

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 (,) 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).

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.

NOTE

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

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")
```

Or

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

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

```
Dfile.close()
```

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.

NOTE

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

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 specifying 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.

That is, open your csv file as :

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

Or

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

null string ; no space in between

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

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* functions².

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 :

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

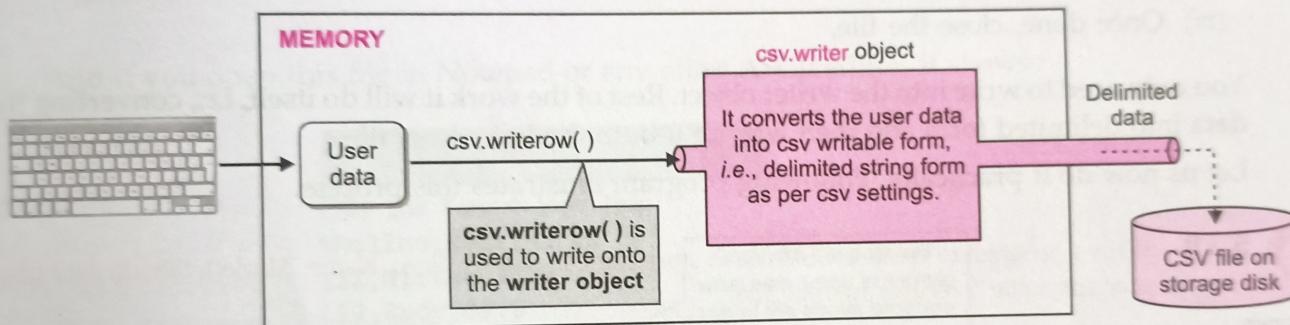


Figure 5.8 Role of the csv writer object.

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 `writerows()` 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.

**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.

**P**rogram

```
import csv
fh = open("Student.csv", "w") #open file
stuwriter = csv.writer(fh)
stuwriter.writerow(['Rollno', 'Name', 'Marks']) #write header row
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]
 stuwriter.writerow(sturec) #create sequence of user data
fh.close() #close file
```

Column headings written

Python sequence created from user data

Python data sequence written on the writer object using `writerow()`

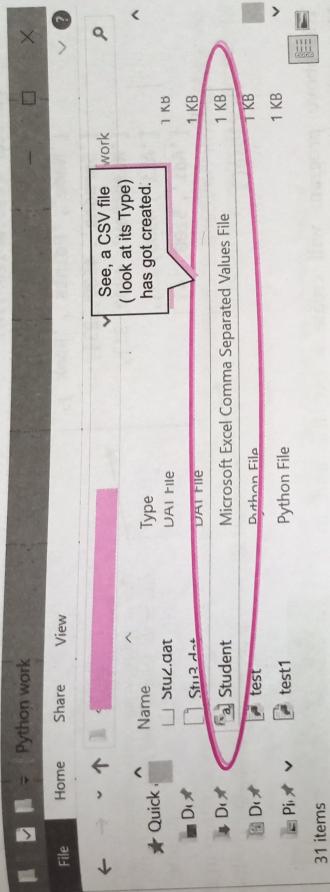
The sample run of the above program is as shown below :

Student record 1  
Enter rollno:11  
Enter name:Nistha  
Enter marks:79  
Student record 2  
Enter rollno:12  
Enter name:Rudy  
Enter marks:89

Student record 3  
Enter rollno:13  
Enter name:Rustom  
Enter marks:75  
Student record 4  
Enter rollno:14  
Enter name:Gurjot  
Enter marks:89

Student record 5  
Enter rollno:15  
Enter name:Sadaf  
Enter marks:85

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                                                                  |
| Kollno,Name,Marks |      |                                                                         |
| 11,Nistha,79.0    |      | Compare the data with what was typed in sample run of the above program |
| 12,Rudy,89.0      |      |                                                                         |
| 13,Rustum,75.0    |      |                                                                         |
| 14,Surjot,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,Rustum,75.0]]
```

```
<writerobject>.writerows(Sturec)
```

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

Following program illustrates this.

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

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

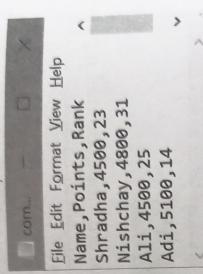
Write a program to create a csv file (comprelist.csv) and write the above data into it.

```

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

```

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



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

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.

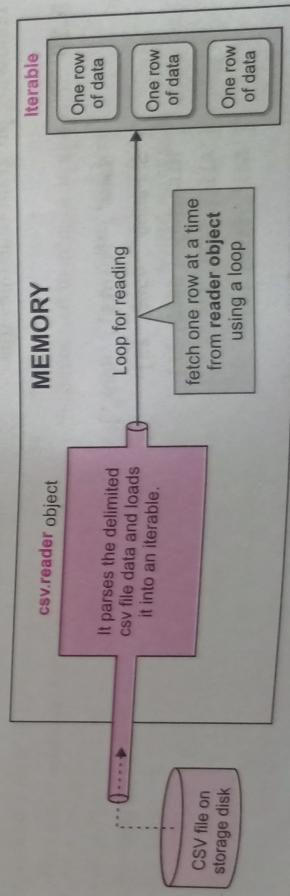
### 5.7.3 Reading in CSV Files

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 :

|                           |                                                                                                             |
|---------------------------|-------------------------------------------------------------------------------------------------------------|
| <code>csv.reader()</code> | returns a <b>reader object</b> 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 whereinfrom you can fetch one row of data at a time. (Fig. 5.9)



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 :

`for rec in stureader :` *Loop to fetch one row at a time in rec from the iterable*  
`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.



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

**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("comprest.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 ?

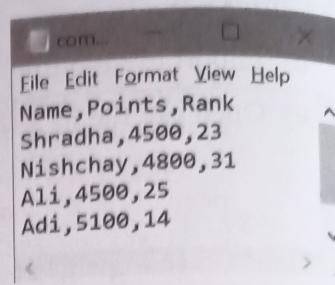
Compare the above output with the sample run of previous program (program 5.20). We never entered empty rows. Then where have these empty rows come from? Any idea?

You guessed it right. We did not specify the **newline** argument in the file `open()` function while creating/writing this csv file (`comprest.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`', *i.e.*, as :

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

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

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.

### P 5.22 Write a program to read a csv file.

The csv file (`compreresult.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("compreresult.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 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.

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

```
Program
import csv
fh = open("Employee.csv", "w", newline = '')
ewriter = csv.writer(fh)
empdata = [
 ['Empno', 'Name', 'Designation', 'Salary'],
 ['1001', 'Trupti', 'Manager', 56000],
 ['1002', 'Rajesh', 'Analyst', 45000]
]
```

*This argument ensures that no EOL translation takes place*

```
[1002, 'Raziya', 'Manager', 55900],
[1003, 'Simran', 'Analyst', 35000],
[1004, 'Silvija', 'Clerk', 25000],
[1005, 'Saji', 'PR Officer', 31000]

]
```

```
ewriter.writerow(empdata)
print("file successfully created")
fh.close()
```

The above program created a file as shown below :

A screenshot of a Windows Notepad window titled "Employee - Notepad". The window displays the following text:

```
File Edit Format View Help
Empno,Name,Designation,Salary
1001,Trupti,Manager,56000
1002,Raziya,Manager,55900
1003,Simran,Analyst,35000
1004,Silvija,Clerk,25000
1005,Saji,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.

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

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

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

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', 'Silvija', 'Clerk', '25000']
['1005', 'Saji', 'PR Officer', '31000']
```

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

7. What is the full form of :

(i) CSV

(ii) TSV