Python CSV

A CSV (Comma Separated Values) format is one of the most simple and common ways to store tabular data. To represent a CSV file, it must be saved with the **.csv** file extension.
Let's take an example:

If you open the above CSV file using a text editor such as sublime text, you will see:

```
SN, Name, City
1, Michael, New Jersey
2, Jack, California
```

As you can see, the elements of a CSV file are separated by commas. Here, `,` is a delimiter.
You can have any single character as your delimiter as per your needs.

Note: The csv module can also be used for other file extensions (like: **.txt**) as long as their contents are in proper structure.

Working with CSV files in Python

While we could use the built-in `open()` function to work with CSV files in Python, there is a dedicated csv module that makes working with CSV files much easier.

Before we can use the methods to the `csv` module, we need to import the module first using:

```
import csv
```

Reading CSV files Using csv.reader()

To read a CSV file in Python, we can use the `csv.reader()` function. Suppose we have a `csv` file named **people.csv** in the current directory with the following entries.

Name	Age	Profession

Jack	23	Doctor
Miller	22	Engineer

Let's read this file using `csv.reader()`:

Example 1: Read CSV Having Comma Delimiter

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

Output

```
['Name', 'Age', 'Profession']
['Jack', '23', 'Doctor']
['Miller', '22', 'Engineer']
```

Here, we have opened the **people.csv** file in reading mode using:

```
with open('people.csv', 'r') as file:
    .. .. ...
```

To learn more about opening files in Python, visit: Python File Input/Output

Then, the `csv.reader()` is used to read the file, which returns an iterable `reader` object.

The `reader` object is then iterated using a `for` loop to print the contents of each row.

In the above example, we are using the `csv.reader()` function in default mode for CSV files having comma delimiter.

However, the function is much more customizable.

Suppose our CSV file was 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 2: Read CSV file Having Tab Delimiter

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

Notice the optional parameter `delimiter = '\t'` in the above example.

The complete syntax of the `csv.reader()` function is:

```
csv.reader(csvfile, dialect='excel', **optional_parameters)
```

As you can see from the syntax, we can also pass the dialect parameter to the `csv.reader()` function. The `dialect` parameter allows us to make the function more flexible.

writing CSV files Using csv.writer()

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

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

Example 3: Write to a CSV file

```
import csv
with open('protagonist.csv', 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(["SN", "Movie", "Protagonist"])
    writer.writerow([1, "Lord of the Rings", "Frodo Baggins"])
    writer.writerow([2, "Harry Potter", "Harry Potter"])
```

When we run the above program, a **protagonist.csv** file is created with the following content:

```
SN,Movie,Protagonist
1,Lord of the Rings,Frodo Baggins
2,Harry Potter,Harry Potter
```

In the above program, we have opened the file in writing mode.

Then, we have passed each row as a list. These lists are converted to a delimited string and written into the CSV file.

Example 4: Writing multiple rows with writerows()

If we need to write the contents of the 2-dimensional list to a CSV file, here's how we can do it.

```
import csv
csv_rowlist = [["SN", "Movie", "Protagonist"], [1, "Lord of the Rings", "Frodo
Baggins"],
                [2, "Harry Potter", "Harry Potter"]]
with open('protagonist.csv', 'w') as file:
    writer = csv.writer(file)
    writer.writerows(csv_rowlist)
```

The output of the program is the same as in **Example 3**.

Here, our 2-dimensional list is passed to the `writer.writerows()` method to write the content of the list to the CSV file.

Example 5: Writing to a CSV File with Tab Delimiter

```
import csv
with open('protagonist.csv', 'w') as file:
    writer = csv.writer(file, delimiter = '\t')
    writer.writerow(["SN", "Movie", "Protagonist"])
    writer.writerow([1, "Lord of the Rings", "Frodo Baggins"])
    writer.writerow([2, "Harry Potter", "Harry Potter"])
```

Notice the optional parameter delimiter= '\t' in the csv.writer() function.

The complete syntax of the csv.writer() function is:

```
csv.writer(csvfile, dialect='excel', **optional_parameters)
```

Similar to `csv.reader()`, you can also pass dialect parameter the `csv.writer()` function to make the function much more customizable.

Python csv.DictReader() Class

The objects of a `csv.DictReader()` class can be used to read a CSV file as a dictionary.

Example 6: Python csv.DictReader()

Suppose we have the same file **people.csv** as in **Example 1**.

Name	Age	Profession
Jack	23	Doctor
Miller	22	Engineer

Let's see how csv.DictReader() can be used.

```
import csv
```

```
with open("people.csv", 'r') as file:
    csv_file = csv.DictReader(file)
    for row in csv_file:
        print(dict(row))
```

Output

```
{'Name': 'Jack', ' Age': ' 23', ' Profession': ' Doctor'}
{'Name': 'Miller', ' Age': ' 22', ' Profession': ' Engineer'}
```

As we can see, the entries of the first row are the dictionary keys. And, the entries in the other rows are the dictionary values.

Here, `csv_file` is a `csv.DictReader()` object. The object can be iterated over using a `for` loop. The `csv.DictReader()` returned an `OrderedDict` type for each row. That's why we used `dict()` to convert each row to a dictionary.
Notice that, we have explicitly used the dict() method to create dictionaries inside the `for` loop.

```
print(dict(row))
```

Note: Starting from Python 3.8, csv.DictReader() returns a dictionary for each row, and we do not need to use dict() explicitly.
The full syntax of the `csv.DictReader()` class is:

```
csv.DictReader(file, fieldnames=None, restkey=None, restval=None,
dialect='excel', *args, **kwds)
```

To learn more about it in detail, visit: Python csv.DictReader() class

Python csv.DictWriter() Class

The objects of `csv.DictWriter()` class can be used to write to a CSV file from a Python dictionary.
The minimal syntax of the `csv.DictWriter()` class is:

```
csv.DictWriter(file, fieldnames)
```

Here,

- `file` - CSV file where we want to write to
- `fieldnames` - a `list` object which should contain the column headers specifying the order in which data should be written in the CSV file

Example 7: Python csv.DictWriter()

```
import csv

with open('players.csv', 'w', newline='') as file:
    fieldnames = ['player_name', 'fide_rating']
    writer = csv.DictWriter(file, fieldnames=fieldnames)

    writer.writeheader()
    writer.writerow({'player_name': 'Magnus Carlsen', 'fide_rating': 2870})
    writer.writerow({'player_name': 'Fabiano Caruana', 'fide_rating': 2822})
    writer.writerow({'player_name': 'Ding Liren', 'fide_rating': 2801})
```

The program creates a **players.csv** file with the following entries:

```
player_name,fide_rating
Magnus Carlsen,2870
Fabiano Caruana,2822
Ding Liren,2801
```

The full syntax of the `csv.DictWriter()` class is:

```
csv.DictWriter(f, fieldnames, restval='', extrasaction='raise',
dialect='excel', *args, **kwds)
```

To learn more about it in detail, visit: Python csv.DictWriter() class

Using the Pandas library to Handle CSV files

Pandas is a popular data science library in Python for data manipulation and analysis. If we are working with huge chunks of data, it's better to use pandas to handle CSV files for ease and efficiency.

Before we can use pandas, we need to install it. To learn more, visit: How to install Pandas?

Once we install it, we can import Pandas as:

```
import pandas as pd
```

To read the CSV file using pandas, we can use the `read_csv()` function.

```
import pandas as pd
pd.read_csv("people.csv")
```

Here, the program reads **people.csv** from the current directory.

To write to a CSV file, we need to call the `to_csv()` function of a DataFrame.

```
import pandas as pd

# creating a data frame
df = pd.DataFrame([['Jack', 24], ['Rose', 22]], columns = ['Name', 'Age'])

# writing data frame to a CSV file
df.to_csv('person.csv')
```

Here, we have created a DataFrame using the `pd.DataFrame()` method. Then, the `to_csv()` function for this object is called, to write into **person.csv**.