.
(x) If the offset value is set to 2, beginning of the file would be taken as seek position.

3. Multiple Choice Questions (MCQs)
(a) To open a file c:\test.txt for reading, we should give the statement:
(i) `file1= open("c:\ test.txt", "r")`
(ii) `file1 = open("c:\\ test.txt", "r")`
(iii) `file1 = open(file = "c:\ test.txt", "r")`
(iv) `file1 = open(file = "c:\\s test.txt", "r")`
(b) To open a file c:\ test.txt for writing, we should use the statement:
(i) `fobj = open("c:\test.txt", "w")`
(ii) `fobj = open("c:\\ test.txt", "w")`
(iii) `fobj = open(file = "c:\ test.txt", "w")`
(iv) `fobj = open(file = "c:\\ test.txt", "w")`
(c) To open a file c:\ test.txt for appending data, we can give the statement:
(i) `fobj = open("c:\\ test.txt", "a")`
(ii) `fobj = open("c:\\ test.txt", "rw")`
(iii) `fobj = open(file = "c:\test.txt", "w")`
(iv) `fobj = open(file = "c:\\ test.txt", "w")`

(d) Which of the following statements is/are true?
(i) When you open a file for reading, if the file does not exist, an error occurs.
(ii) When you open a file for writing, if the file does not exist, a new file is created.
(iii) When you open a file for writing, if the file exists, the existing file is overwritten with the new file.
(iv) All of the above.

(e) To read two characters from a file object fobj, the command should be:
(i) `fobj.read(2)` (ii) `fobj.read()`
(iii) `fobj.readline()` (iv) `fobj.readlines()`

(f) To read the entire contents of the file as a string from a file object fobj, the command should be:
(i) `fobj.read(2)` (ii) `fobj.read()`
(iii) `fobj.readline()` (iv) `fobj.readlines()`

(g) What will be the output of the following snippet?

```
f = None
for i in range(5):
    with open("data.txt", "w") as f:
        if i > 2:
            break
print(f.closed)
```

(i) True (ii) False (iii) None (iv) Error

(h) To read the next line of the file from a file object fobj, we use:
(i) `fobj.read(2)` (ii) `fobj.read()`
(iii) `fobj.readline()` (iv) `fobj.readlines()`

(i) To read the remaining lines of the file from a file object fobj, we use:
(i) `fobj.read(2)` (ii) `fobj.read()`
(iii) `fobj.readline()` (iv) `fobj.readlines()`

(j) The `readlines()` method returns:
(i) String (ii) A list of integers
(iii) A list of single characters (iv) A list of strings

(k) Which module is required to use the built-in function dump()?
(i) math (ii) flush (iii) pickle (iv) unpickle

(l) Which of the following functions is used to write data in the binary mode?
(i) write (ii) output (iii) dump (iv) send

(m) Which is/are the basic I/O (input-output) stream(s) in file?
(i) Standard Input (ii) Standard Output
(iii) Standard Errors (iv) All of these

(n) Which of the following is the correct syntax of file object 'fobj' to write sequence data type using writelines() function?
(i) `file.writelines(sequence)` (ii) `fobj.writelines()`
(iii) `fobj.writelines(sequence)` (iv) `fobj.writeline()`

(o) In file handling, what do the terms "r" and "a" stand for?
(i) read, append (ii) append, read
(iii) write, append (iv) None of these

(p) Which of the following is not a valid mode to open a file?
(i) ab (ii) rw (iii) r+ (iv) w+

(q) Which statement is used to change the file position to an offset value from the start?
(i) `fp.seek(offset, 0)` (ii) `fp.seek(offset, 1)`
(iii) `fp.seek(offset, 2)` (iv) None of these

(r) The difference between r+ and w+ modes is expressed as?
(i) No difference
(ii) In r+ mode, the pointer is initially placed at the beginning of the file and the pointer is at the end for w+
(iii) In w+ mode, the pointer is initially placed at the beginning of the file and the pointer is at the end for r+
(iv) Depends on the operating system

(s) What does CSV stand for?
(i) Cursor Separated Variables
(ii) Comma Separated Values
(iii) Cursor Separated Values
(iv) Cursor Separated Version

(t) Which module is used for working with CSV files in Python?
(i) random (ii) statistics (iii) csv (iv) math

(u) Which of the following modes is used for both writing and reading from a binary file?
(i) wb+ (ii) w (iii) wb (iv) w+

(v) Which statement is used to retrieve the current position within the file?
(i) fp.seek() (ii) fp.tell() (iii) fp.loc (iv) fp.pos

(w) What happens if no arguments are passed to the seek() method?
(i) file position is set to the start of file
(ii) file position is set to the end of file
(iii) file position remains unchanged
(iv) results in an error

(x) Which of the following modes will refer to binary data?
(i) r (ii) w (iii) + (iv) b

(y) Every record in a CSV file is stored in reader object in the form of a list using which method?
(i) writer() (ii) append() (iii) reader() (iv) list()

(z) For storing numeric data in a text file, it needs to be converted into ..... using ..... function.
(i) integer, int() (ii) integer, float() (iii) string, int() (iv) string, str()

## SOLVED QUESTIONS

**1.** What are the different file-processing modes supported by Python?

**Ans.** Python provides three modes to process files:
1. Read only mode
2. Write only mode
3. Read-write mode

**2.** How can you create Unicode string in Python?

**Ans.** Use "Unicode" or "u" before the string. For example, Unicode (text).

**3.** What is the difference between read() and readlines() function?

**Ans.** The read() function reads the entire data from a file in read mode and stores its contents in a string type variable.

The readlines() function, on the other hand, reads from a file in read mode and returns a list of strings of all lines in the file.

**4.** What is the difference between readline() and readlines() function?

**Ans.** The readline()function reads a line from a file in read mode till the line is terminated by end of line character '\n' and returns the next line in the file or a blank string if there are no more lines. (The returned data is of string type.)

The readlines() function also reads the entire lines from a file in read mode and returns a list of strings of all lines each separated by end of line character '\n' in the file. (The returned data is of list type.)

**5.** Write a single loop to display all the contents of a text file "file1.txt" after removing leading and trailing whitespaces.

**Ans.**
```
for line in open("file1.txt"):
    print(line.strip())
```

6. Write a function file_long() that accepts a filename and reports the file's longest line. [CBSE Sample Paper]

**Ans.**
```
def file_long(filename):
    longest = ""
    for line in open(filename):
        if len(line) > len (longest):
            longest = line
    print("Longest line's length =", len(longest))
    print(longest)
```

7. What is the output of the following code fragment? Explain.
```
>>> out = open("output.txt", "w")
>>> out.write("hello, world!\n")
>>> out.write("How are you?")
>>> out.close()
>>> open("output.txt").read()
```

**Ans.** The output will be:
```
'hello, world!
How are you?'
```
The first line of the code is opening the file in write mode; if the file 'output.txt' does exist, then it will overwrite, otherwise a new file is created. The next two lines write text to the file. out.close() function explicitly closes the file. The last line opens the file and from that reference reads the file content.

8. Write a function remove_lowercase() that accepts two file names, and copies all lines that do not start with a lower case letter from the first file to the second file.

**Ans.**
```
def remove_lowercase(infile, outfile):
    output = open(outfile, "w")
    for line in open(infile):
        if not line[0] in "abcdefghijklmnopqrstuvwxyz":
            output.write(line)
    output.close()
```

9. What is the output of the following code?
**Note:** We assume that file 'test1.txt' contains four lines of text.
```
f1=open("test1.txt", "r")
Size = len(f1.read())
print(f1.read(5))
```

**Ans.** No output.
*Explanation:* The f1.read() of line 2 will read the entire file content and place the file pointer at end of file. For f1.read(5), it will return nothing as there are no bytes to be read from EOF and, thus, print statement prints nothing.

10. Write a statement in Python to perform the following operations:
- To open a text file "BOOK.TXT" in read mode
- To open a text file "BOOK.TXT" in write mode

**Ans.**
```
f1 = open("BOOK.TXT",'r')
f2 = open("BOOK.TXT",'w')
```

11. Write a statement in Python to perform the following operations: [CBSE D 2015]
- To open a text file "MYPET.TXT" in write mode
- To open a text file "MYPET.TXT" in read mode

**Ans.**
```
f1 = open("MYPET.TXT",'w')
f2 = open("MYPET.TXT", 'r')
```

12. Write a method write1() in Python to write multiple lines of text contents into a text file 'daynote.txt'. [CBSE OD 2015]

**Ans.**
```
def write1():
    f = open("daynote.txt",'w')
    while True:
        line = input("Enter line:")
        f.write(line)
        choice =input("Are there more lines(Y/N):")
        if choice == 'N':
            break
    f.close()
```

13. Write a user-defined function countH() in Python that displays the number of lines starting with 'H' in the file "**Para.txt**". Example, if the file contains: [CBSE Sample Paper 2015]

Whose woods these are I think I know.
His house is in the village though;
He will not see me stopping here
To watch his woods fill up with snow.

**Output:** The line count should be 2.

**Ans.**
```
def countH():
    f = open("Para.txt", "r")
    lines =0
    l =f.readlines()
    for i in l:
        if i[0]=='H':
            lines+=1
    print("No. of lines are: ", lines)
    f.close()
```

14. Consider a binary file "**Employee.dat**" containing details such as: empno: ename: salary: (separator ':'). Write a Python function to display details of those employees who are earning between 20000 and 40000 (both values inclusive). [CBSE D 2017]

**Ans.**
```
def Readfile():
    i = open("Employee.dat","rb+")
    x = i.readline()
    while(x):
        I = x.split(':')
        if ((float(I[2])>=20000) and (float(I[2])<=40000)):
            print(x)
        x = i.readline()
    i.close()
```

15. Write a function countmy() in Python to read the text file "DATA.TXT" and count the number of times "my" occurs in the file. For example, if the file "DATA.TXT" contains—"This is my website. I have displayed my preferences in the CHOICE section."—the `countmy()` function should display the output as: "my occurs 2 times". [CBSE Sample Paper 2016]

**Ans.**
```
def countmy():
    f = open("DATA.txt","r")
    count = 0
    x = f.read()
    word= x.split()
    for i in word:
        if ( i == "my"):
            count = count + 1
    print("my occurs",count,"times")
    f.close()
```

16. Differentiate between file modes **r+** and **w+** with respect to text files in Python.

**Ans.**
- r+ opens a file for both reading and writing. The file pointer placed at the beginning of the file.
- w+ opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.

17. Write a method in Python to read lines from a text file "**DIARY.TXT**" and display those lines which start with the alphabet 'P'. [CBSE OD 2015]

**Ans.**
```
def display():
    file=open('DIARY.txt','r')
    line=file.readline()
    while line:
        if line[0]=='P' :
            print(line)
        line=file.readline()
    file.close()
```

18. Write definition of a method MSEARCH(STATES) to display all the state names from a list of STATES which start with the alphabet M.

For example:

If the list STATES contains

["MP","UP","WB","TN","MH","MZ","DL","BH","RJ","HR"]

The following should get displayed:

MP

MH

MZ

**Ans.**
```
def MSEARCH(STATES):
    for i in STATES:
        if i[0]=='M':
            print(i)
```

19. Differentiate between file modes r+ and rb+ with respect to files in Python.

**Ans.** r+ opens a file for both reading and writing. The file pointer is placed at the beginning of the file.
rb+ opens a file for both reading and writing in binary format. The file pointer is placed at the beginning of the file.

20. Write a method in Python to read lines from a text file "**MYNOTES.TXT**" and display those lines which start with the alphabet 'K'. [CBSE OD 2014]

**Ans.**
```
def display():
    file = open('MYNOTES.TXT','r')
    line = file.readline()
    while line:
        if line[0]=='K':
            print(line)
        line = file.readline()
    file.close()
```

21. Write a program using Dictionary and Text files to store roman numbers and find their equivalents. [CBSE OD 2015]

**Ans.**
```
import pickle
numerals = {1:'I', 4:'IV', 5:'V', 9:'IX', 10:'X', 40: 'XL', 50: 'L',
            90:'XC', 100:'C', 400:'CD', 500:'D',900: 'CM', 1000:'M'}
file1 = open("roman.log","wb")
pickle.dump(numerals,file1)
file1.close()
file2 = open("roman.log","rb")
num = pickle.load(file2)
file2.close()
```

```
n = 0
while n != -1:
    print("Enter 1,4,5,9,10,40,50,90,100,400,500,900,1000:")
    print("or enter -1 to exit")
    n = int(input("Enter number:"))
    if n != -1:
        print("Equivalent roman number of this numeral is:", num[n])
    else:
        print("Thank You")
```

**22.** Write a program to compute the Total salary of the employee and also calculate the size of the binary file named "**empfile.dat**". The file consists of the following fields: employee number, employee name, basic salary, allowances). **Hint:** (Total salary = basic + allowance)

**Ans.**

```
# Program to write and read employee records in a binary file
import pickle
print("WORKING WITH BINARY FILES")
bfile=open("empfile.dat","ab")
recno=1
print ("Enter Records of Employees")
print()
#taking data from user and dumping in the file as list object
while True:
      print("RECORD No.", recno)
      eno=int(input("\tEmployee number : "))
      ename=input("\tEmployee Name : ")
      ebasic=int(input("\tBasic Salary : "))
      allow=int(input("\tAllowances : "))
      totsal=ebasic+allow
      print("\tTOTAL SALARY : ", totsal)
      edata=[eno,ename,ebasic,allow,totsal]
      pickle.dump(edata,bfile)
      ans=input("Do you wish to enter more records (y/n)? ")
      recno=recno+1
      if ans.lower()=='n':
          print("Record entry OVER ")
          print()
          break
# retrieving the size of file
print("Size of binary file (in bytes):",bfile.tell())
bfile.close()
# Reading the employee records from the file using load() module
print("Now reading the employee records from the file")
print()
readrec=1
try:
      with open("empfile.dat","rb") as bfile:
          while True:
                edata=pickle.load(bfile)
                print("Record Number : ",readrec)
                print(edata)
                readrec=readrec+1
except EOFError:
       pass
bfile.close()
```

**23.** Write a program to display all the records in a file along with line/record number.

**Ans.**
```
f = open("result.dat", "r")
count = 0
rec = ""
while True :
    rec = f.readline(0)
    if rec == "" :
        break
    count = count + 1
print(count, rec)              #to suppress extra newline by print
f.close()
```

**24.** What is pickling and unpickling?

**Ans.** While working with a binary file, we need to store the data in a machine-readable format and to access the data in human-readable format, we need to use specific functions in Python. Python has a set of modules known as Pickle which enables you to store a specific object at some destination which you can call back at a later stage.

Pickling refers to the process of converting the structure to a byte stream (machine readable) before writing to a binary file while unpickling refers to a reverse process where the byte stream is converted to the original structure (human readable). Pickle module is used for implementing the binary files structure. dump() and load() functions are used for pickling and unpickling the binary files respectively.

**25.** What are the various read file operations in Python?

**Ans.** Python provides read(), read(n), readline() and readlines() functions to read a file.

read() reads entire file at once,
read(n) reads the specified number of characters,
readline() reads the complete line from the opened file,
readlines() returns a list where each element is a line in the file.

**26.** What are write file operations in Python?

**Ans.** The file can also be opened in write mode. Python provides write() to write a string in a text file. writelines() is used to write a sequence of lines at once.

**27.** Observe the following code and answer the questions that follow: [CBSE D 2014]

```
File = open("Mydata","a")
_____________________ #Blank1
File.close()
```

(i) What type (Text/Binary) of file is Mydata?

(ii) Fill the Blank 1 with statement to write "ABC" in the file "Mydata"

**Ans.** (i) Text File (ii) File.write("ABC")

**28.** Read the code given below and answer the question:

```
f1 = open("main.txt", "w")
f1.write("Bye")
f1.close()
```

If the file contains "GOOD" before execution, what will be the contents of the file after execution of this code?

**Ans.** The file would now contain "Bye" only, because when an existing file is opened in write mode ("w"), it truncates the existing data in the file.

**29.** How many file objects would you need to manage the following situations: [HOTS]

(a) To process four files sequentially. (b) To merge two sorted files into third file.

**Ans.** (a) Four objects are needed (b) Three objects are needed.

30. A given text file "data.txt" contains:

```
Line 1\n
\n
Line 3
Line 4
\n
Line 6
```

What would be the output of the following code?

```
f1=open("data.txt", "r")
lst = f1.readlines()
print(lst[0])
print(lst[2])
print(lst[5])
print(lst[1])
print(lst[4])
print(lst[3])
```

**Ans.**

```
Line 1
Line 3
Line 6


Line 4
```

31. Write code to print just the last line of a text file '**data.txt**' along with total lines in a text file.

**Ans.**

```
fin = open("data.txt", "r")
lineList = fin.readlines()
fin.close()
print("Last line=", lineList[-1],)
print("Total lines=", len(lineList))
```

32. Write a program that copies a text file "**source.txt**" onto "**target.txt**" barring the lines starting with a "@" sign.

**Ans.**

```
def filter(oldfile, newfile):
    fin = open(oldfile, "r")
    fout = open(newfile, "w")
    while True:
        text = fin.readline()
        if len(text) == 0:
            break
        if text[0] == "@":
            continue
        fout.write(text)
    fin.close()
    fout.close()
filter("source.txt", "target.txt")
```

33. Write a menu-driven program to perform read and write operations using a text file called "student.txt" containing student roll_no, name and address using two separate functions as given below:

- student_record(filename)—Entering student details
  While writing to a file (student.txt), the roll_no field will be separated from the remaining fields with a comma operator.
- student_readdata(filename)—Display student details
- student_search(filename)—Search a student on the basis of roll_no

**Ans.**
```
import os
filename = "student.txt"
def student_record(filename):
    if not os.path.isfile(filename):
        print("File not created")
    else:
        ch = 'y'
        print("Enter student detail:")
        with open (filename, 'a') as file1: #File gets opened in append mode
            while ch=='Y' or ch=='y':
                roll_no = input("Enter roll no.:")
                name = input("Enter name:")
                address = input("Enter address:")
                #Write record using a comma separator after first field roll_no
                file1.write(str(roll_no)+", "+name.upper()+" - "+address+"\n")
                file1.flush()
                ch = input("Want to add more records: ? <y/n>:")
                if ch=='Y' or ch=='y':
                    continue
                else:
                    break

def student_readdata(filename):
    if os.path.isfile(filename):
        with open(filename) as file1: #File gets opened in read mode
            print("Student information")
            print("---------------------------------------")
            for student in file1: #Accessing records one by one till EOF
                print(student, end = "")
    else:
        print("File does not exist")

def student_search(filename):
    if os.path.isfile(filename):
        with open(filename) as file1: #File gets opened in read mode
            roll_no = int(input("Enter roll no. to be searched:"))
            flag = False #To check whether record is not found
            for student in file1:
                str_len = len(student)
                i = 0
                str_roll_no = ""
                while True:
                    if student[i] == ",": #loop will break after the first
                        break #comma gets encountered
                    #Extracting roll number from Student records.
                    if student[i]>='0' and student[i]<='9':
                        str_roll_no = str_roll_no + student[i]
                        i += 1
                    #Str object converted into integer type
                    s_rollno = int(str_roll_no)
                    #Checks if the entered roll number matches with file data
                    if roll_no == s_rollno:
                        print("Student found: ",student, end="")
                        flag = True
                        break
```

```
            if flag == False:
                print("Record not found.")
        else:
            print("File does not exist")

    def get_menu():
        #print menu
        choice = int(input("Select a choice(1-4):"))
        return choice

    def main():
        '''Main Menu = Student Menu
        1) Add new student record
        2) Display student details
        3) Search student
        4) Save and Quit '''
        choice = get_menu()
        while choice !=4:
            if choice == 1:
                student_record(filename)
            elif choice == 2:
                student_readdata(filename)
            elif choice == 3:
                student_search(filename)
            else:
                print("Invalid choice, try again")
            choice = get_menu()

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

**34.** Write a function in Python to count the number of lines in a text file **'STORY.TXT'** which are starting with the alphabet **'A'**. [CBSE Sample Paper 2019-20]

**Ans.**
```
def COUNTLINES():
    file=open('STORY.TXT','r')
    lines = file.readlines()
    count=0
    for w in lines:
        if w[0]=="A" or w[0]=="a":
            count=count+1
            print("Total lines:", count)
    file.close()
```

**35.** Write a method/function DISPLAYWORDS() in Python to read lines from a text file **POEM.TXT** and display those words which are less than 4 characters.

**Ans.**
```
def DISPLAYWORDS():
    c=0
    file=open('POEM.TXT','r')
    line = file.read()
    word = line.split()
    for w in word:
        if len(w)<4:
            print(w)
    file.close()
```

**36.** Write a Python code to find out the size of the file in bytes, number of lines and number of words.

**Ans.**
```
# reading data from a file and find size, lines, words
f = open('Lines.txt', 'r')
str = f.read()
size = len(str)
print('size of file n bytes ', size)
f.seek(0)
L = f.readlines()
word = L.split()
print('Number of lines ', len(L))
print('Number of words ', len(word))
f.close()
```

**37.** Consider the following code:
```
f = open("test","w+")
f.write("0123456789abcdef")
f.seek(13)              #Statement 1
print(f.read(3))        #Statement 2
```
Explain statement 1 and give output of statement 2.

**Ans.** Statement 1 uses seek() method that can be used to position the file object at a particular place in the file.

Its syntax is:
```
fileobject.seek(offset [, from_what])
```
So, f.seek(13) positions the fileobject to 13 bytes before end of file.

Output of Statement 2 is:
```
def
```
It reads 3 bytes from where the file object is placed.

**38.** Write a program to know the cursor position and print the text according to below-given specifications:
- (a) Print the initial position
- (b) Move the cursor to 4th position
- (c) Display next 5 characters
- (d) Move the cursor to the next 10 characters
- (e) Print the current cursor position
- (f) Print next 10 characters from the current cursor position

**Ans.**
```
def program_rndom():
    f = open("merge.txt","r")
    print(f.tell())
    f.seek(4,0)
    print(f.read(5))
    f.seek(10,0)
    print(f.tell())
    print(f.seek(7,0))
    print(f.read(10))
program_rndom()
```

**39.** Yogendra intends to position the file pointer to the beginning of a text file. Write Python statement for the same assuming "F" is the Fileobject.

**Ans.**
```
F.seek(0)
```

**40.** Differentiate between file modes r+ and rb+ with respect to Python.

**Ans.** r+ opens a file for both reading and writing. The file pointer is placed at the beginning of the file.
rb+ opens a file for both reading and writing in binary format. The file pointer is placed at the beginning of the file.

**41.** In which of the following file modes will the existing data of the file not be lost?
rb, ab, w, w+b, a+b, wb, wb+, w+, r+

**Ans.** In file modes rb, ab, a+b and r+, data will not be lost.
In file modes w, w+b, wb, wb+ and w+, data will be truncated, *i.e.*, lost.

42. Write a statement in Python to perform the following operations:
(a) To open a text file "Book.txt" in read mode (b) To open a binary file "Book.dat" in write mode

Ans. (a) `f = open("Book.txt", "r")` (b) `f = open("Book.dat", "wb")`

43. What is a CSV file?

Ans. CSV (Comma Separated Values) is a simple file format used to store tabular data, such as a spreadsheet or database. A CSV file stores tabular data (numbers and text) in plain text.

44. What are the advantages of CSV file formats?

Ans. Advantages:
(a) A simple, compact and ubiquitous format for data storage.
(b) A common format for data interchange.
(c) It can be opened in popular spreadsheet packages like MS Excel, Calc, etc.
(d) Nearly all spreadsheets and databases support import/export to CSV format.

45. Differentiate between a text file, a binary file and a CSV file.

Ans.

| | Text File | Binary File | CSV File |
|---|---|---|---|
| 1. | It is capable of handling textual data. | It is capable of handling large files. | It has a very common format and is platform-independent. |
| 2. | It consists of a series of lines of a set of letters, numbers or symbols (string). | It consists of data with a specific pattern without any delimiter. | It consists of plain text with a list of data with a delimiter. |
| 3. | Any text editors like notepad can be used to read data. | No specific programs can be used to read data; Python provides functions to read data. | It can be read using text editors like notepad and spreadsheet software. |
| 4. | Every line ends with EOL. | There is no specific EOL character. | It terminates a line automatically when the delimiter is not used after data. |

46. Define a function to replace all spaces from text with – (dash).

Ans.
```
def rep_program():
    with open("merge.txt","r") as f1:
        data = f1.read()
        data=data.replace(' ','-')
    with open("merge.txt","w") as f1:
        f1.write(data)
rep_program()
```

47. Develop a Python code to insert records in g_meet.dat binary file until the user presses 'n'. The information is Google meeting id, Google meeting time and class.

Ans.
```
f1 = open("g_meet.dat","ab")
while True:
    gmeet_id=input("Enter id:")
    gmeet_time=input("Enter time:")
    gmeet_class =int(input("Enter google meet class:"))
    rec={"Google Meeting id":gmeet_id,"Google Meet Time":gmeet_time,
         "Google Meet Class":gmeet_class}
    pickle.dump(rec,f1)
    ch = input("Want more records:")
    ch=ch.lower()
    if ch=='n':
        break
f1.close()
```

**48.** Ranjan Kumar of Class 12 is writing a program to create a CSV file "user.csv" which will contain user name and password for some entries. He has written the following code. As a programmer, help him to successfully execute the given task. [CBSE Sample Question Paper 2020-21]

**Ans.**
```
import .....                                    # Line 1
def addCsvFile(UserName,PassWord):              # to write / add data into the CSV file
    f=open('user.csv',' ')                      # Line 2
    newFileWriter = csv.writer(f)
    newFileWriter.writerow([UserName,PassWord])
    f.close()                                   #csv file reading code
def readCsvFile():                              # to read data from CSV file
    with open('user.csv','r') as newFile:
    newFileReader = csv. (newFile)              # Line 3
    for row in newFileReader:
        print (row[0],row[1])
        newFile .....                           # Line 4
addCsvFile("Arjun","123@456")
addCsvFile("Arunima","aru@nima")
addCsvFile("Frieda","myname@FRD")
readCsvFile()                                   # Line5
```

**49.** (a) Name the module he should import in Line 1.
(b) In which mode should Ranjan open the file to add data into the file?
(c) Fill in the blank in Line 3 to read the data from a CSV file.
(d) Fill in the blank in Line 4 to close the file.
(e) Write the output he will obtain while executing Line 5.

**Ans.** (a) Line1 ; csv
(b) Line2 ; a
(c) Line3 ; reader
(d) Line4 ; close()
(e) Arjun 123@456
Arunima aru@nima
Frieda myname@FRD

**50.** Write a program to add (append) Employee records such as employee number, employee name, salary onto a CSV file.

**Ans.**
```
import csv
with open('myfile.csv',mode = 'a') as csvfile:
    mywriter = csv.writer(csvfile, delimiter = ',')
    ans = 'y'
    while ans.lower() == 'y':
        eno = int(input("Enter Employee Number:"))
        name = input("Enter Employee Name:")
        salary = int(input("Enter Employee Salary:"))
        mywriter.writerow([eno,name,salary])
        print("## Data Saved... ##")
        ans = input("Add More?")
```

**51.** Write user-defined functions to perform read and write operations onto a 'student.csv' file having fields roll number, name, stream and marks.

**Ans.**
```
import csv
row = ['2','Akshat Chauhan','Commerce','98']
def readcsv():
    with open("student.csv", 'r') as f:
        data = csv.reader(f)
        # reader function to generate a reader object
        for row in data:
            print(row)
```

```
def writecsv():
    with open("student.csv", 'w', newline='') as fobj:
        # write new record in file
        csv_w = csv.writer(fobj, delimiter=',')
        csv_w.writerow(row)
print("Press-1 to Read Data and Press-2 to Write data: ")
a = int(input())
if a == 1:
    readcsv()
elif a == 2:
    writecsv()
else:
    print("Invalid value")
```

**52.** A binary file "Book.dat" has structure [BookNo, Book_Name, Author, Price].
Write a user-defined function `CreateFile()` to input data for a record and add to 'Book.dat'.
Also write a function `CountRec(Author)` in Python which accepts the Author name as a parameter and count and return number of books by the given Author are stored in the binary file "Book.dat".
[CBSE Sample Question Paper 2020-21]

**Ans.**
```
import pickle
def createFile():
    fobj=open("Book.dat","ab")
    BookNo=int(input("Book Number : "))
    Book_name=input("Name :")
    Author = input("Author: ")
    Price = int(input("Price : "))
    rec=[BookNo,Book_Name,Author,Price]
    pickle.dump(rec,fobj)
    fobj.close()
def CountRec(Author):
    fobj=open("Book.dat","rb")
    num = 0
    try:
      while True:
          rec=pickle.load(fobj)
      if Author==rec[2]:
          num = num + 1
    except:
          fobj.close()
    return num
```

**53.** Write a Python program to read specific columns from a "department.csv" file and print the content of the columns, department id and department name.

**Ans.**
```
import csv
with open('department.csv', newline='') as csvfile:
    data = csv.reader(csvfile)
    print("Department Id and name")
    print("--------------------------------")
    for row in data:
        print(row[0], row[1])
```

**54.** Explain briefly the CSV format of storing files.

**Ans.** The acronym CSV is short for Comma Separated Values, which refers to tabular data saved as plain text where data values are separated by commas. In CSV format:

- Each row of the table is stored in one row, *i.e.*, the number of rows in a CSV file are equal to the number of rows in the table (or sheet or database table, etc.).
- The field values of a row are stored together with commas after every field value; but after the last field's value, no comma is given, just the end of line.

**55.** Write a menu-driven program implementing user-defined functions to perform different functions on a csv file "student" such as:

(a) Write a single record to csv. (b) Write all the records in one single go onto the csv.

(c) Display the contents of the csv file.

**Ans.** (a)

```
# To create a CSV File by writing individual lines
import csv
def CreateCSV1():
    # Open CSV File
    Csvfile = open('student.csv', 'w', newline='')
    # CSV Object for writing
    Csvobj = csv.writer(Csvfile)
    while True:
        Rno = int(input("Rno:"))
        Name = input("Name:")
        Marks = float(input("Marks:"))
        Line = [Rno, Name, Marks]
        # Writing a line in CSV file
        Csvobj.writerow(Line)
        Ch = input("More(Y/N)?")
        if Ch == 'N':
            break
    Csvfile.close()          # Closing a CSV File
```

(b)

```
# To create a CSV File by writing all lines in one go
def CreateCSV2():
    # Open CSV File
    Csvfile = open('student.csv', 'w', newline='')
    # CSV Object for writing
    Csvobj =csv.writer(Csvfile)
    Lines = []
    while True:
        Rno = int(input("Rno:"))
        Name = input("Name:")
        Marks = float(input("Marks:"))
        Lines.append([Rno, Name, Marks])
        Ch = input("More(Y/N)?")
        if Ch == 'N':
            break
    # Writing all lines in CSV file
    Csvobj.writerows(Lines)
    Csvfile.close()          # Closing a CSV File
```

(c)

```
# To read and show the content of a CSV File
def ShowAll():
    # Opening CSV File for reading
    Csvfile = open('student.csv', 'r', newline='')
    # Reading the CSV content in object
    Csvobj = csv.reader(Csvfile)
    for Line in Csvobj:      # Extracting line by line content
        print(Line)
    Csvfile.close()          # Closing a CSV File
```

```
print("CSV File Handling")
while True:
    Option = input("1:CreateCSV 2:CreateCSVAll 3:ShowCSV 4:Quit ")
    if Option == "1":
        CreateCSV1()
    elif Option == "2":
        CreateCSV2()
    elif Option == "3":
        ShowAll()
    else:
        break
```

56. (a) Write a program that reads a binary file "emp" and displays all the records of employees one by one.
    (b) Display the records of all those employees who are getting salaries between 25000 to 30000.

Ans. (a)
```
import pickle
f1 = open('emp.dat','rb')
e = pickle.load(f1)
for x in e:
    print(x)
f1.close()
```

(b)
```
import pickle
f1 = open('emp.dat','rb')
e = pickle.load(f1)
for x in e:
    if(e[x]>=25000 and e[x]<=30000):
        print(x)
f1.close()
```

57. Write the use of with statement.

Ans. **with statement** ensures that all the resources allocated to the file objects get deallocated automatically once we stop using the file. In the case of exceptions also, we are not required to close the file explicitly using **with** statement.

## UNSOLVED QUESTIONS

1. What is the difference between "w" and "a" modes?
2. What is the significance of file-object?
3. How are open() functions different from close() functions other than their functionality of opening and closing files?
4. Write statements to open a binary file 'C:\Myfiles\text1.dat' in read and write mode by specifying the file path in two different formats.
5. In which of the following file modes will the existing data of the file not be lost?
   (a) rb (b) ab (c) w
   (d) w + b (e) a+b (f) wb
   (g) wb+ (h) w+ (i) r+
6. What would be the data type of variable data in the following statements?
   (a) data = f.read() (d) data = f.read(10)
   (c) data = f.readline() (d) data = f.readlines()
7. When a file is opened for output, what happens when
   (a) the mentioned file does not exist? (b) the mentioned file exists?
8. What role is played by file modes in file operations? Describe the various file mode constants and their meanings.
9. Write a code snippet that will create an object called fileout for writing; associate it with the filename 'STRS'. The code should keep on writing strings to it as long as the user wants.

10. Explain how many file objects would you need to create to manage the following situations:
    (a) To process three files sequentially (b) To merge two sorted files into a third file.
11. What are the advantages of saving data in:
    (a) Binary form, (b) Text form?
12. Why is it required to close a file after performing the reading and writing operations on a file?
13. When do you think text files should be preferred over binary files?
14. Write a program that reads a text file and creates another file that is identical except that every sequence of consecutive blank spaces is replaced by a single space.
15. A file 'sports.dat' contains information in the following format: EventName,–Participant
    Write a function that would read contents from file 'sports.dat' and create a file named 'Athletic.dat' copying only those records from 'sports.dat' where the event name is "Athletics".
16. Write a program to count the words "to" and "the" present in a text file "Poem.txt".
17. Write a program to count the number of uppercase alphabets present in a text file "Poem.txt".
18. Write a program that copies one file to another. Have the program read the file names from user.
19. Write a program that appends the contents of one file to another. Have the program take the file names from the user.
20. Write a program that reads characters from the keyboard one by one. All lower case characters get stored inside the file 'LOWER', all upper case characters get stored inside the file 'UPPER' and all other characters get stored inside file 'OTHERS'.
21. Write a program to search the names and addresses of persons having age more than 30 in the data list of persons stored in a text file.
22. Write a function in Python to count and display the number of lines starting with alphabet 'A' present in a text file "LINES.TXT", *e.g.*, the file "LINE.TXT" contains the following lines:
    A boy is playing there.
    There is a playground.
    An aeroplane is in the sky.
    A cricket match is being played.
    The function should display the output as 3.
23. Write the file mode that will be used for opening the following files. Also, write the Python statements to open the following files: a text file "example.txt" in both read and write mode.
    (a) a binary file "bfile.dat" in write mode
    (b) a text file "try.txt" in append and read mode
    (c) a binary file "btry.dat" in read only mode.
24. Why is it advised to close a file after we are done with the read and write operations? What will happen if we do not close it? Will some error message be flashed?
25. What is the difference between the following sets of statements (a) and (b):
    (a)
    ```
    P = open("practice.txt","r")
    P.read(10)
    ```
    (b)
    ```
    with open("practice.txt", "r") as P:
        x = P.read()
    ```
26. Write a program to accept string/sentences from the user till the user enters "END". Save the data in a text file and then display only those sentences which begin with an uppercase alphabet.
27. Write a function to insert a sentence in a text file, assuming that text file is very big and can't fit in computer's memory.
28. Write a program to read a file 'Story.txt' and create another file, storing an index of 'Story.txt', telling which line of the file each word appears in. If word appears more than once, then index should show all the line numbers containing the word.
29. Write a program to accept a filename from the user and display all the lines from the file which contain Python comment character '#'.

30. Reading a file line by line from the beginning is a common task. What if you want to read a file backward? This happens when you need to read log files. Write a program to read and display content of file from end to beginning.

31. Write a Python program to display the size of a file after removing EOL characters, leading and trailing white spaces and blank lines.

32. Write a function Remove_Lowercase() that accepts two file names, and copies all lines that do not start with lowercase letter from the first file into the second file.

33. Write a program to display all the records in a file along with line/record number.

34. Write a method in Python to write multiple line of text contents into a text file "mylife.txt".

35. Write a method in Python to read the content from a text file "DIARY.TXT" line by line and display the same on the screen.

36. Write appropriate statements to do the following:
    (a) To open a file named "RESULT.DAT" for output.
    (b) To go to the end of the file at any time.

37. Write a program to add two more employees' details (empno, ename, salary, designation) to the file "emp.txt" already stored in disk.

38. How are the following codes different from one another?

(a)
```
fp = open("file.txt", 'r')
fp.read()
```

(b)
```
fp=open("file.txt", 'r')
```

39. What is the output of the following code fragment? Explain.

```
fout = open("output.txt", 'w')
fout.write("Hello, world!\n")
fout.write("How are you?")
fout.close()
f=open("output.txt")
print(f.read())
```

40. Write the output of the following code with justification if the contents of the file "ABC.txt" are:
"Welcome to Python Programming!"

```
f1 = open("ABC.txt", "r")
size = len(f1.read())
print(size)
data = f1.read(5)
print(data)
```

41. Anant has been asked to display all the students who have scored less than 40 for Remedial Classes. Write a user-defined function to display all those students who have scored less than 40 from the binary file "Student.dat".

42. Give the output of the following snippet:

```
import pickle
list1, list2 = [2, 3, 4, 5, 6, 7, 8, 9, 10], []
for i in list1:
    if (i%2==0 and i%4==0):
        list2.append(i)
f = open("bin.dat","wb")
pickle.dump(list2, f)
f.close()
f = open("bin.dat", "rb")
data = pickle.load(f)
f.close()
for i in data:
    print(i)
```

43. Following is the structure of each record in a data file named "PRODUCT.DAT".

```
{"prod_code": value, "prod_desc": value, "stock": value}
```

The values for prod_code and prod_desc are strings and the value for stock is an integer.
Write a function in Python to update the file with a new value of stock. The stock and the product_code, whose stock is to be updated, are to be inputted during the execution of the function.

44. Given a binary file "STUDENT.DAT", containing records of the following type:

```
[S_Admno, S_Name, Percentage]
```

Where these three values are:

`S_Admno` – Admission Number of student (string)
`S_Name` – Name of student (string)
`Percentage` – Marks percentage of student (float)

Write a function in Python that would read contents of the file "STUDENT.DAT" and display the details of those students whose percentage is above 75.

45. Write a statement to open a binary file C:\Myfiles\Text1.dat in read and write mode by specifying for file path in two different formats.

46. Write a statement in Python to perform the following operations: [CBSE D 2016]
(a) To open a text file "BOOK.TXT" in read and append mode
(b) To open a text file "BOOK.TXT" in write mode
(c) To open a text file "BOOK.TXT" in append mode

47. What is the following code doing?

```
import csv
File = open("contacts.csv", "a")
Name = input("Please enter name: ")
Phno = input("Please enter phone number: ")
data = [Name, Phno]
csvwriter = csv.writer(File)
csvwriter.writerow(data)
  file.close()
```

48. Write code to open the file in the previous question and print it in the following form:

Name : <name>  Phone: <phone number>

49. Consider the file "contacts.csv" created in the question above and figure out what the following code is trying to do?

```
name = input("Enter name:")
file = open("contacts.csv", "r")
for line in file:
    if name in line:
        print(line)
```

50. Write a program to enter the following records in a binary file:

| | |
|---|---|
| Item No | integer |
| Item_Name | string |
| Qty | integer |
| Price | float |

Number of records to be entered should be accepted from the user. Read the file to display the records in the following format:

Item No:
Item Name:
Quantity:
Price per item:

Amount: (to be calculated as Price * Qty)

51. Create a CSV file "Groceries" to store information of different items existing in a shop. The information is to be stored w.r.t. each item code, name, price, qty. Write a program to accept the data from user and store it permanently in CSV file.

## CASE-BASED/SOURCE-BASED INTEGRATED QUESTIONS

1. National Library has a wide collection of books for readers. This library provides a soothing environment for reading. The library wishes to update its records for any new books being purchased or added to the library. Write a Python program to add the records of new books in a file so that the records are stored for future retrieval and can be accessed whenever required.

**Ans.**

prog_lib_stock1.py - C:/Python382/prog_lib_stock1.py (3.8.2)

File Edit Format Run Options Window Help

```
#To add the details of books in National Library
file1 = open("D:/book.txt","a")
choice = 'y'
while choice == 'y':
    book_id =int(input("Enter book number"))
    book_name = input("Enter book name")
    author = input("Enter Author name")
    price = input("Enter Book price")
    book_rec = str(book_id) + "," +book_name+","+author+","+str(price)+"\n"
    file1.write(book_rec)
    print("Book Added")
    choice= input("Want to add more records :")
file1.close()
```

Ln: 18 Col: 0

```
>>>
========== RESTART: C:/Python382/prog_lib_stock1.py ==========
Enter book number 103
Enter book name Java Programming
Enter Author name abc
Enter Book price 460
Book Added
Want to add more records :y
Enter book number 104
Enter book name C#
Enter Author name Neil Rodricks
Enter Book price 700
Book Added
Want to add more records :n
```

**Contents of the file "book.txt" with records of newly added books**

book - Notepad

File Edit Format View Help

```
101, Working with Python, abc, 450
102, C++ Programming, E.Balagurusamy, 500
103, Java Programming, abc, 460
104, C#, Neil Rodricks, 700
```

2. Genesis Infotech plans to develop a computerized system for payroll processing of all its employees for storing salary details and furnishing relevant and required data in terms of total contribution towards its employees.

   Develop a Python program which stores data in "employee.csv", calculates and displays total salary remitted to its employees and to display the number of employees who are drawing a salary of more than ₹ 5000 per month.

**Ans.**

prog_case_csv1.py - C:/Python382/prog_case_csv1.py (3.8.2)

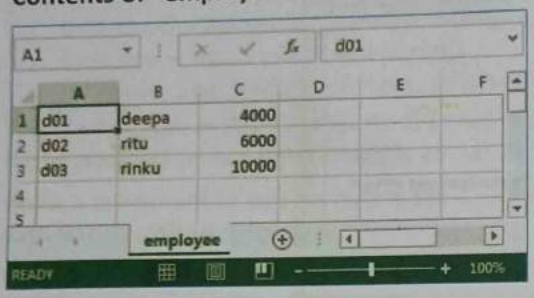
File Edit Format Run Options Window Help

```
import csv
with open("D:/employee.csv") as csvfile:
    csv_reader = csv.reader(csvfile,delimiter=',')
    count = 0
    sum = 0
    print("EMPNO","\t\t","EMPNAME","\t\t","SALARY")
    print("-------------------------------------------------------")
    for rec in csv_reader:
        print(rec[0],"\t\t",rec[1],"\t\t",rec[2])
        sum +=int(rec[2])
        if (int(rec[2])) > 5000:
            count += 1
    print("-------------------------------------------------------")
    print("\t Sum of Salary of all employees: ",sum)
    print("Employee Getting Salary > 5000 :",count)
    print("-------------------------------------------------------")
```

Ln: 26 Col: 4

```
>>>
========== RESTART: C:/Python382/prog_case_csv1.py ==========
EMPNO           EMPNAME         SALARY
d01             deepa           4000
d02             ritu            6000
d03             rinku           10000
Sum of Salary of all employees:  20000
Employee Getting Salary > 5000 : 2
```

**Contents of "employee.csv"**




3. New Horizon Technology Institute stores the details of the students enrolled with it. A binary file "STUDENT.DAT" has the structure—admission_number, Name, Percentage. Write a function countRec() in Python that would read contents of the file "STUDENT.DAT" and display the details of those students whose percentage is above 75. Also display the number of students scoring above 75%.

**Ans.**