5.7 WORKING WITH CSV FILES

You know that CSV files are delimited files that store tabular data (data stored in rows and columns as we see in spreadsheets or databases) where comma delimits every value, *i.e.*, the values are separated with comma. **Typically the default delimiter of CSV files is the comma** (,) but modern implementations of CSV files allow you to choose a different delimiter character other than comma. Each line in a CSV file is a **data record**. Each record consists of one or more fields, separated by commas (or the chosen delimiter).

NOTE

The separator character of CSV files is called a delimiter. Default and most popular delimiter is comma. Other popular delimiters include the tab (\t), colon (:), pipe (|) and semi-colon (;) characters.

Since CSV files are the *text files*, you can apply text file procedures on these and then split values using **split( )** function BUT there is a better way of handling CSV files, which is – using ***csv* module of Python**. In this section, we are going to talk about how you can read and write in CSV files using the *csv module* methods.

Python csv Module

The **csv** module of Python provides functionality to read and write tabular data in CSV format. It provides *two* specific types of objects – the **reader** and **writer** objects – to read and write into CSV files. The csv module's **reader** and **writer** objects read and write delimited sequences as records in a CSV file.

You will practically learn to use the reader and writer objects and the functions to read and write in CSV files in the coming sub-sections. But before you use **csv** module, make sure to import it in your program by giving a statement as :

```
import csv
```

5.7.1 Opening/Closing CSV Files

A CSV file is opened in the same way as you open any other text file (as explained in section 5.3 earlier), but make sure to do the following *two* things :

(*i*) Specify the file extension as **.csv**

(*ii*) Open the file like other text files, *e.g.*,

```
Dfile = open("stu.csv", "w")
```

$\longleftarrow$ *CSV file opened in **write mode** with file handle as **Dfile***

Or

```
File1 = open("stu.csv", "r")
```

$\longleftarrow$ *CSV file opened in **read mode** with file handle as **File1***

An open CSV file is closed in the same manner as you close any other file, *i.e.*, as :

```
Dfile.close()
```

NOTE

The csv files are popular because of these reasons : (*i*) Easier to create, (*ii*) preferred export and import format for databases and spread-sheets, (iii) capable of storing large amounts of data.

Like text files, a CSV file will get created when opened in an output file mode and if it does not exist already. That is, for the file modes, "*w*", "*w* +", "*a*" "*a* +", the file will get created if it does not exist already and if it exists already, then the file modes "*w*" and "*w* +"will overwrite the existing file and the file mode "*a*" or "*a* +" will retain the contents of the file.

5.7.1A Role of Argument newline in Opening of csv Files

While opening csv files, in the **open( )**, you can specify an additional argument **newline**, which is an optional but important argument. The role of **newline** argument is to specify how would Python handle newline characters while working with csv files.

As csv files are text files, while storing them, some types of translations occur – such as translation of **end of line (EOL) character** as per the operating system you are working on etc. Different operating systems store EOL characters differently. The MS-DOS (including Windows and OS/2), UNIX, and Macintosh operating systems all use different characters to designate the end of a line within a text file. Following table 5.4 lists the EOL characters used by different operating systems.

Table 5.4 *EOL characters used in different operating systems.*

Symbol/Char	Meaning	Operating System
CR [\r]	Carriage Return	Macintosh
LF [\n]	Line Feed	UNIX
CR/LF [\r \n]	Carriage Return/Line Feed	MS-DOS, Windows, OS/2
NULL [\0]	Null character	Other OSs

Now what would happen if you create a csv file on one operating system and use it on another. The EOL of one operating system will not be treated as EOL on another operating system. Also, if you have given a character, say '\r', in your text string (not as EOL but as part of a text string) and if you try to open and read from this file on a Macintosh system, what would happen ? Mac OS will treat every '\r' as EOL – yes, including the one you gave as part of the text string.

So what is the solution? Well, the solution is pretty simple. Just suppress the EOL translation by specify third argument of **open( )** as newline = ' ' (null string – no space in quotes).

If you specify the **newline** argument while writing onto a csv file, it will create a csv file with no EOL translation and you will be able to use csv file in normal way on any platform.

NOTE

Additional optional argument as newline = '' (null string; no space in between) with file **open( )** will ensure that ***no translation of end of line (EOL) character*** takes place.

That is, open your csv file as :

```
Dfile = open("stu.csv", "w", newline = '')
```

null string ; no space in between

*CSV file opened in **write mode** with file handle as **Dfile** (no EOL translation)*

Or

```
File1 = open("stu.csv", "r", newline = '')
```

*CSV file opened in **read mode** with file handle as **File1** (no EOL translation)*

Not only this is useful for working across different platforms, it is also useful when you work on the same platform for writing and reading. You will be able to appreciate the role of the *newline* argument when we talk about reading from the csv files. Program 5.21 onwards it will be clear to you.

Let us now learn to work with **csv module's** methods to write / read into CSV files.

5.7.2 Writing in CSV Files

Writing into csv files involves the conversion of the user data into the writable delimited form and then storing it in the form of csv file. For writing onto a csv files, you normally use the following *three* key players functions[2].

2. Other than the above shown functions, there are other functions too like ***DictWriter( )*** function but these are beyond the scope of the syllabus, hence we shall not cover these in this chapter.

These are :

`csv.writer()`	returns a writer object which writes data into CSV file
`<writerobject>.writerow()`	writes one row of data onto the *writer object*
`<writerobject>.writerows()`	writes multiple rows of data onto the *writer object*

Before we proceed, it is important for you to understand the significance of the **writer object.** Since csv files are delimited flat files and before writing onto them, the data must be in **csv-writable-delimited-form**[3] , it is important to convert the received user data into the form appropriate for the csv files. **This task is performed by the writer object**. The data row written to a writer object (written using **writerow( )** or **writerows( )** functions) gets converted to csv writable delimited form and then written on to the linked csv file on the disk.

Following figure 5.8 illustrates this process.

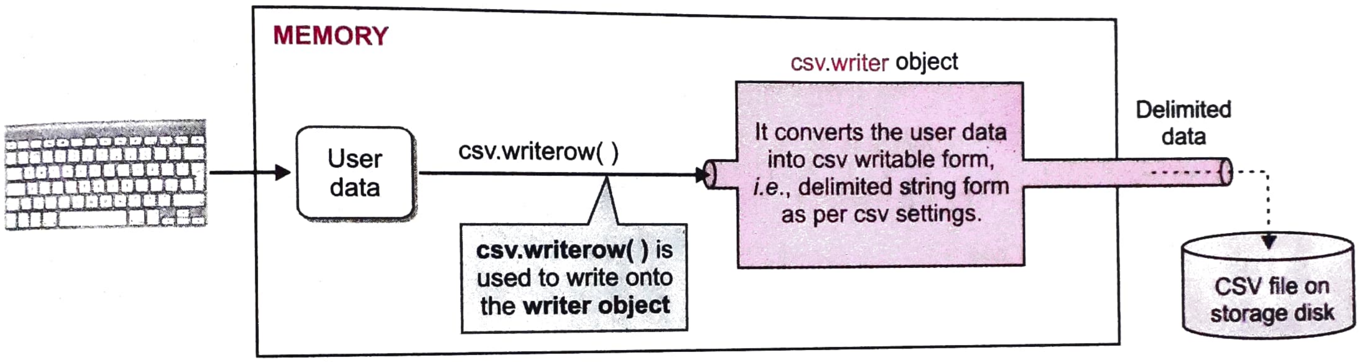

MEMORY
User data
csv.writerow( )
csv.writer object
It converts the user data into csv writable form, i.e., delimited string form as per csv settings.
csv.writerow( ) is used to write onto the writer object
Delimited data
CSV file on storage disk

Figure 5.8 Role of the csv writer object.

Let us now learn to write onto csv files. In order to write onto a csv file, you need to do the following (**Note**, in order to appreciate the use of *newline* argument, we shall initially create csv files without the *newline* argument first and later we shall create csv files with *newline* argument too.)

(*i*) Import csv module

(*ii*) Open csv file in a file-handle (just as you open other text files), *e.g.*,

```
fh = open("student.csv", "w")
```

(*iii*) Create the **writer object** by using the syntax as shown below:

```
<name-of-writer-object> = csv.writer(<file-handle>, [delimiter = <delimiter character>])
```

If you skip this argument, comma will be used as the delimiter

e.g.,

```
stuwriter = csv.writer(fh)
```

you opened the file with this file handle in previous step

In the above statement, delimiter argument is not specified. When you skip the delimiter argument, the default delimiter character, which is comma, is used for separating characters. But if you specify a character with **delimiter** argument then the specified character is used as the delimiter, *e.g.*,

```
stuwriter = csv.writer(fh, delimiter = '|')
```

The above statement will create file with delimiter character as pipe symbol ('|').

3. It depends on something called ***dialect***, which by default is set for producing ***excel*** format like csv files. It can even be ***excel-tab*** where tab character is the delimiter. You can set the dialect using ***dialect*** argument but we shall only work with the default dialect and hence won't use *dialect* argument here. For further details on *dialect* argument, you may refer to the Python's documentation.

(*iv*) Obtain user data and form a Python sequence (list or tuple) out of it, *e.g.*,

```
Sturec = (11, 'Neelam', 79.0)
```

(*v*) Write the Python sequence containing user data onto the writer object using csv.writerow( ) or csv.writerows( ) functions, *e.g.*,

```
csv.writerow(Sturec)
```

Both the **writerow( )** and **writerrows( )** functions can be used for writing onto the writer object. We shall discuss about how to use **writerows( )** function little later. For now, let us focus on writing through **writerow( )** function, *i.e.*, writing single row at a time.

CSV files can also take the names of columns as header rows, which are written in the same way as any row of data is written, *e.g.*, to give column headings as "Rollno", "Name" and "Marks", you may write :

```
stuwriter.writerow(['Rollno', 'Name', 'Marks'])
```

(*vi*) Once done, close the file.

You only need to write into the writer object. Rest of the work it will do itself, *i.e.*, converting the data into delimited form and then writing it onto the linked csv file.

Let us now do it practically. Following program illustrates this process.

Program 5.19 Write a program to create a CSV file to store student data (Rollno., Name, Marks). Obtain data from user and write 5 records into the file.

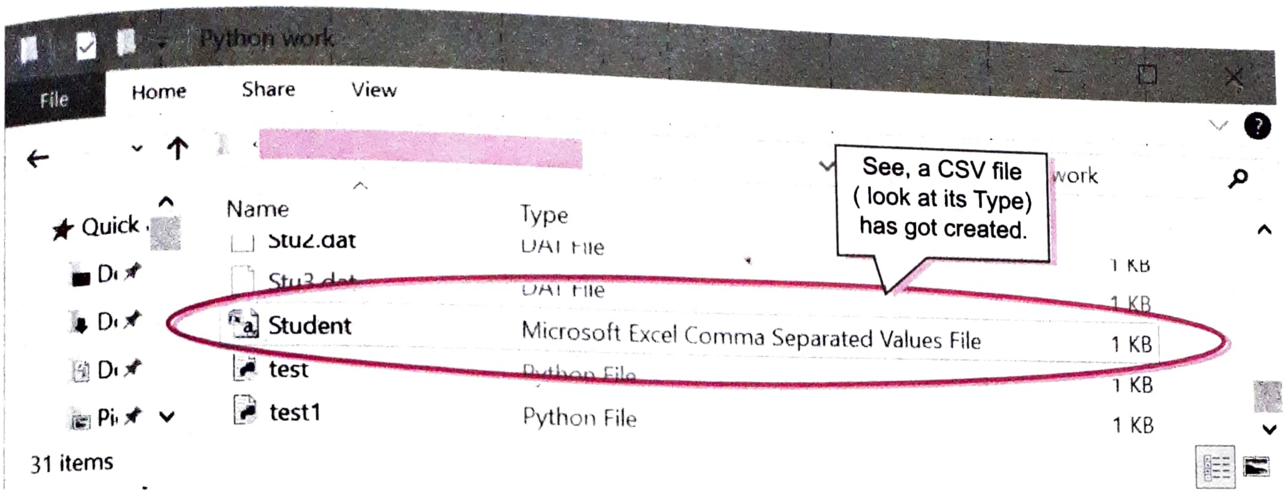
```
import csv
fh = open("Student.csv", "w")            #open file
stuwriter = csv.writer(fh)
stuwriter.writerow(['Rollno', 'Name', 'Marks'])        #write header row
                                          Column headings written
for i in range(5):
    print("Student record", (i+1))
    rollno = int(input("Enter rollno:"))
    name = input("Enter name:")
    marks = float(input("Enter marks:"))
    sturec = [rollno, name, marks]        #create sequence of user data
    stuwriter.writerow(sturec)            Python sequence created from user data

fh.close()          #close file            Python data sequence written on the writer
                                          object using writerow( )
```

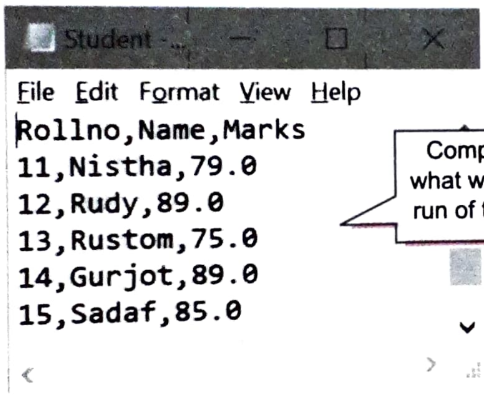
The sample run of the above program is as shown below :

```
Student record 1          Student record 3          Student record 5
Enter rollno:11           Enter rollno:13           Enter rollno:15
Enter name:Nistha         Enter name:Rustom         Enter name:Sadaf
Enter marks:79            Enter marks:75            Enter marks:85
Student record 2          Student record 4
Enter rollno:12           Enter rollno:14
Enter name:Rudy           Enter name:Gurjot
Enter marks:89            Enter marks:89
```

Now if you open the folder of your Python program, you will see that Python has created a file by the name **Student.csv**



And if you open this file in Notepad or any other ASCII editor, it shows :


Student
File Edit Format View Help
Compare the data with what was typed in sample run of the above program

```
Rollno,Name,Marks
11,Nistha,79.0
12,Rudy,89.0
13,Rustom,75.0
14,Gurjot,89.0
15,Sadaf,85.0
```

The writerows( ) Function

If you have all the data available and data is not much lengthy then it is possible to write all data in one go. All you need to do is create a nested sequence out of the data and then write using the **writerows( )** function. The **writerows( )** method writes all given rows to the CSV file, *e.g.*, to write following nested sequence, you can use the **writerows( )** function :

```
Sturec = [ [11,Nistha,79.0], [12,Rudy,89.0], [13,Rustom,75.0] ]

<writerobject>.writerows(Sturec)
```

Each inner list will now be written as a separate record in the csv file.

Following program illustrates this.

Program 5.20 The data of winners of four rounds of a competitive programming competition is given as :

```
['Name', 'Points', 'Rank']
['Shradha', 4500, 23]
['Nishchay', 4800, 31]
['Ali', 4500, 25]
['Adi', 5100, 14]
```

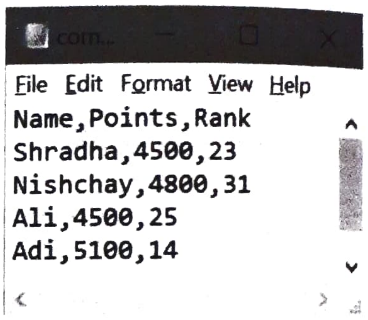
Write a program to create a csv file (compresult.csv) and write the above data into it.

```
import csv
fh = open("compresult.csv", "w")
cwriter = csv.writer(fh)
compdata = [
     ['Name', 'Points', 'Rank'],
     ['Shradha', 4500, 23],
     ['Nishchay', 4800, 31],
     ['Ali', 4500, 25],
     ['Adi', 5100, 14] ]
cwriter.writerows(compdata)
fh.close()
```

This nested sequence contains multiple records' data in the form of inner lists

Nested list written in one go using ***writerows( )***

The above program will create a csv file and in Notepad, it will look like :



```
Name,Points,Rank
Shradha,4500,23
Nishchay,4800,31
Ali,4500,25
Adi,5100,14
```

You can also create the nested sequence programmatically by appending one record to a list while writing using **writerows( )** function. ***Solved problem 48 performs the same.***

Please note that till now we have created csv files without using the **newline** argument, which means that EOL translation has taken place internally. How this impacts a csv file, will become clear to you soon when we start programming in the following section.

5.7.3 Reading in CSV Files

Reading from a csv file involves loading of a csv file's data, **parsing** it (*i.e.*, removing its delimitation), loading it in a ***Python iterable*** and then reading from this iterable. Recall that any Python sequence that can be iterated over in a for-loop is a *Python iterable, e.g., lists, tuples,* and *strings* are all Python iterables.

For reading from a csv files, you normally use following *function* :

`csv.reader()`	returns a reader object which loads data from CSV file into an iterable after parsing delimited data

The **csv.reader** object does the opposite of **csv.writer** object. The *csv.reader* object loads data from the csv file, parses it, *i.e.*, removes the delimiters and returns the data in the form of a Python iterable wherefrom you can fetch one row of data at a time. (Fig. 5.9)

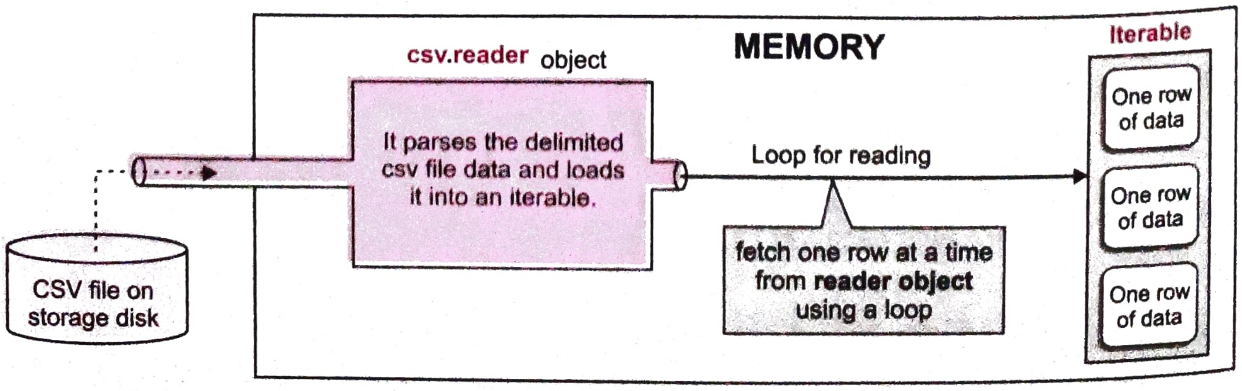

MEMORY
csv.reader object
It parses the delimited csv file data and loads it into an iterable.
CSV file on storage disk
Loop for reading
fetch one row at a time from reader object using a loop
Iterable
One row of data
One row of data
One row of data

Figure 5.9 Role of csv.reader object.

Let us now learn to read from csv files. In order to read from a csv file, you need to do the following :

(*i*) Import csv module

(*ii*) Open csv file in a file-handle in read mode (just as you open other text files),

```
<file handle> = open (<csv-file>, <read mode>)
```

e.g.,

```
fh = open("student.csv", "r")
```

The file being opened must already exist otherwise an exception will get raised. Your code should be able to handle the exception *i.e.,* through **try..except**. (Alternatively you can use the **with** statement as mentioned below.)

(*iii*) Create the **reader object** by using the syntax as shown below :

```
<name-of-reader-object> = csv.reader(<file-handle>,[delimiter =<delimiter character>])
```

e.g.,

```
stureader = csv.reader(fh)
```

you opened the file with this file handle in previous step

You may also specify a delimiter character other than a comma using the **delimiter** argument, *e.g.,*

```
stureader = csv.reader(fh, delimiter = '|')
```

(*iv*) The reader object stores the parsed data in the form of iterable and thus you can fetch from it row by row through a traditional for loop, one row at a time :

*Loop to fetch one row at a time in **rec** from the iterable*

```
for rec in stureader :
    print (rec)              # or do any other processing
```

(*v*) Process the fetched single row of data as required.

(*vi*) Once done, close the file.

All the above mentioned steps are best performed through **with** statement as the **with** statement will also take care of any exception that may arise while opening/reading a file. Thus you should process the csv file as per the following syntax, which combines all the above mentioned steps :

```
with open (<csv-file>, <read mode>) as <file handle> :
    <name-of-reader-object> = csv.reader(<file-handle>)
    for <row identifier> in <reader object> :
        : # process the fetched row in <row identifier>
```

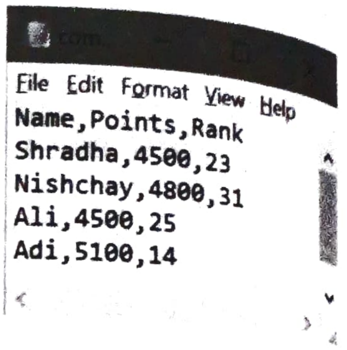
Let us do it practically.

Following program is doing the same.

NOTE

Csv files are flat, text files.

Program 5.21 You created the file compresult.csv in the previous program as shown below. Write a program to read the records of this csv file and display them.



```
Name,Points,Rank
Shradha,4500,23
Nishchay,4800,31
Ali,4500,25
Adi,5100,14
```

Note. *Before we write the code for this program, recall that the given file created in the previous program,* ***was created without specifying the newline argument*** *in the file* ***open( )*** *function. So have a look at the previous program's (program 5.20) code before starting with this program's code.*

```
import csv
with open("compresult.csv", "r") as fh :
    creader = csv.reader(fh)
    for rec in creader :
        print(rec)
```

Loop to fetch one row at a time in ***rec*** *from the iterable in* ***creader***

The above program will produce the result as :

```
['Name', 'Points', 'Rank']
[]
['Shradha', '4500', '23']
[]
['Nishchay', '4800', '31']
[]
['Ali', '4500', '25']
[]
['Adi', '5100', '14']
[]
```

Where have these empty rows come from ?

We never entered empty rows. Refer to the sample run of program 5.20.

Then what is the source / reason of these empty rows ?

Compare the above output with the sample run of previous program (program 5.20). We never entered empty rows. Then where have these empty rows come from? Any idea?

You guessed it right. We did not specify the **newline** argument in the file **open( )** function while creating/writing this csv file (*compresult.csv*, in this case) and thus EOL translation took place, which resulted in the blank rows after every record while reading. In fact, the above shown file data was internally stored as :

```
['Name', 'Points', 'Rank'] \r
\n
['Shradha', '4500', '23']\r
\n
['Nishchay', '4800', '31'] \r
\n
:
```

Every data record line was appended with EOL character '\r\n' on Windows OS because of EOL translation

So what is the solution ? You are right again ☺. We should have created the file with **newline** argument set to null string (' ') so that no EOL translation could take place.

In fact, the above situation can be handled in *two* ways :

(*i*) **Method1.** Create/write onto csv file with **newline** = '' argument in open( ) function so this situation would not arise. ***Programs*** 5.23(*a*) ***and*** (*b*) illustrate this solution.

(*ii*) **Method2.** To read from a file which is created with EOL translation (*i.e.*, no newline argument), open it with **newline** argument set to the EOL character of the OS where the csv file was created. For instance, if you created the csv file on a Windows OS, open file for reading with **newline** argument set as '\r\n' because the EOL on Windows is stored as '\r\n', *i.e.*, as :

Now the file will be read considering every '\r\n' character as newline.

```
open(<csvfilename>, <readmode>, newline = '\r\n')
```

Recall that the table 5.4 lists EOL characters on various operating systems.

Program 5.22 handles this situation using method 2 listed here. Let us have a look at it.

Program **5.22** The csv file (compresult.csv) used in the previous program was created on Windows OS where the EOL character is '\r\n'. Modify the code of the previous program so that blank lines for every EOL are not displayed.

```
import csv
with open("compresult.csv", "r", newline = '\r\n') as fh :
    creader = csv.reader(fh)
    for rec in creader :
        print(rec)
```

Now the file will be read considering every '\r\n' character as ***end of line*** *and not as a separate line*

The output produced by above program is :

```
['Name', 'Points', 'Rank']
['Shradha', '4500', '23']
['Nishchay', '4800', '31']
['Ali', '4500', '25']
['Adi', '5100', '14']
```

See, no blank lines this time

The newline argument can be specified in independent open( ) statement as well as in the open( ) of with statement.

Let us now treat the earlier listed problem of blank lines in between records using the method 1 listed above. For this, we shall create the file with newline argument and then we shall not need any newline argument while reading as no EOL translation would have taken place.

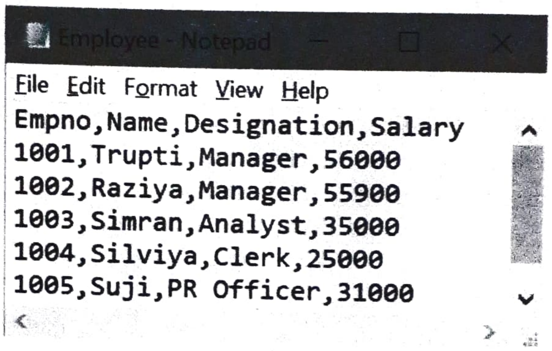
Programs 5.23(*a*) and (*b*) are illustrating the same.

Program **5.23(a)** Write a program to create a csv file by suppressing the EOL translation.

This argument ensures that no EOL translation takes place

```
import csv
fh = open("Employee.csv", "w", newline = '')
ewriter = csv.writer(fh)
empdata = [
    ['Empno','Name','Designation','Salary'],
    [1001,'Trupti','Manager',56000],
        [1002,'Raziya','Manager',55900],
        [1003,'Simran','Analyst',35000],
        [1004,'Silviya','Clerk',25000],
        [1005,'Suji','PR Officer',31000 ]
        ]
    ewriter.writerows(empdata)
    print("File successfully created")
    fh.close()
```

The above program created a file as shown below :


Employee - Notepad

File Edit Format View Help

```
Empno,Name,Designation,Salary
1001,Trupti,Manager,56000
1002,Raziya,Manager,55900
1003,Simran,Analyst,35000
1004,Silviya,Clerk,25000
1005,Suji,PR Officer,31000
```

Let us now read from the above created file. This time we need not specify the **newline** argument as no EOL translation has taken place. Program 5.23(*b*) is doing the same.

Program **5.23(b)** Write a program to read and display the contents of Employee.csv created in the previous program.

```
import csv

with open("Employee.csv", "r") as fh :
    ereader = csv.reader(fh)
    print("File Employee.csv contains :")
    for rec in ereader :
        print(rec)
```

See, we did not specify newline argument in the open( ) as no EOL translation took place when the file was created.

Check Point **5.2**

1. What are text files ?
2. What are binary files ?
3. What are CSV files ?
4. Name the functions used to read and write in plain text files.
5. Name the functions used to read and write in binary files.
6. Name the functions used to read and write in CSV files.
7. What is the full form of :
   (*i*) CSV (*ii*) TSV

[CBSE Sample Paper 2020-21]

The output produced by the above program is :

```
File Employee.csv contains :
['Empno', 'Name', 'Designation', 'Salary']
['1001', 'Trupti', 'Manager', '56000']
['1002', 'Raziya', 'Manager', '55900']
['1003', 'Simran', 'Analyst', '35000']
['1004', 'Silviya', 'Clerk', '25000']
['1005', 'Suji', 'PR Officer', '31000']
```

See, no blank lines in between the records this time because the csv file was created with no EOL translation.

With this we have come to the end of this chapter.

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 Python after 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 three ways : (i) Text files (ii) Binary files (iii) CSV 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 manipulations 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 and a csv 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 text files 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 text files are write( ) and writelines( ).*
- *While write( ) writes a string in file, writelines( ) writes a list in a file.*
- *Pickle module's dump( ) and load( ) functions write and read into binary files.*
- *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(..).*
- *Every open file maintains a file-pointer to determine and control the position of read or write operation in file.*
- *The seek( ) function places the file pointer at specified position.*
- *The tell( ) function returns the current position of file-pointer in open file.*
- *CSV files are delimited files that store tabular data (data stored in rows and columns as we see in spreadsheets or databases) where comma delimits every value.*
- *For writing onto a csv file, user data is written on a csv.writer object which converts the user data into delimited form and writes it on to the csv file.*
- *For reading from a csv file, **csv.reader** loads data from the csv file, parses it and makes it available in the form of an iterator wherefrom the records can be fetched row by row.*
- *CSV files should be opened with newline argument to suppress EOL translation.*