PADHAAII

COMPUTER SCIENCE With python

PADHAAII.COM
WE MAKE EXAM EASY

Textbook for
Class XII

- Programming & Computational Thinking
- Computer Networks
- Data Management (SQL, Django)
- Society, Law and Ethics

SUMITA ARORA

DHANPAT RAI & Co.

padhaaii

5

173 – 200

File Handling

5.1 Introduction 173
5.2 Data Files 174
5.3 Opening and Closing Files 174
5.3.1 Opening Files 174
5.3.2 Closing Files 177
5.4 Reading and Writing Files 177
5.4.1 Reading from Files 177
5.4.2 Writing onto Files 183
5.4.3 The flush( ) Function 187
5.4.4 Removing Whitespaces after Reading from File 187
5.4.5 Significance of File Pointer in File Handling 189
5.5 Standard Input, Output and Error Streams 190

padhaaii

5 File Handling

In This Chapter

5.1 Introduction
5.2 Data Files
5.3 Opening and Closing Files
5.4 Reading and Writing Files
5.5 Standard Input, Output and Error Streams

5.1 INTRODUCTION

Most computer programs work with files. This is because files help in storing information permanently. Word processors create document files ; Database programs create files of information ; Compilers read source files and generate executable files. So, we see, it is the files that are mostly worked with, inside programs. *A file in itself is a bunch of bytes stored on some storage device like hard-disk, thumb-drive etc.* Every programming language offers some provision to use and create files through programs. Python is no exception and in this chapter, you shall be learning to work with data files through Python programs.

padhaaii

PADHAAII

5.2 DATA FILES

The data files are the files that store data pertaining to a specific application, for later use. The data files can be stored in *two* ways :

- Text files
- Binary files

1. Text Files

A text file stores information in ASCII or Unicode characters (the one which is default for your programming platform). In text files, each line of text is terminated, (delimited) with a special character known as EOL (End of Line) character. In text files, some internal translations take place when this EOL character is read or written. In Python, by default, this EOL character is the newline character ('\n') or carriage-return, newline combination ('\r\n').

2. Binary Files

A binary file is just 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 you is raw (with no translation or no specific encoding). In binary file, there is no delimiter for a line. Also no translations occur in binary files. As a result, binary files are faster and easier for a program to read and write than are text files. As long as the file doesn't need to be read by people or need to be ported to a different type of system, binary files are the best way to store program information.

5.3 OPENING AND CLOSING FILES

In order to work with a file from within a Python program, you need to open it in a specific mode as per the file manipulation task you want to perform. The most basic file manipulation tasks include *adding, modifying* or *deleting data* in a file, which in turn include any one or combination of the following operations :

- reading data from files
- writing data to files
- appending data to files

Python provides built-in functions to perform each of these tasks. But before you can perform these functions on a file, you need to first open the file.

5.3.1 Opening Files

In data file handling through Python, the first thing that you do is open the file. It is done using *open( )* function as per one of the following syntaxes :

```
<file_objectname> = open(<filename>)
<file_objectname> = open(<filename>, <mode>)
```

For example, `myfile = open("taxes.txt")` ← *Python will look for this file in current working directory*

The above statement opens file **"taxes.txt"** in **file mode** as ***read mode*** (default mode) and attaches it to ***file object namely myfile.***

Consider another statement :

```
file2 = open("data.txt", "r")
```

The above statement opens the file **"data.txt"** in ***read mode*** (because of "r" given as mode) and attaches it to file object namely ***file2***.

padhaaii

PADHAAII

Consider one more file-open statement :

```
file3 = open("e:\\main\\result.txt", "w")
```

Python will look for this file in E:\main folder

The above statement opens file **"result.txt"** (stored in folder **E:\ main**) in write mode (because of "w" given as mode) and attaches it to file object namely ***file3***.

(Notice that in above file open statement, the file path contains double slashes in place of single slashes.)

NOTE

A ***file-object*** *is* also known as ***file-handle***.

The above given *three* file-open statements must have raised these questions in your mind :

(*i*) *What is file-object* ? (*ii*) *What is mode or file-mode* ?

The coming lines will have answers to all your questions, but for now, let us summarize the file *open( )* function.

- Python's *open( )* function creates a *file object* which serves as a link to a file residing on your computer.
- The first parameter for the *open( )* function is a path to the file you'd like to open. If just the file name is given, then Python searches for the file in the current folder.
- The second parameter of the open function corresponds to a mode which is typically read ('r'), write ('w'), or append ('a'). If no second parameter is given, then by default it opens it in read ('r') *mode*.

NOTE

Please note that when you open a file in *readmode*, the given file must exist in the folder, otherwise Python will raise **FileNotFoundError**.

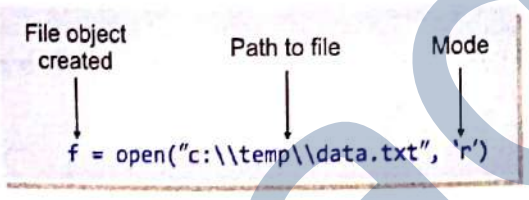


Figure 5.1 Working of file open( ) function.

Figure 5.1 summarizes the *open( )* function of Python for you.

As you can see in Fig. 5.1, the slashes in the path are doubled. This is because the slashes have special meaning and to suppress that special meaning escape sequence for slash *i.e.*, \\ is given.

However, if you want to write with single slash, you may write in raw string as :

with r here, you can give single slashes in pathnames

```
f = open(r "c :\temp\data.txt", "r")
```

The prefix *r* in front of a string makes it ***raw string*** that means there is no special meaning attached to any character Remember, following statement might give you incorrect result :

NOTE

The prefix ***r*** in front of a string makes it ***raw string*** that means there is no special meaning attached to any character.

```
f = open("c :\temp\data.txt", "r")
```

This might give incorrect result as \t is tab character

Reason being that '\t' will be treated as **tab character** and '\d' on some platforms as numeric-digit or some error.

Thus the *two* ways to give paths in filenames correctly are :

(*i*) Double the slashes *e.g.*,

```
f = open("c:\\temp\\data.txt", "r")
```

NOTE

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

padhaaii

(ii) Give **raw string** by prefixing the file-path string with an *r e.g.*,

```
f = open(r"c:\temp\data.txt", "r")
```

NOTE

If you just give the file name without its path (*e.g.*, just "data.txt" rather than "c :\\temp\\data.txt"), Python will open/create the file in the same directory in which the module file is stored.

5.3.1A File Object/File Handle

File objects are used to read and write data to a file on disk. The file object is used to obtain a reference to the file on disk and open it for a number of different tasks.

File object (also called ***file-handle***) is very important and useful tool as through a *file-object* only, a Python program can work with files stored on hardware. All the functions that you perform on a data file are performed through file-objects.

FILE OBJECT

A **file object** is a reference to a file on disk. It opens and makes it available for a number of different tasks.

When you use file **open( )**, Python stores the reference of mentioned file in the file-object. A file-object of Python is a stream of bytes where the data can be read either byte by byte or line by line or collectively. All this will be clear to you in the coming lines (under topic – *File access modes*).

5.3.1B File Access Modes

When Python opens a file, it needs to know the file-mode in which the file is being opened. A file-mode governs the type of operations (such as *read* or *write* or *append*) possible in the opened file *i.e.*, it refers to how the file will be used once it's opened. File modes supported by Python are being given in Table 5.1.

Table 5.1 *File-modes*

Text File Mode	Binary File Mode	Description	Notes
'r'	'rb'	read only	❖ File must exist already, otherwise Python raises I/O error.
'w'	'wb'	write only	❖ If the file does not exist, file is created. ❖ If the file exists, Python will truncate existing data and over-write in the file. So this mode must be used with caution.
'a'	'ab'	append	❖ File is in **write only** mode. ❖ If the file exists, the data in the file is retained and new data being written will be appended to the end. ❖ If the file does not exist, Python will create a new file.
'r+'	'r+b' or 'rb+'	read and write	❖ File must exist otherwise error is raised. ❖ Both reading and writing operations can take place.
'w+'	'w+b' or 'wb+'	write and read	❖ File is created if does not exist. ❖ If file exists, file is truncated (past data is lost). ❖ Both reading and writing operations can take place.
'a+'	'a+b' or 'ab+'	write and read	❖ File is created if does not exist. ❖ If file exists, file's existing data is retained ; new data is appended. ❖ Both reading and writing operations can take place.

Adding a ***'b'*** to *text-file mode* makes it *binary-file mode*.

padhaaii

PADHAAII

To create a file, you need to open a file in a mode that supports *write* mode (*i.e.*, 'w', or 'a' or 'w+' or 'a+' modes).

NOTE

A **file-mode** governs the type of operations (*e.g.*, read/write/append) possible in the opened file *i.e.*, it refers to how the file will be used once it's opened.

5.3.2 Closing Files

An open file is closed by calling the ***close( )*** method of its file-object. Closing of file is important. In Python, files are automatically closed at the end of the program but it is good practice to get into the habit of closing your files explicitly. Why ? Well, the operating system may not write the data out to the file until it is closed (this can boost performance). What this means is that if the program exits unexpectedly there is a danger that your precious data may not have been written to the file! So the moral is : once you finish writing to a file, close it.

The *close( )* function accomplishes this task and it takes the following general form :

```
<fileHandle>.close( )
```

For instance, if a file **Master.txt** is opened *via* file-handle ***outfile***, it may be closed by the following statement :

```
outfile.close( )
```

The close( ) must be used with filehandle

NOTE

A close( ) function breaks the link of file-object and the file on the disk. After close( ), no tasks can be performed on that file through the file-object (or file-handle).

Please remember, **open( )** is a built-in function (used standalone) while **close( )** is a method used with file-handle object.

5.4 READING AND WRITING FILES

Python provides many functions for reading and writing the open files. In this section, we are going to explore these functions. Most common file reading and writing functions are being discussed in coming lines.

5.4.1 Reading from Files

Python provides mainly *three* types of *read functions* to read from a data file. But before you can read from a file, the file must be opened and linked via a ***file-object*** or ***file handle***. Most common file reading functions of Python are listed below in Table 5.2.

Table 5.2 *Python data files — reading writing functions*

S.No.	Method	Syntax	Description

1.	read( )	`<filehandle>.read([n])`	reads at most ***n*** bytes ; if no ***n*** is specified, reads the entire file.
			Returns the read bytes in the form of a *string*.
			`In [11]: file1 = open("E:\\mydata\\info.txt")`
			`In [12]: readInfo = file1.read(15)`
			`In [13]: print(readInfo)`
			`It's time to wo` — 15 bytes read
			`In [14]: type(readInfo)`
			`Out[14]: str` — Bytes read into **string** type

padhaaii

PADHAAII

S.No.	Method	Syntax	Description

2.	readline( )	<filehandle>.readline([n])	reads a line of input ; if *n* is specified reads at most *n* bytes.
			Returns the read bytes in the form of a *string* ending with ln(line) character or returns a blank string if no more bytes are left for reading in the file.
			`In [20]: file1 = open("E:\\mydata\\info.txt")`
			`In [21]: readInfo = file1.readline()`
			`In [22]: print(readInfo)`
			`It's time to work with files.` 1 line read

3.	readlines( )	<filehandle>.readlines()	reads all lines and returns them in a *list*
			`In [23]: file1 = open("E:\\mydata\\info.txt")`
			`In [24]: readInfo = file1.readlines()`
			`In [25]: print(readInfo)`
			`["It's time to work with files.\n", 'Files offer and ease and power to store your work/data/information for later use.\n', 'simply create a file and store(write) in it .\n', 'Or open an existing file and read from it.\n']` All lines read
			`In [26]: type(readInfo)`
			`Out[26]: list` Read into list type

The <filehandle> in above syntaxes is the file-object holding open file's reference.

Let us consider some examples now. For the examples and explanations below, we are using a file namely *poem.txt* storing the content shown in Fig. 5.2.

WHY ?

We work, we try to be better
We work with full zest
But, why is that we just don't know any letter.

We still give our best.
We have to steal,
But, why is that we still don't get a meal.

We don't get luxury,
We don't get childhood,
But we still work,
Not for us, but for all the others.

Why is it that some kids wear shoes, BUT we make them ?

by Mythili, class 5

Figure 8.2 Contents of a sample file *poem.txt*

padhaaii

PADHAAII

Code Snippet 1 *Reading a file's first 30 bytes and printing it*

File-object created →

```
myfile = open(r'E:\poem.txt', "r")
str = myfile.read( 30 )
print(str)
```

File-object being used →

See the number of bytes to be read specified as argument of read( )

When we specify number of bytes with read( ), Python will read only the specified number of bytes from the file. The next read( ) will start reading onwards from the last position read. ***Code snippet*** 2 illustrates this fact.

The output produced by above code is :

```
>>>
    WHY?
We work, we try
```

TIP

You may combine the file( ) or open( ) with the file-object's function, if you need to perform just a single task on the open file.

If need to perform just a single task with the open file, then you may combine the two steps of opening the file and performing the task, *e.g.*, following statement :

```
>>> file(r"E :\poem.txt", "r").read(30)
```

First function will open the file and the second function will perform with the result of first function, i.e., the reference of open file

would give the same result as that of above code.

For example, the following code will return the first line of file *poem.txt* :

```
open("poem.txt", "r").readline()
```

Code Snippet 2 *Reading n bytes and then reading some more bytes from the last position read*

```
myfile = open(r'E:\poem.txt', 'r')
str = myfile.read( 30 )
print(str)
str2 = myfile.read( 50 )
print(str2)
myfile.close()
```

reading 30 bytes

reading next 50 bytes

The output produced by above code is :

```
>>>
    WHY?
We work, we try

 to be better
We work with full zest
But, why is t
```

Output by first print statement, i.e., ***print(str)***

print *has entered a new line after its output i.e., here*

Output by second print statement, i.e., ***print(str2)***

padhaaii

PADHAAII

If you do not want to have the newline characters inserted by print statement in the output then use **end = ' '** argument in the end of *print( )* statement so that print does not put any additional newline characters in the output. Refer to ***code snippets 3 and 4*** below.

Code Snippet 3 *Reading a file's entire content*

```
myfile = open(r'E :\poem.txt', "r")
str = myfile.read()
print(str)
myfile.close()
```

See when no value is specified as read()'s argument, entire file is read

The output produced by above code is :

```
>>>
    WHY?

We work, we try to be better
We work with full zest
But, why is that we just don't know any letter.

We still give our best.
We have to steal,
But, why is that we still don't get a meal.

We don't get luxury,
We don't get childhood,
But we still work,
Not for us, but for all the others.

Why is it that some kids wear shoes, BUT we make them ?

by      Mythili, class 5
```

Code Snippet 4 *Reading a file's first three lines - line by line*

```
myfile = open(r'E:\poem.txt', "r")
str = myfile.readline()
print(str, end = ' ')
str = myfile.readline()
print(str, end = ' ')
str = myfile.readline()
print(str, end = ' ')
myfile.close()
```

*Argument **end = ' '** at the end of print( ) statement will ensure that output shows exact content of the data file and no print-inserted newline characters are*

The *readline( )* function will read a full line. A line is considered till a newline character (EOL) is encountered in the data file.

The output produced by above code is :

```
>>>
    WHY?
```

Line 1

Line 2

```
We work, we try to be better
```

Line 3

padhaaii

PADHAAII

Code Snippet 5 *Reading a complete file - line by line*

```
myfile = open(r'E:\poem.txt', "r")
str = " "          #initially storing a space (any non-None value)
while str :
    str = myfile.readline( )
    print(str, end = ' ')
myfile.close( )
```

The output produced by the above code will print the entire content of file **poem.txt**.

```
>>>
    WHY?

We work, we try to be better
We work with full zest
But, why is that we just don't know any letter.

We still give our best.
We have to steal,
But, why is that we still don't get a meal.

We don't get luxury,
We don't get childhood,
But we still work,
Not for us, but for all the others.

Why is it that some kids wear shoes, BUT we make them ?

by      Mythili, class 5
```

The ***readline( )*** function reads the leading and trailing spaces (if any) along with trailing newline character('\n') also while reading the line. You can remove these leading and trailing white spaces (spaces or tabs or newlines) using ***strip( )*** (without any argument) as explained below. Recall that ***strip( )*** without any argument removes leading and trailing whitespaces.

There is another way of printing a file line by line. This is a simpler way where after opening a file you can browse through the file using its file handle line by line by writing following code :

```
<filehandle> = open(<filename>, [<any read mode>])
for <var> in <filehandle> :
    print(<var>)
```

For instance, for the above given file *poem.txt*, if you write following code, it will print the entire file line by line :

```
myfile = open(r'E:\poem.txt', "r")
for line in myfile :
    print(line)
```

The output produced by above code is just the same as the output produced by above program. The reason behind this output is that when you iterate over a file-handle using a for loop, then

padhaaii

PADHAAII

the for loop's variable moves through the file line by line where a line of a file is considered as a sequence of characters up to and including a special character called the newline character (\n). So the *for loop's* variable starts with first line and with each iteration, it moves to the next line. As the *for loop* iterates through each line of the file the loop variable will contain the current line of the file as a string of characters.

Code Snippet 6 *Displaying the size of a file after removing EOL characters, leading and trailing white spaces and blank lines*

```
myfile = open(r'E:\poem.txt', "r")
str1 = " "                #initially storing a space (any non-None value)
size = 0
tsize = 0
while str1 :
    str1 = myfile.readline()
    tsize = tsize + len(str1)
    size = size + len(str1.strip())
print("Size of file after removing all EOL characters & blank lines:", size)
print("The TOTAL size of the file:", tsize)
myfile.close()
```

The output produced by the above code fragment is :

```
Size of file after removing all EOL characters & blank lines : 360
The TOTAL size of the file : 387
```

All the above code fragments read the contents from file in a string. However, if you use re*adlines( )* (notice 's' in the end of the function), then the contents are read in a List. Go through next code fragment.

Code Snippet 7 *Reading the complete file-in a list*

```
myfile = open(r'E:\poem.txt', "r")
s = myfile.readlines()
print(s)
myfile.close()
```

*notice it is **readlines( )** not **readline( )***

Now carefully look at the output. The ***readlines( )*** has read the entire file **in a list** of strings where each line is stored as one string :

```
['  WHY?\n', '\n', 'We work, we try to be better\n', 'We work with
full zest\n', "But, why is that we just don't know any letter.\n",
'\n', 'We still give our best.\n', 'We have to steal,\n', "But, why
is that we still don't get a meal.\n", '\n', "We don't get luxury,
\n", "We don't get childhood,\n", 'But we still work,\n', 'Not for
us, but for all the others.\n', '\n', 'Why is it that some kids wear
shoes, BUT we make them ?\n', '\n', 'by     Mythili, class 5\n']
```

padhaaii

PADHAAII

Now that you are familiar with these reading functions' working, let us write some programs.

Program 5.1 Write a program to display the size of a file in bytes.

```
myfile = open(r'E:\poem.txt', "r")
str = myfile.read()
size = len(str)
print("Size of the given file poem.txt is")
print(size, "bytes")
```

```
>>>
Size of the given file poem.txt is
387 bytes
```

Program 5.2 Write a program to display the number of lines in the file.

```
myfile = open(r'E:\poem.txt', "r")
s = myfile.readlines()
linecount = len(s)
print("Number of lines in poem.txt is", linecount)
myfile.close()
```

```
>>>
Number of lines in poem.txt is 18
```

5.4.2 Writing onto Files

After working with file-reading functions, let us talk about the writing functions for data files available in Python. (See Table 5.3). Like reading functions, the writing functions also work on open files, *i.e.*, the files that are opened and linked via a *file-object* or *file-handle*.

Table 5.3 *Python Data Files – Writing Functions*

S.No.	Name	Syntax	Description
1.	write( )	`<filehandle>.write(str1)`	writes string *str1* to file referenced by ***<filehandle>***
2.	writelines( )	`<filehandle>.writelines(L)`	writes all strings in list **L** as lines to file referenced by ***<filehandle>***
The `<filehandle>` in above syntaxes is the *file-object* holding open file's reference			

Appending a File

When you open a file in "**w**" or ***write mode***, Python overwrites an existing file or creates a non-existing file. That means, for an existing file with the same name, the earlier data gets lost. If, however, you want to write into the file while retaining the old data, then you should open the file in "**a**" or ***append mode***. A file opened in *append mode* retains its previous data while allowing you to add newer data into. You can also add a plus symbol (+) with file read mode to facilitate reading as well as writing.

That means, in Python, writing in files can take place in following forms :

(*i*) In an existing file, while retaining its content

(*a*) if the file has been opened in append mode ("a") to retain the old content.

(*b*) if the file has been open in 'r+' or 'a+' modes to facilitate reading as well as writing.

padhaaii

PADHAAII

(*ii*) to create a new file or to write on an existing file after truncating/ overwriting its old content

(*a*) if the file has been opened in write-only mode ("w")

(*b*) if the file has been open in 'w+' mode to facilitate writing as well as reading

(*iii*) Make sure to use **close( )** function on ***file-object*** after you have finished writing as sometimes, the content remains in memory buffer and to force-write the content on file and closing the link of ***file-handle*** from file, *close( )* is used.

Let us consider some examples now.

Code Snippet 8 *Create a file to hold some data*

```
fileout = open("Student.dat", "w")
for i in range(5) :
    name = input("Enter name of student :")
    fileout.write(name)
fileout.close()
```

The write() will simply write the content in file without adding any extra character.

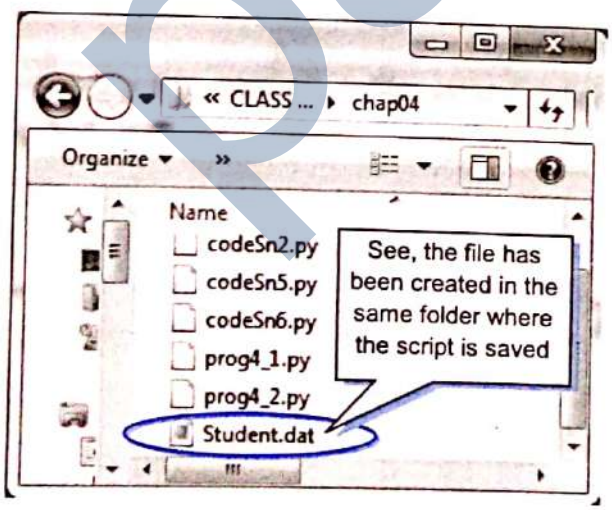
It's important to use close()

The sample run of above code is as shown below :

```
>>>
Enter name of student : Riya
Enter name of student : Rehan
Enter name of student : Ronaq
Enter name of student : Robert
Enter name of student : Ravneet
```

Now you can see the file created in the same folder where this Python script/program is saved (see Fig. 5.3(*a*) below). However, if you want to create the file in specific folder then you must specify the file-path as per the guidelines discussed earlier.

Also, you can open the created file ("student.dat" in above case) in *Notepad* to see its contents. Fig. 5.3(*b*) shows the contents of file created through *code snippet 5*.



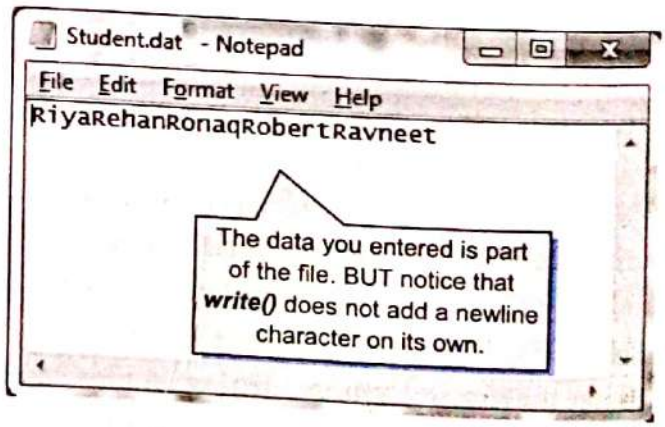


Figure 5.3 (*a*) File created through open( ) with "w" mode is stored in the same folder as that of script file (*b*) The write( ) function does not add any extra character in the file.

padhaaii

PADHAAII

IMPORTANT

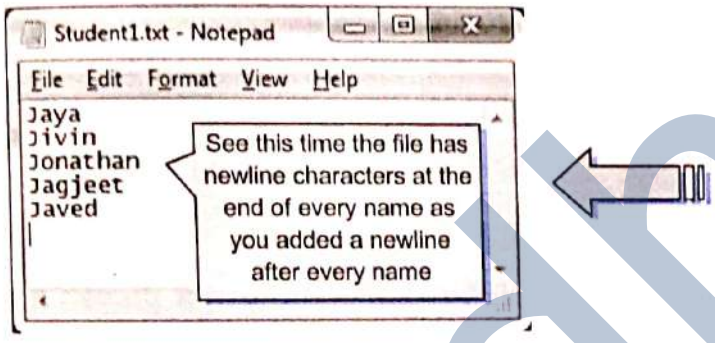
Carefully look at Fig. 5.3(*b*) that is showing contents of a file created through **write( )**. It is clear from the file-contents that ***write( )*** does not add any extra character like newline character('\n') after every write operation. Thus to group the contents you are writing in file, line wise, make sure to write newline characters on your own as depicted in ***code snippet 9***.

Code Snippet 9 *Create a file to hold some data, separated as lines*

(This code is creating a different file than created with code snippet 8)

```
fileout = open("Student1.txt", "w")
for i in range(5) :
    name = input("Enter name of student :")
    fileout.write(name)
    fileout.write('\n')
fileout.close()
```

The newline character '\n' written after every name



The sample run of the above code is as shown below :

```
>>>
Enter name of student : Jaya
Enter name of student : Jivin
Enter name of student : Jonathan
Enter name of student : Jagjeet
Enter name of student : Javed
```

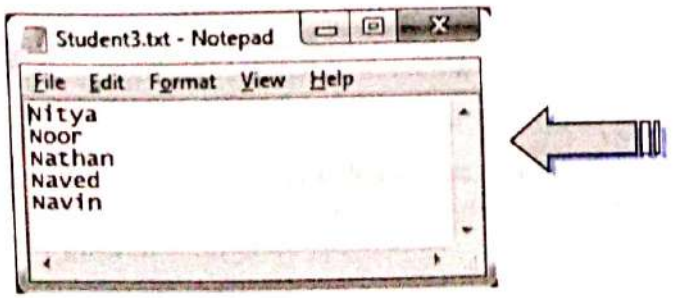
The file created by ***code snippet 9*** is shown above.

Code snippet 10 *Creating a file with some names separated by newline characters without using write( ) function.*

*(For this, we shall use **writelines()** inplace of **write()** function which writes the content of a list to a file. Function writelines() also, does not add newline character, so you have to take care of adding newlines to your file.)*

```
fileout = open("Student3.txt", "w")
List1 = []
for i in range(5) :
    name = input("Enter name of student :")
    List1.append(name + '\n')
fileout.writelines(List1)
fileout.close()
```

Responsibility to add newline character is of programmer's



Sample run of the above code is as shown below :

```
>>>
Enter name of student : Nitya
Enter name of student : Noor
Enter name of student : Nathan
Enter name of student : Naved
Enter name of student : Navin
```

padhaaii

Now that you are familiar with these writing functions' working, let us write some programs based on the same.

Program 5.3 Write a program to get roll numbers, names and marks of the students of a class (get from user) and store these details in a file called "Marks.det".

```
count = int(input("How many students are there in the class?"))
fileout = open ("Marks.det", "w")

for i in range(count) :
    print("Enter details for student", (i+1), "below:")
    rollno = int(input("Rollno:))"
    name = input("Name :")
    marks = float(input("Marks:"))
    rec = str(rollno) + "," + name + "," + str(marks) +'\n'
    fileout.write(rec)
fileout.close()
```

Joining individual information by adding commas in between

```
>>>
How many students are there in the class? 3
Enter details for student 1  below :
Rollno : 12
Name : Hazel
Marks : 67.75
Enter details for student 2  below :
Rollno : 15
Name : Jiya
Marks : 78.5
Enter details for student 3  below :
Rollno : 16
Name : Noor
Marks : 68.9
```

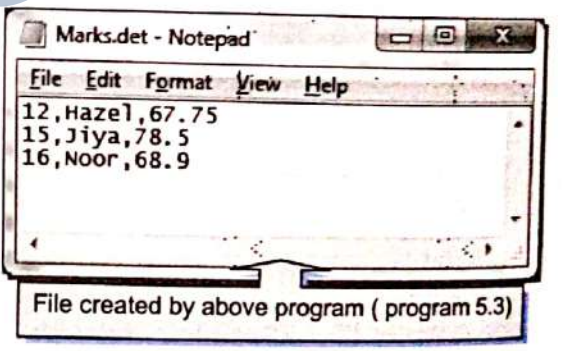


Figure 5.4 File created through program.

If you carefully notice, we have created comma separated values in one student record while writing in file. So we can say that the file created by program 5.4 is in CSV (comma separated values) format or it is a delimited file where comma is the delimiter.

Program 5.4 Write a program to add two more students' details to the file created in program 5.3.

```
fileout = open ("Marks.det", "a")

for i in range(2) :
    print("Enter details for student", (i+1), "below :")
    rollno = int(input("Rollno:"))
    name = input("Name :")
    marks = float(input("Marks:"))
    rec = str(rollno) + "," + name + "," + str(marks) +'\n'
    fileout.write(rec)
fileout.close()
```

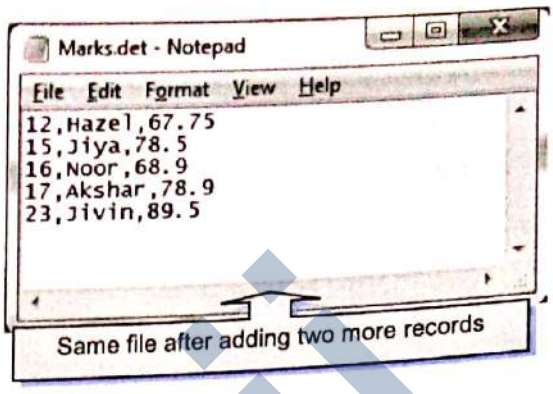
Notice the file is opened in append mode ("a") this time so as to retain old content

We want to add two records this time

padhaaii

PADHAAII


```
>>>
Enter details for student 1 below :
Rollno : 17
Name : Akshar
Marks : 78.9
Enter details for student 2 below :
Rollno : 23
Name : Jivin
Marks : 89.5
```



Program 5.5 Write a program to display the contents of file "Marks .det" created through programs 5.3 and 5.4.

```
fileinp = open("Marks.det", "r")
while str :
    str = fileinp.readline()
    print(str)
fileinp.close()
```

```
>>>
12,Hazel,67.75
15,Jiya,78.5
16,Noor,68.9
17,Akshar,78.9
23,Jivin,89.5
```

5.4.3 The flush( ) Function

When you write onto a file using any of the write functions, Python holds everything to write in the file in buffer and pushes it onto actual file on storage device a later time. If however, you want to force Python to write the contents of buffer onto storage, you can use **flush( )** function.

Python automatically flushes the files when closing them *i.e.*, this function is implicitly called by the **close( )** function. But you may want to flush the data before closing any file. The syntax to use flush( ) function is :

```
<fileObject>.flush()
```

FLUSH( )

The flush( ) function forces the writing of data on disc still pending in *output buffer*.

Consider the following example code :

```
f = open( 'out.log', 'w+')
f.write( 'The output is \n')
f.write( "My" + "work-status "+" is ")
f.flush()
s = 'OK.'
f.write(s)
f.write( '\n')
# some other work
f.write( 'Finally Over\n')
f.flush()
f.close()
```

With this statement, the strings written so far, i.e., 'The output is' and 'My work-status is' have been pushed on to actual file on disk.

These write statements' strings may still be pending to be written on to disk

The flush( ) function ensures that whatever is held in Output buffer, is written on to the actual file on disk

5.4.4 Removing Whitespaces after Reading from File

The ***read( )*** and ***readline( )*** functions discussed above, read data from file and return it in string form and the ***readlines( )*** function returns the entire file content in a list where each line is one item of the list.

padhaaii

PADHAAII

All these read functions also read the leading and trailing whitespaces *i.e.*, spaces or tabs or newline characters. If you want to remove any of these trailing and leading whitespaces, you can use strip( ) functions [rstrip( ), lstrip( ) and strip( )] as shown below.

Recall that :

- the *strip*( ) removes the given character from both ends.
- the *rstrip*( ) removes the given character from trailing end *i.e., right end.*
- the *lstrip*( ) removes the given character from leading end *i.e., left end.*

To understand this, consider the following examples :

1. **Removing EOL '\n' character from the line read from the file.**

```
fh = file("poem.txt, "r")
line = fh.readline()
line = line.rstrip('\n')
```

*Will remove the end of line **newline** character '\n'*

```
>>> fh = file("e :\\poem.txt", "r")
>>> line = fh.readline()
>>> len(line)
14

>>> line2 = line.rstrip('\n')
>>> len(line2)
13

>>> line3 = line.strip()
>>> len(line3)
4
```

2. **Removing the leading whilespaces from the line read from the file**

```
line = file("poem.txt, "r").readline()
line = line.lstrip()
```

Now can you justify the output of following code that works with first line of file *poem.text* shown above where first line containing leading 8 spaces followed by word 'WHY?' and a '\n' in the end of line.

Steps to Process a File

Following lines list the steps that need to be followed in the order as mentioned below. The five steps to use files in your Python program are :

1. Determine the type of file usage

Under this step, you need to determine whether you need to open the file for reading purpose (input type of usage) or writing purpose (output type of usage). If the data is to be brought in from a file to memory, then the file must be opened in a mode that supports reading such as "r" or "r+" etc.

Similarly, if the data (after some processing) is to be sent from memory to file, the file must be opened in a mode that supports writing such as "w" or "w+" or "a" etc.

2. Open the file and assign its reference to a *file-object* or file-handle

Next, you need to open the file using **open( )** and assign it to a file-handle on which all the file-operations will be performed. Just remember to open the file in the file-mode that you decided in step 1.

3. Now process as required

As per the situation, you need to write instructions to process the file as desired. For example, you might need to open the file and then read it one line at a time while making some computation, and so on.

4. Close the file

This is very important step especially if you have opened the file in write mode. This is because, sometimes the last lap of data remains in buffer and is not pushed on to disk until a close( ) operation is performed.

padhaaii

5.4.5 Significance of File Pointer in File Handling

Every file maintains a *file pointer* which tells the current position in the file where writing or reading will take place. (A file pointer in this context works like a book-mark in a book).

Whenever you read something from the file or write onto a file, then these *two* things happen involving file-pointer :

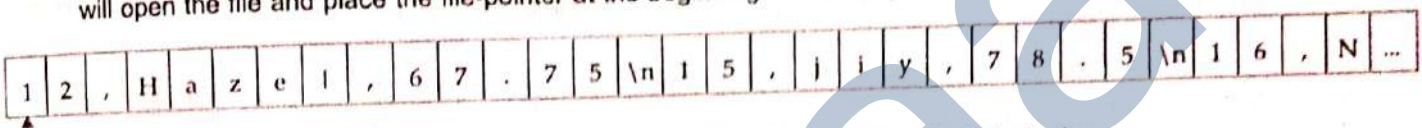
(*i*) this operation takes place at the position of *file-pointer* and

(*ii*) *file-pointer* advances by the specified number of bytes

Figure 5.5 illustrates this process. It is important to understand how a *file pointer* works in Python files. The working of file-pointer has been described in Fig. 5.5.

1. `fh = open("Marks.det, "r")`

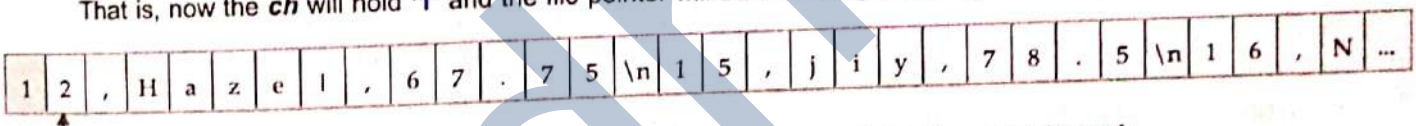
will open the file and place the file-pointer at the beginning of the file (see below)



position of file-pointer when file is opened in read mode ("r"), i.e., file-pointer lies in beginning

2. `ch = fh.read(1)`

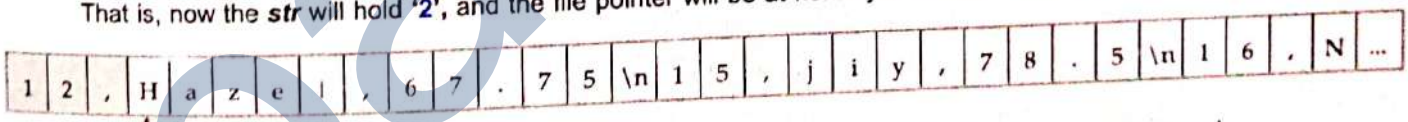
will read 1 byte from the file from the position the file-pointer is currently at ; and the file pointer advances by one byte. That is, now the ***ch*** will hold **'1'** and the file pointer will be at next byte holding value '2' (see below)



See now, the file-pointer has advanced by 1 position as previous read operation read 1 byte only.
Now the next read will take place at the current position of file-pointer.

3. `str = fh.read(2)`

will read 2 bytes from the file from the position the file-pointer is currently at and the file pointer advances by 2 bytes. That is, now the ***str*** will hold **'2'**, and the file pointer will be at next byte holding value 'H' (see below)



See now, the file-pointer has advanced by 2 positions as previous read operation read 2 bytes only.
Now the next read will take place at the current position of file-pointer.

Figure 5.5 Working of a file-pointer.

5.4.5A File Modes and Opening Position of File-Pointer

The position of a file-pointer is governed by the *filemode* it is opened in. Following table lists the opening position of a file-pointer as per *filemode*.

Table 5.4 *File modes and opening position of file-pointer*

File Modes	Opening position of file - pointer
r, rb, r+, rb+, r+b	*beginning of the file*
w, wb, w+, wb+, w+b	*beginning of the file* (Overwrites the file if the file exists).
a, ab, a+, ab+, a+b	*at the end of the file* if the file exists otherwise creates a new file.

padhaaii

PADHAAII

5.5 STANDARD INPUT, OUTPUT AND ERROR STREAMS

If someone asks you to give input to a program interactively or by typing, you know what device you need for it – ***the keyboard***. Similarly, if someone says that the output is to be displayed, you know which device it will be displayed on – ***the monitor***. So, we can safely say that the ***Keyboard*** is the **standard input device** and the ***monitor*** is **standard output device**. Similarly, any error if occurs is also displayed on the monitor. So, *the* ***monitor*** is also **standard error device**. That is,

- standard input device (**stdin**) – reads from the keyboard
- standard output device (**stdout**) – prints to the display and can be redirected as standard input.
- standard error device (**stderr**) – Same as *stdout* but normally only for errors. Having error output separately allows the user to divert regular output to a file and still be able to read error messages.

Do you know internally how these devices are implemented in Python? These standard devices are implemented as files called *standard streams*. In Python, you can use these standard stream files by using **sys** module. After importing, you can use these standard streams (**stdin, stdout** and **stderr**) in the same way as you use other files.

Interesting : Standard Input, Output Devices as Files

If you import **sys** module in your program then, **sys.stdin.read( )** would let you read from keyboard. This is because the keyboard is the *standard input device* linked to **sys.stdin**. Similarly, **sys.stdout.write( )** would let you write on the standard output device, *the monitor*. **sys.stdin** and **sys.stdout** are standard input and standard output devices respectively, treated as files.

Thus **sys.stdin** and **sys.stdout** are like files which are opened by the Python when you start Python. The **sys.stdin** is always opened in read mode and **sys.stdout** is always opened in write mode. Following code fragment shows you interesting use of these. It prints the contents of a file on monitor without using **print** statement :

These statements would write on file/device associated with sys.stdout, which is the monitor

```
import sys
fh = open(r"E:\poem.txt")
line1 = fh.readline()
line2 = fh.readline()
sys.stdout.write(line1)
sys.stdout.write(line2)
sys.stderr.write("No errors occurred\n")
```

Output produced is :

```
>>> ========================
>>>
        WHY ?
We work, we try to be better
No errors occurred
>>>
```

*See, **stderr** also displayed its text on monitor.*

PADHAAII

Python Data Files : with Statement

Python's **with** statement for files is very handy when you have two related operations which you'd like to execute as a pair, with a block of code in between. The syntax for using with statement is :

```
with open(<filename>, <filemode>) as <filehandle> :
    <file manipulation statements>
```

The classic example is opening a file, manipulating the file, then closing it :

```
with open('output.txt', 'W') as f:
    f.write('Hi there!')
```

The above **with** statement will automatically close the file after the nested block of code. The advantage of using a ***with statement*** is that it is guaranteed to close the file no matter how the nested block exits. Even if an exception (a runtime error) occurs before the end of the block, it will close the file.

Absolute and Relative Paths

Full name of a file or directory or folder consists of path\primaryname.extension.

Path is a sequence of directory names which give you the hierarchy to access a particular directory or file name. Let us consider the following directory structure :

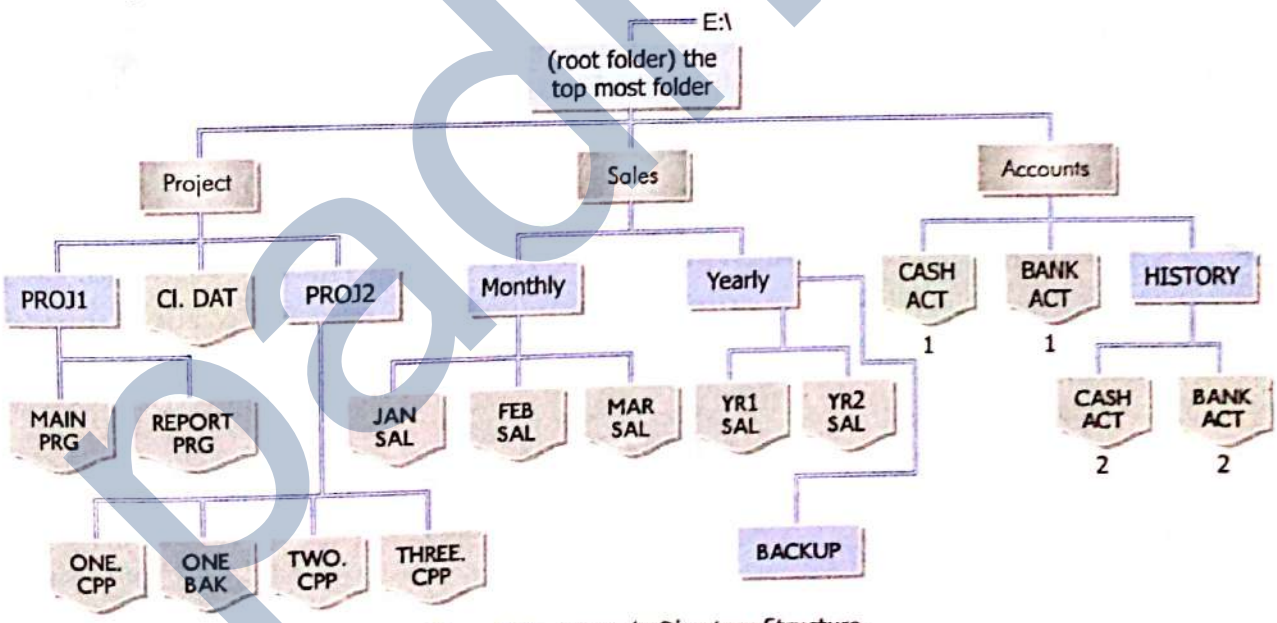


Figure 5.7 A Sample Directory Structure.

Now the *full name* of directories PROJECT, SALES and ACCOUNTS will be

`E:\PROJECT`, `E:\SALES` and `E:\ACCOUNTS` respectively.

So format of path can be given as

```
Drive-letter:\directory [\directory...]
```

where **first \ (backslash) refers to root directory** and other ('\'s) separate a directory name from the previous one.

padhaaii

Now the directory BACKUP'S path will be :

```
E:\SALES\YEARLY\BACKUP
```

As to reach BACKUP the sequence one has to follow is : under drive E, under root directory (first \), under SALES subdirectory of root, under YEARLY subdirectory of SALES, there lies BACKUP directory.

Similarly full name of ONE.VBP file under PROJ2 subdirectory will be :

```
E:\ PROJECT\PROJ2\ONE.VBP
  ↓
refers to
root directory
```

PATHNAME

The full name of a file or a directory is also called *pathname*.

Now see there are *two* files with the same name CASH.ACT, one under ACCOUNTS directory and another under HISTORY directory but according to Windows rule no two files can have same names (*path names*). Both these files have same names but their path names differ, therefore, these can exist on system. Therefore, $CASH.ACT_1$'s path will be

```
E:\ACCOUNTS\CASH.ACT
```

and $CASH.ACT_2$'s path will be

```
E:\ACCOUNTS\HISTORY\CASH.ACT
```

Above mentioned *path names* are ***Absolute Pathnames*** as they mention the *paths* from the top most level of the directory structure.

NOTE

The absolute paths are from the topmost level of the directory structure. The relative paths are relative to **current working directory** denoted as a **dot(.)** while its **parent directory** is denoted with two dots(..).

Relative pathnames mention the paths relative to current working directory. Let us assume that current working directory now is, say, PROJ2. The symbols . (*one dot*) and .. (*two dots*) can be used now in relative paths and pathnames. The symbol . can be used in place of current directory and .. denotes the parent directory.

So, with PROJ2 as current working folder, pathname of TWO.DOC will be :

```
.\TWO.DOC
```

TWO.DOC in current folder

(PROJ2 being working Folder)

which means under current working folder, there is file TWO.PAS. Similarly, path name for file CL.DAT will be

```
..\CL.DAT
```

CL.DAT in parent folder

(PROJ2 being working Folder)

which means under parent folder of current folder, there lies a file CL.DAT. Similarly, path name for REPORT.PRG will be

```
..\PROJ1\REPORT.PRG
```

(PROJ2 being working Folder)

that is under parent folder's subfolder PROJ1, there lies a file REPORT.PRG.

Check Point 5.1

1. In which of the following file modes, the existing data of file will not be lost ?
   (*a*) 'rb' (*b*) ab (*c*) w
   (*d*) w + b (*e*) 'a + b' (*f*) wb
   (*g*) wb+ (*h*) w+ (*i*) r+
2. What would be the data type of variable data in following statements ?
   (*a*) data = f.read.( )
   (*b*) data = f.read.(10)
   (*c*) data = f.readline( )
   (*d*) data = f.readlines( )
3. How are following statements different ?
   (*a*) f.readline( )
   (*b*) f.readline( ).rstrip( )
   (*c*) f.readline( ).strip( )
   (*d*) f.readline.rstrip('\n')

padhaaii

PADHAAII

DATA FILES IN PYTHON

PriP

Progress In Python 5.1

This PriP session aims at giving you practical exposure to file handling in Python.

Fill it in PriP 5.1 under Chapter 5 of practical component-book – Progress in Computer Science with Pythonafter practically doing it on the computer.

>>>❖<<<

LET US REVISE

- *A file in itself is a bunch of bytes stored on some storage devices like hard-disk, thumb-drive etc.*
- *The 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) character. In text files some internal translations take place when this EOL character is read or written.*
- *A binary file is just 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 you 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 opened file i.e., it refers to how the file will be used once it's opened.*
- *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 three file reading functions of Python are : read( ), readline( ), readlines( )*
- *While read( ) reads some bytes from the file and returns it as a string, readline( ) reads a line at a time and readlines( ) reads all the lines from the file 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.*
- *The input and output devices are implemented as files, also called standard streams.*
- *There are three standard streams : stdin (standard input), stdout (standard output) and stderr (standard error)*
- *The absolute paths are from the topmost level of the directory structure. The relative paths are relative to current working directory denoted as a dot(.) while its parent directory is denoted with two dots(..).*

Solved Problems

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

   Solution. The read( ) reads from a file in read mode, and stores its contents in a string type variable. The readlines( ) function, reads from a file in read mode and returns a list of all lines in the file.

2. *Differentiate between the following :* [CBSE D 2015]

   (*i*) `f = open('diary.txt', 'r')` (*ii*) `f = open('diary.txt', 'w')`

   Solution. (*i*) File has been opened in *read mode* with file handle *f*.

   (*ii*) File has been opened in *write mode* with file handle *f*.

padhaaii

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

Solution. The ***readline( )*** function reads from a file in read mode 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 from a file in read mode and returns a **list** of all lines in the file. (The returned data is of list type)

4. *Write a single loop to display all the contents of a text file e :\poem.txt after removing leading and trailing whitespaces.*

Solution.
```
for line in file("poem.txt") :
    print(line.strip())
```

5. *Write a function stats( ) that accepts a filename and reports the file's longest line.*

Solution.
```
def stats(filename) :
    longest = " "
    for line in file(filename) :
        if len(line) > len(longest) :
            longest = line
    print("Longest line's length =", len(longest))
    print(longest)
```

6. *What is the output of following code fragment ? Explain.*

```
out = file("output.txt", "w")
out.write("Hello, world!\n")
out.write("How are you?")
out.close()
file("output.txt").read()
```

Solution. The output will be :

```
'Hello, world!\nHow are you?'
```

The first line of the code is opening the file in write mode ; the next two lines write text to the file. The last line opens the file and from that reference reads the file-content. Function file( ) does the same as that of open( ). Thus file("output.txt") will give the reference to open file, on which read( ) is applied.

7. *Write a function remove_lowercase( ) that accepts two filenames, and copies all lines that do not start with a lowercase letter from the first file into the second.*

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

8. *What is the output of following code ?*

```
file("e:\\poem.txt", "r").readline().split()
```

Recall that poem.txt has some leading and trailing whitespaces.

Solution. `['WHY?']`

padhaaii

9. *What is the output of following code ?*

```
file("e:\\poem.txt", "r").readline()
```

Solution. ' WHY?\n'

10. *What is the output of following code ?*

```
fh = file("poem.txt", "r")
size = len(fh.read())
print(fh.read(5))
```

Solution. No output

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

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

Solution.

```
fh = open("Result.det", "r")
count = 0
rec = " "
while True :
    rec = fh.readline()
    if rec == " " :
        break
    count = count + 1
print(count, rec, end = '')          # to suppress extra newline by print
fh.close()
```

12. *What is the output produced by following code ?*

```
obj = open("New.txt","w")
obj.write("A poem by Paramhansa Yogananda) "
obj.write("Better than Heaven or Arcadia")
obj.write("I love thee, O my India!")
obj.write("And thy love I shall give")
obj.write("To every brother nation that lives.")
obj.close()
obj1=open("New.txt","r")
s1 = obj1.read(48)
print(s1)
obj1.close()
```

Solution. The output produced by above code will be :

```
A poem by Paramhansa Yogananda Better than Heaven
```

13. *The file "New.txt" contains the following :*

```
Better than Heaven or Arcadia
I love thee, O my India!
And thy love I shall give
To every brother nation that lives.
```

padhaaii

PADHAAII

Considering the given file, what output will be produced by the following code ?

```
obj1=open("New.txt","r")
s1 = obj1.readline()
s2 = obj1.readline()
s3 = obj1.readline()
s4 = obj1.read(15)
print(s4)
obj1.close()
```

Solution. The output produced by above code is : `And thy love I`

14. *Two identical files (p1.txt and p2.txt) were created by following two codes (carefully go through the two codes given below)*

(*a*)
```
obj = open("p1.txt","w")
obj.write("Better than Heaven or Arcadia")
obj.write("I love thee, O my India!")
obj.write("And thy love I shall give")
obj.write("To every brother nation that lives.")
obj.close( )
```

(*b*)
```
obj = open("p2.txt","w")
obj.write("Better than Heaven or Arcadia\n")
obj.write("I love thee, O my India!\n")
obj.write("And thy love I shall give\n")
obj.write("To every brother nation that lives.\n")
obj.close( )
```

What would be the output produced if the files are read and printed with following code.

Solution. The output produced by code (a) will be :

```
Better than Heaven or ArcadiaI love thee, O my India!And thy love
I shall giveTo every brother nation that lives.
```

The output produced by code (b) will be :

```
A poem by Paramhansa Yogananda
Better than Heaven or Arcadia
I love thee, O my India!
And thy love I shall give
To every brother nation that lives.
```

15. *Considering the two files p1.txt and p2.txt created in previous question, what output will be produced by following code fragments ?*

(*a*)
```
obj1 = open("p1.txt","r")
s1 = obj1.readline()
s2 = obj1.read(15)
print(s2)
obj1.close()
```

(*b*)
```
obj1 = open("p2.txt","r")
s1 = obj1.readline()
s2 = obj1.read(15)
print(s2)
obj1.close()
```

Solution. No output or blank output will be produced by code (*a*).

For code(*b*), the output produced will be : `Better than Hea`

padhaaii

16. *Consider the file p2.txt created above. Now predict the output of following code that works with p2.txt. Explain the reason behind this output.*

```
fp1 = open("p2.txt", "r")
print(fp1.readline(20))
s1 = fp1.readline(30)
print(s1)
print(fp1.readline(25))
```

Solution. The output produced by above code will be :

```
A poem by Paramhansa
    Yogananda
Better than Heaven or Arc
```

The reason behind this output is that the first file-read line (*i.e.*, **fp1.readline(20))** read 20 bytes from the file pointer. As just after opening the file, the file-pointer is at the beginning of the file, the 20 bytes are read from the beginning of the file which returned string as "A poem by Paramhansa\n" – this is because readline( ) returns the read string by adding an end-line character to it (\n).

So the first line of output was printed as : A poem by Paramhansa

After the first readline( ), the file pointer was at the space following word 'Paramhansa', so the next readline( ) started reading from there and read 15 character or end-of the-line, whichever is earlier. So the read string was **"Yogananda\n"** – notice the space before word Yogananda. Hence came the second line of the output.

Now the file-pointer was at the beginning of the third line and the next readline (*i.e.*, **fp1.readline(25)**) read 25 characters from this line and gave the last line of output.

17. *A text file contains alphanumeric text (say an.txt). Write a program that reads this text file and prints only the numbers or digits from the file.*

Solution.

```
F = open("an.txt", "r")
for line in F:
    words = line.split()
    for i in words:
        for letter in i:
            if(letter.isdigit()):
                print(letter)
```

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

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

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

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

19. *A given text file "data.txt" contains :*

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

padhaaii

PADHAAII

What would be the output of following code?

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

Solution.

```
Line 1
 Line 3
 Line 6  Line 3

 Line 4
```

20. *Write code to print just the last line of a text file "data.txt".*

Solution.

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

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

Solution.

```
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")
```

GLOSSARY

File	A bunch of bytes stored on some storage device.
File mode	A constant describing how a file is to be used.
Stream	A sequence of bytes.

Assignment

Type A : Short Answer Questions/Conceptual Questions

1. What is the difference between "w" and "a" modes ?
2. What is the significance of file-object ?
3. How is file open( ) function different from close( ) function ?
4. Write statements to open a binary file C:\Myfiles\Text1.txt in read and write mode by specifying the file path in two different formats.
5. When a file is opened for output, what happens when
   (*i*) the mentioned file does not exist (*ii*) the mentioned file does exist ?
6. What role is played by file modes in file operations ? Describe the various file mode constants and their meanings.
7. What are the advantages of saving data in : (*i*) binary form (*ii*) text form ?
8. When do you think text files should be preferred over binary files ?
9. Write a statement in Python to perform the following operations : [CBSE D 16]
   (*a*) To open a text file "BOOK.TXT" in read mode (*b*) To open a text file "BOOK.TXT" in write mode

padhaaii

PADHAAII

Type B : Application Based Questions

1. How are following codes different from one another ?

(*a*)
```
my_file = open('poem.txt','r')
my_file.read()
```

(*b*)
```
my_file = open('poem.txt','r')
my_file.read(100)
```

2. If the file 'poemBTH.txt' contains the following poem (by Paramhans Yoganand) :

God made the Earth;
Man made confining countries
And their fancy-frozen boundaries.
But with unfound boundless love
I behold the borderland of my India
Expanding into the World.
Hail, mother of religions, lotus, scenic beauty,and sages!

Then what outputs will be produced by both the code fragments given in question1.

3. Consider the file poemBTH.txt given above (in previous question). What output will be produced by following code fragment ?

```
obj1 = open("poemBTH.txt","r")
s1 = obj1.readline()
s2.readline(10)
s3 = obj1.read(15)
print(s3 )
print(obj1.readline())
obj1.close()
```

4. Consider the file "poemBTH.txt" and predict the outputs of following code fragments if the file has been opened in filepointer file1 with code:

```
file1 = open("E:\\mydata\\poemBTH.txt","r+")
```

(*a*)
```
print("A. Output 1")
print(file1.read())
print()
```

(*b*)
```
print("B. Output 2")
print(file1.readline() )
print()
```

(*c*)
```
print("C. Output 3")
print(file1.read(9))
print()
```

(*d*)
```
print("D. Output 4")
print(file1.readline(9) )
```

(*e*)
```
print("E. Output of Readlines function is")
print(file1.readlines() )
print()
```

NOTE. Consider the code fragments in succession, *i.e.*, code (*b*) follows code (*a*), which means changes by code (*a*) remain intact. Similarly, code (*c*) follows (*a*) and (*b*), and so on.

5. What is following code doing ?

```
file = open("contacts.csv", "a")
name = input("Please enter name.")
phno = input("Please enter phone number.")
file.write(name + "," + phno + "\n")
```

6. Write code to open file created in previous question and print it in following form :

```
Name : <name>      Phone :<phone number>
```

padhaaii

PADHAAII

7. Consider the file "contacts.csv" created in above question 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)
```

8. Consider the file poemBTH.txt and predict the output of following code fragment. What exactly is following code fragment doing ?

```
f = open("poemBTH.txt", "r")
nl = 0
for line in f:
  nl += 1
print(nl)
```

9. If you use the code of Q.8 with p1.txt created in solved problem 14, what would be its output ?
10. Write a method in python to read the content from a text file diary.txt line by line and display the same on screen. [CBSE D 2015]
11. Write a method in python to write multiple line of text contents into a text file mylife.txt.line. [CBSE D 2016]

Type C : Programming Practice/Knowledge based Questions

1. 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.
2. A file sports.dat contains information in following format : Event ~ Participant
   Write a function that would read contents from file *sports.dat* and creates a file named *Atheletic.dat* copying only those records from *sports.dat* where the event name is *"Atheletics"*.
3. A file contains a list of telephone numbers in the following form :

```
Arvind 7258031
Sachin 7259197
```

   The names contain only one word the names and telephone numbers are separated by white spaces. Write program to read a file and display its contents in two columns.
4. Write a program to count the words "to" and "the" present in a text file "Poem.txt".
5. Write a program to count the number of upper- case alphabets present in a text file "Article.txt".
6. Write a program that copies one file to another. Have the program read the file names from user ?
7. Write a program that appends the contents of one file to another. Have the program take the filenames from the user.
8. 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.
9. 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 "LINES.TXT" contains the following lines :

```
A boy is playing there.
There is a playground.
An aeroplane is in the sky.
Alphabets & numbers are allowed in password.
```

   the function should display the output as 3.