HANDLING CSV FILES IN PYTHON

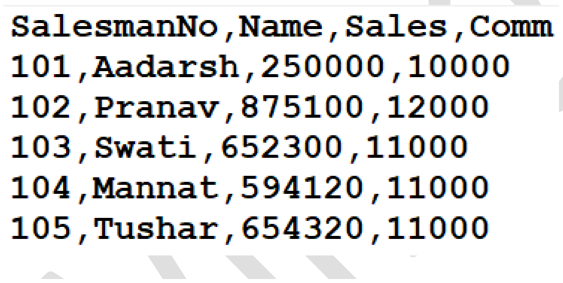
LEARNING OBJECTIVES

- Introduction to CSV Files
- Advantages of CSV files
- Difference between CSV and Excel files
- Python CSV Module
- Opening CSV files
- Closing CSV files
- Writing into a CSV file
- Reading from a CSV file

INTRODUCTION TO CSV FILES

CSV (Comma Separated Values) is a type of plain text file used to store tabular data. They are a convenient way to export data from spreadsheets and databases as well as import or use it in other programs. These files are saved with the **.csv** file extension.

A CSV file stores tabular data (numbers and text) in plain text. Here's what the structure of CSV file looks like:



```
SalesmanNo,Name,Sales,Comm
101,Aadarsh,250000,10000
102,Pranav,875100,12000
103,Swati,652300,11000
104,Mannat,594120,11000
105,Tushar,654320,11000
```

Here, each piece of data is separated by a comma. Normally, the first line identifies each piece of data—in other words, the name of a data column. Every subsequent line after that is actual data and forms a data record. Each record consists of one or more fields, separated by commas. In general, the separator character is called a delimiter, and other popular delimiters include the tab (\t), colon (:) and semi-colon (;) characters. Usually reading a CSV file requires us to know which delimiter is being used.

ADVANTAGES OF CSV FILES

- CSV is human readable and easy to edit manually

- CSV is simple to implement and parse
- CSV is processed by almost all existing applications
- CSV is easy to handle and smaller in size
- CSV is considered to be standard format
- CSV is compact and easy to generate

DIFFERENCE BETWEEN CSV AND EXCEL FILES

Excel and CSV both help store data in tabular format. Besides this commonality, there are many important differences in their respective features and usages.

- CSV is a format for saving tabular information into a delimited text file with extension .csv whereas Excel is a spreadsheet that keeps files into its own proprietary format viz xls or xlsx.
- CSV is a plain text format with a series of values separated by commas whereas Excel is a binary file that holds information about all the worksheets in a workbook.
- CSV file can’t perform operations on data while Excel can perform operations on the data.
- CSV files are faster and also consumes less memory whereas Excel consumes more memory while importing data.
- CSV files can be opened with any text editor in windows while Excel files can't be opened with text editors.

PYTHON CSV MODULE

The CSV library in Python provides functionality to both read from and write to CSV files. It contains objects and other code to read, write, and process data from and to CSV files. Before using the CSV module, you need to import it in your program by giving a statement:

import csv

The two important functions of CSV module are:

csv.reader()	returns a reader object which iterates over lines of a CSV file
csv.writer()	returns a writer object which writes data into CSV file

OPENING CSV FILE

Python has a built in function – open() to open a file. The open() function takes two arguments and returns a file object.

SYNTAX

file_object = open(filename [, mode] [,newline=None])

where

filename is the name of the file to be opened. It can be string constant or a variable. If the file is not present in the same folder / directory in which we are working we need to give the complete path where the file is present.

mode is an optional parameter and it describes the type of operation we are going to perform on file like read, write, append etc. Please note that the default mode is reading.

newline is an optional parameter and it tells how newlines mode works (available values: None, ' ', '\n', 'r', and '\r\n'.

file_object is the name of the object returned by the open function.

File objects contain methods and attributes that can be used to collect information about the file you have opened and to manipulate it.

Python File Modes	
Mode	Description
'r'	Open a file for reading. (default)
'w'	Open a file for writing. Creates a new file if it does not exist or truncates the file if it exists.
'x'	Open a file for exclusive creation. If the file already exists, the operation fails.
'a'	Open for appending at the end of the file without truncating it. Creates a new file if it does not exist.
't'	Open in text mode. (default)
'b'	Open in binary mode.
'+'	Open a file for updating (reading and writing)

EXAMPLE 1

Python Code	Explanation
file= open("test.csv","w", newline='')	will open a csv file called test.csv for writing purpose.
f = open("test.csv", newline='')	will open a csv file called test.csv for reading purpose.
file= open("example.csv", "a+", newline='')	will open a csv file called example.csv for both appending (writing at the end) and reading purpose.

Note: It's very important to add newline = '' to ensure that no translation of end of line (EOL) character takes place. if newline='' is not specified, newlines embedded inside quoted fields will not be interpreted correctly, and on platforms that use \r\n line on write an extra \r will be added. It should always be safe to specify newline='', since the csv module does its own (universal) newline handling.

Specifying newline argument while writing into a CSV file will create a CSV file with no EOL translation and we can use CSV file normally on any platform.

- **USING WITH STATEMENT**

Apart from using open() function for creation of file, **with statement** can also be used. Using **with** ensures that all the resources allocated to file objects gets deallocated automatically once we stop using the file.

SYNTAX

with open() as fileobject :

EXAMPLE 2:

```
with open("example.csv","r+", newline='') as file:
        file manipulation statements
```

CLOSING CSV FILE

After performing desired operations on file, we need to close it. This can be done using an in-built function- close().

SYNTAX

fileobject. close()

Closing a file frees up the resources that were tied with the file. Before closing a file, any material which is not written in file, is flushed off i.e. written to file. So, it is a good practice to close the file once we have finished using it. If we reassign the file object to some other file, Python will automatically close the file attached to it earlier.

WRITING INTO A CSV FILE

To write into a CSV file in Python, we can use the csv.writer() function.

The csv.writer() function returns a writer object that converts the user's data into a delimited string. This string can later be used to write into CSV files using the writerow() or the writerows() function.

SYNTAX

writer_object=csv.writer(file-handle [,delimiter])

Here, delimiter parameter is optional. If it not specified then the default delimiter character i.e. comma is taken. If it is specified then the specified character is taken as the delimiter character.

USING WRITEROW() FUNCTION

The **writerow()** function is used to write a single row to the specified CSV file.

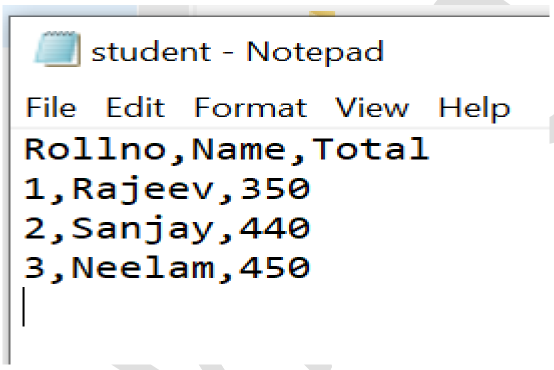
EXAMPLE 3

Write a Python script to create a file students.csv having fields rollno, name and total and write three records into it.

```
import csv
with open('student.csv', 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(["Rollno", "Name", "Total"])
    writer.writerow([1, "Rajeev", 350])
    writer.writerow([2, "Sanjay", 440])
    writer.writerow([3, "Neelam", 450])
```

OUTPUT

No output is generated after running the above code. When we run the above program, a student.csv file is created with the following content:



```
Rollno,Name,Total
1,Rajeev,350
2,Sanjay,440
3,Neelam,450
```

WORKING

In the above program, we have opened the file student.csv in writing mode. The writer function returns a writer object that converts the user's data into a delimited string so that it can be written to the csv file. Then, we have passed each row as a list. These lists are converted to a delimited string and written into the CSV file using writer.writerow() function.

Note: The csv file is created in your Python folder.

USING WRITEROWS() FUNCTION

The **writerows()** function is used to write multiple rows to the specified CSV file. For e.g. it can be used to write the contents of the 2-dimensional list to a CSV file.

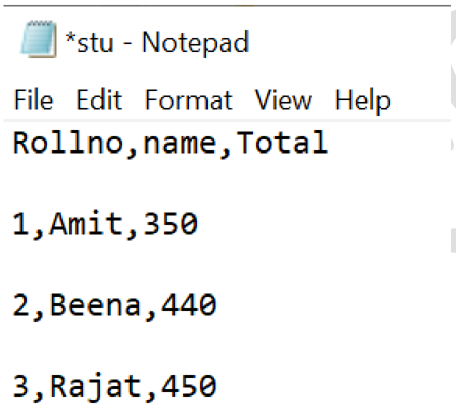
EXAMPLE 4

Write a Python script to create a file students.csv having fields rollno, name and total and write three records into it using writerrows() function.

```
import csv
csv_rowlist = [["Rollno", "name", "Total"], [1, "Amit", 350], [2, "Beena", 440],
            [3, "Rajat", 450]]
with open('stu.csv', 'w') as file:
   writer = csv.writer(file)
   writer.writerows(csv_rowlist)
```

OUTPUT

No output is generated after running the above code. When we run the above program, a stu.csv file is created with the following content:



```
Rollno,name,Total

1,Amit,350

2,Beena,440

3,Rajat,450
```

WORKING

In the above program, we have opened the file student.csv in writing mode. A 2-dimensional list is passed to the writer.writerows() method to write the content of the list to the CSV file. As newline parameter was not added at the time of opening of file thus a blank line has been inserted after every row. The blank line can be removed by adding **newline=''** to the open function.

Writing to a CSV File with CUSTOM Delimiter

CSV files can be written in different formats. By default, a comma is used as a delimiter in a CSV file. However, other delimiter characters like | and \t can also be used in CSV files. Here, we will then learn how to customize the csv.writer() function to write them.

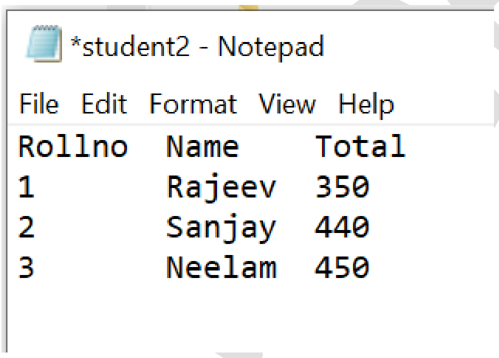
EXAMPLE 5

Write a Python script to create a file students.csv having fields rollno, name and total and use TAB as a delimiter.

```
import csv
with open('student2.csv', 'w', newline='') as file:
    writer = csv.writer(file, delimiter = '\t')
    writer.writerow(["Rollno", "Name", "Total"])
    writer.writerow([1, "Rajeev", 350])
    writer.writerow([2, "Sanjay", 440])
    writer.writerow([3, "Neelam", 450])
```

OUTPUT

No output is generated after running the above code. When we run the above program, a student2.csv file is created with the following content:



```
Rollno  Name    Total
1       Rajeev  350
2       Sanjay  440
3       Neelam  450
```

WORKING

The program is similar to the programs we have already written except for the optional parameter delimiter = '\t' in the csv.writer() function. The csv file created has tab as the delimiter character instead of comma.

READING CSV FILES

To read a CSV file in Python, we can use the csv.reader() function. It returns a reader object which iterates over lines of a CSV file.

EXAMPLE

Write a Python script to read a file students.csv having fields rollno, name and total and display all information stored in that file.

```
import csv
with open('student.csv', 'r') as file:
    reader = csv.reader(file)
    for row in reader:
        print(row)
```

OUTPUT

```
['Rollno', 'Name', 'Total']
['1', 'Rajeev', '350']
['2', 'Sanjay', '440']
['3', 'Neelam', '450']
```

WORKING

First, we first open the CSV file in READ mode. The file object is named as file. Next, we create the reader object. The reader object stores the parsed data in the form of iterable. The reader object is then iterated using a for loop to print the contents of each row. Each row returned by the reader is a list of string elements containing the data found by removing the delimiters. The first row returned contains the column names, which is handled in a special way.

READ CSV FILE HAVING CUSTOM DELIMITER

In the above example, we are using the csv.reader() function in default mode for CSV files having comma delimiter. Suppose our CSV file is using tab as a delimiter. To read such files, we can pass optional parameters to the csv.reader() function. Let's take an example.

EXAMPLE

Write a Python script to read a file student2.csv having fields rollno, name and total and having tab as the delimiter. Display all information stored in that file.

```
import csv
with open('student2.csv', 'r',) as file:
    reader = csv.reader(file, delimiter = '\t')
    for row in reader:
        print(row)
```

OUTPUT

```
['Rollno', 'Name', 'Total']
['1', 'Rajeev', '350']
['2', 'Sanjay', '440']
['3', 'Neelam', '450']
```

WORKING

Here, we have added delimiter = '\t' in the reader() function so as to facilitate reading of csv file having delimiter as tab.