CSV File Reading and Writing

What is a CSV file?

A csv (Comma Separated File) is a file format for data storage. This file format organizes the information, storing containing one record per line, each field separated by a delimiter. The delimiter is usually a comma.

This format has been standardized in the RFC 4180. However, there is a lack of universal standard usage.

CSV files are easy to read and manage, are small in size, fast to process and transfer. Because of these benefits, they are frequently used in online stores and on many popular platforms, such as Magento.

In addition, most applications, such as Microsoft Excel, Notepad, and Google Docs, can be used to generate csv files.

The CSV Module

The csv module implements classes to operate with csv files. It is focused on the format preferred by Microsoft Excel. However, its functionality also tailors for csv files created with different delimiters and quoting characters.

Main CSV functions

This module provides the functions reader and writer, which work in a sequential manner. It also has the DictReader and DictWriter classes to manage data in the form of a dictionary.

1. The csv.reader(csv file name, dialect name, formatting parameters) method.

   As its name suggests, this method can be used to extract data from a csv file. The parameters are the csv file name and two optional formatting parameters: a dialect name and a list of formatting parameters that overwrite what was specified in the dialect.

   The method returns a reader object, which can be iterated. The data is read as a list of strings. If we specify the QUOTE_NONNUMERIC format, non-quoted values are converted into float values.

   An example on how to use this method is given below in this article.

2. The csv.writer(csv file name, dialect name, formatting parameters) method.

   Similar to the reader method we described above, this method permits us to write to a csv file. The parameters are the csv file name, and two optional ones: a dialect name, and formatting parameters. If we specify a dialect together with formatting parameters, the latter overwrites the values specified in the dialect.

A note of caution with this method. If the csv file specified is a file object, we need to open it with newline=''. If this is not specified, newlines inside quoted fields will not be interpreted in the correct manner, and depending on the working platform, extra characters, such as '\r' may be added.

3. Other features

   The csv module also gives us the DictReader and DictWriter classes, which allow us to read and write to files in dictionary manner.

   The class DictReader(csv file name, field names) works in a similar manner as a reader, but in Python 2 maps the data to a dictionary and in Python 3 to an OrderedDict. The keys are given by the field-names parameter. In addition, the method allows you to use dialects.

   The class DictWriter(csv file name, field names) works in a similar manner as a writer, but it maps the dictionary to output rows. However, since Python's dictionaries are not ordered, we cannot predict the row order in the output file. The method includes the option to use dialects.

4. Dialects

   A dialect is a construct that provides for the definition of groups of specific formatting parameters. It is a subclass of the Dialect class.

   Python offers two different ways to specify formatting parameters. The first is by declaring a subclass of this class, which contains the specific attributes. The second is by directly specifying the formatting parameters, using the same names as defined in the Dialect class.

   Dialects support several attributes. The most frequently used are:

   Dialogue.delimiter: used to separate fields. The default value is the comma (,).

   Dialogue.quotechar: used to quote fields containing special characters. The default is the double-quote (").

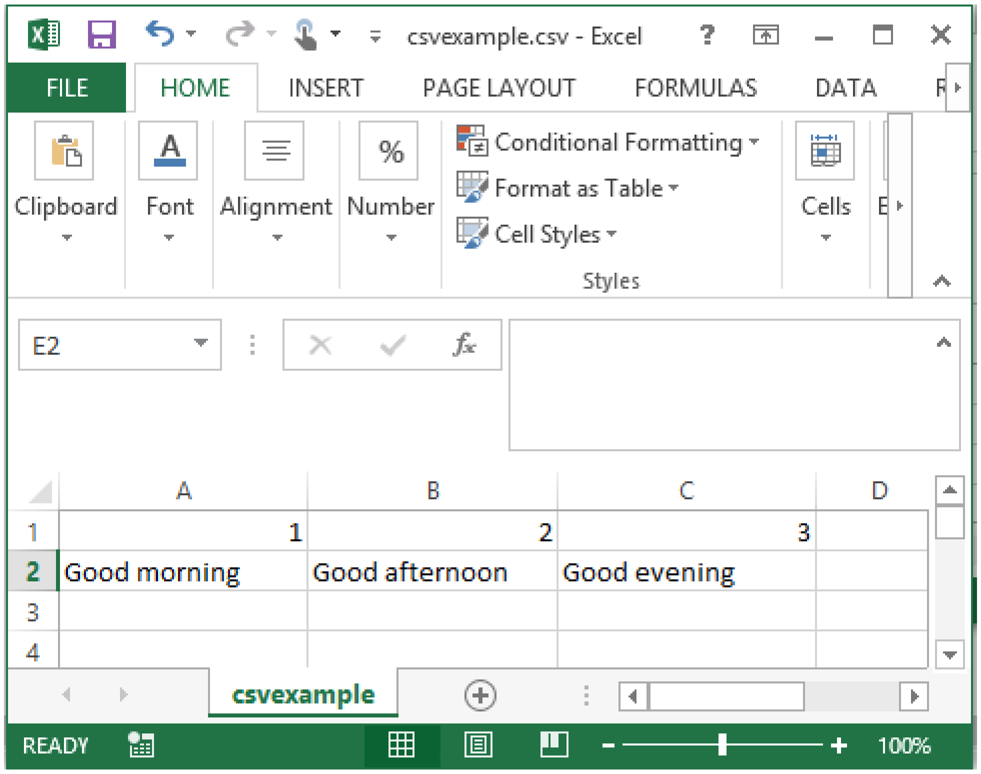
   Dialogue.lineterminator: used to terminate lines. The default is '\r\n'.

5. Versions

   Python 2.7 doesn't support Unicode input. Thus, all input should be in UTF-8 or printable ASCII formats.

CSV file example

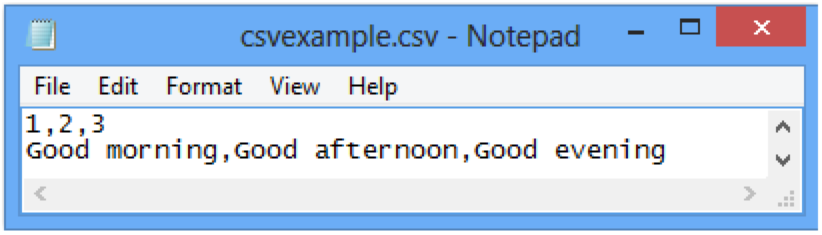
We can create a csv file easily with Excel. In the example below, the Excel file has a combination of numbers (1, 2 and 3) and words (Good morning, Good afternoon, Good evening), each of them in a different cell (see figure 1). The file was saved as csvexample.csv.



	A	B	C	D
1	1	2	3	
2	Good morning	Good afternoon	Good evening	
3				
4				

Figure 1

The structure of the csv file can be seen using a text editor, such as Notepad. Here, we can get the same values as in the Excel file, but separated by commas.

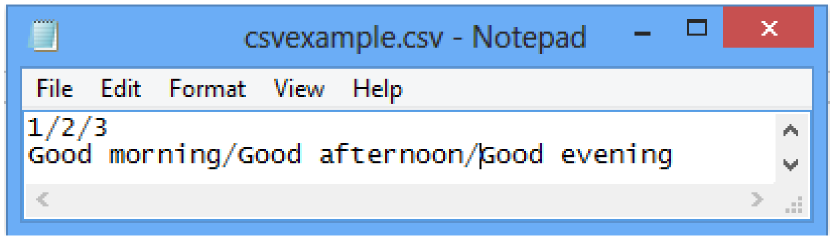


```
1,2,3
Good morning,Good afternoon,Good evening
```

Figure 2

We will use this file in the following examples.

We can change the delimiter to, for example, ‘/’. We can save this file as csvexample2.csv. It will look as follows:



```
1/2/3
Good morning/Good afternoon/Good evening
```

Figure 3

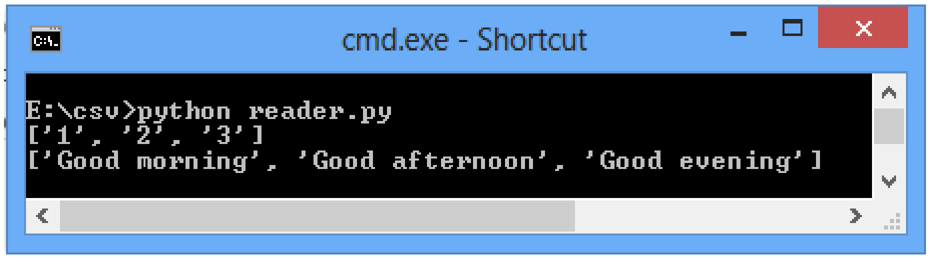
Reading CSV files

Reading a simple csv file

In this example we are going to read the csvexample.csv file previously created. The code is as follows:

```
1    import csv
2    with open('csvexample.csv', newline='') as myFile:
3        reader = csv.reader(myFile)
4        for row in reader:
5            print(row)
```

In line 1 we import the module csv. In line 2 we open our csvexample.csv file, and we create a reference to it named myfile. In line 3 we read the contents of the file, and in lines 4 and 5 we print on the screen the data extracted from csvexample.csv. If we save the code in a file named reader.py and we run it, the result will be as follows:



```
E:\csv>python reader.py
['1', '2', '3']
['Good morning', 'Good afternoon', 'Good evening']
```

Figure 4

As we can see from figure 4, we obtain the contents of the csvexample.csv file on the screen.

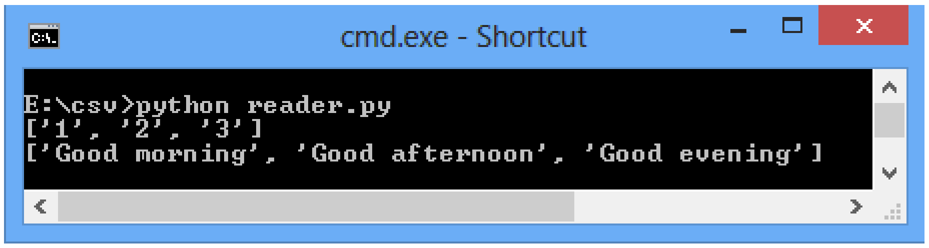
Changing the delimiter

The csv module allows us to read csv files when some of the file format characteristics are different from the usual ones. For example, we can read a file with a different delimiter. In our example, csvexample2.csv, we have replaced the comma with a forward slash.

In order to perform this task, we must modify code, indicating the new delimiter used. In this example, we have saved the code in a file named reader2.py. The modified program is a follows:

```
1   import csv
2   with open('csvexample2.csv', newline='') as myFile:
3     reader = csv.reader(myFile, delimiter='/', quoting=csv.QUOTE_NONE)
4     for row in reader:
5         print(row)
```

As we can see from the code above, we have modified line 3, by adding the delimiter parameter and assigning a value of '/' to it. We have also added the quoting parameter, and assigned it a value of csv.QUOUTE.NONE, which means that we are not using any special quoting. As expected, the result is similar to the previous example (see figure 5).



```
E:\csv>python reader.py
['1', '2', '3']
['Good morning', 'Good afternoon', 'Good evening']
```

Figure 5

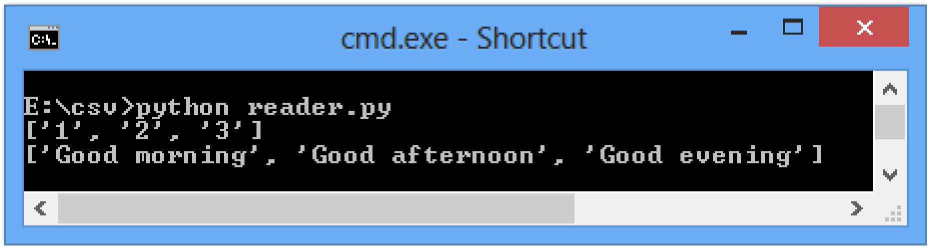
Creating a dialect

The CSV module permits us to create a dialect with the characteristics of our CSV file. Thus, the example above can be done with the following code:

```
1      import csv
2      csv.register_dialect('myDialect', delimiter='/', quoting=csv.QUOTE_NONE)
3      with open('csvexample2.csv', newline='') as myFile:
4        reader = csv.reader(myFile, 'myDialect')
5        for row in reader:
6            print(row)
```

Here, in line 2, we have defined a dialect with the same parameters as in the previous example: forward slash as delimiter, and no quoting.

If we save this code in a file named reader3.py and we run it, the result will be as follows:



```
E:\csv>python reader.py
['1', '2', '3']
['Good morning', 'Good afternoon', 'Good evening']
```

Figure 6

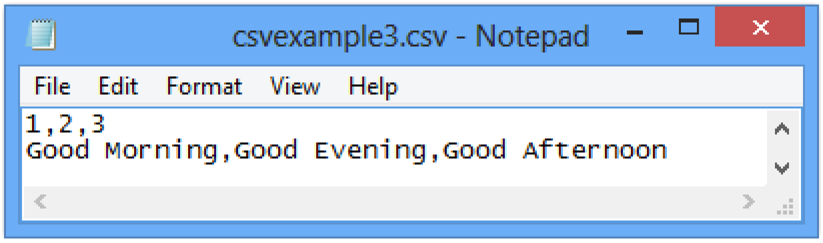
As we can see, this output is exactly the same as the one in figure 5.

Writing to CSV files

The CSV module has good functionality to write data to a CSV file. The writer class presents two functions, namely writerow() and writerows(). The difference between them is that the first function only writes one row. The function writerows() writes several rows at once.

The code below creates a list of data. Each element in the list represents a row in the CSV file. Then, it opens a CSV file named csvexample3.csv, creates a writer object, and sends the data to the file using the writerows() method.

```
import csv
myData = [[1, 2, 3], ['Good Morning', 'Good Evening', 'Good Afternoon']]
myFile = open('csvexample3.csv', 'w')
with myFile:
   writer = csv.writer(myFile)
   writer.writerows(myData)
```



```
1,2,3
Good Morning,Good Evening,Good Afternoon
```

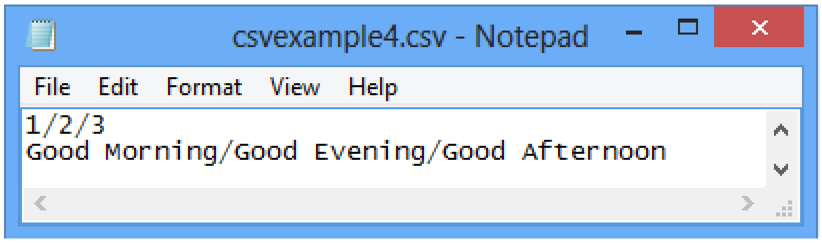
Figure 7

The writer object also caters for different CSV formats. The following example creates a dialect with '/' as delimiter.

```
import csv
myData = [[1, 2, 3], ['Good Morning', 'Good Evening', 'Good Afternoon']]
csv.register_dialect('myDialect', delimiter='/', quoting=csv.QUOTE_NONE)
myFile = open('csvexample4.csv', 'w')
with myFile:
```

```
writer = csv.writer(myFile, 'myDialect')
writer.writerows(myData)
```

Running the code above results in the following output:



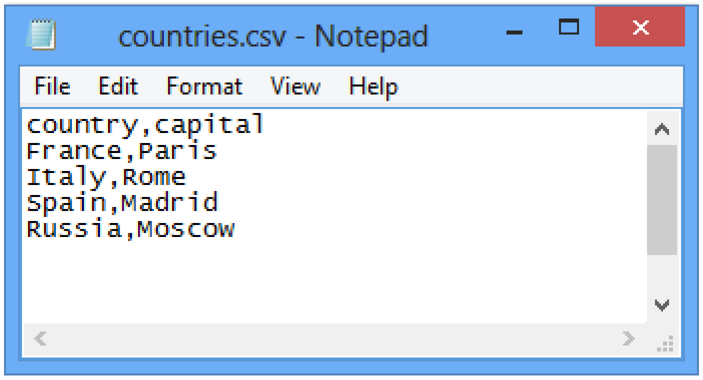
```
1/2/3
Good Morning/Good Evening/Good Afternoon
```

Figure 8

Using dictionaries

Reading a file

Using a text editor, we create a CSV file named countries.csv with the following content:



```
country,capital
France,Paris
Italy,Rome
Spain,Madrid
Russia,Moscow
```

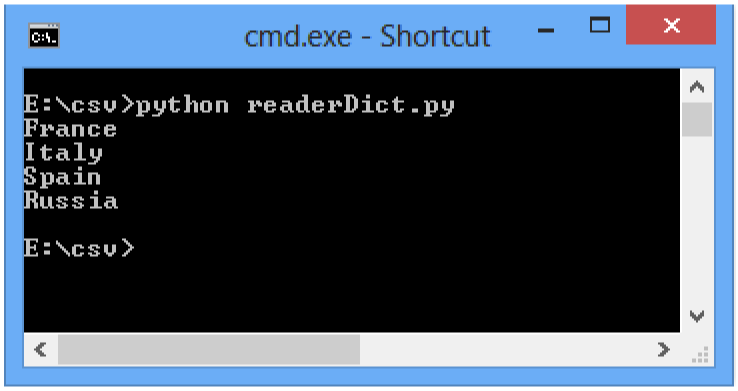
Figure 9

The first row in this file contains the field names. The following rows contain pairs of values (country, capital) separated by a comma.

In order to read this file, we create the following code:

```
import csv
with open('countries.csv') as myFile:
   reader = csv.DictReader(myFile)
   for row in reader:
      print(row['country'])
```

Running the code will loop through the rows in the reader object, and print the values under the field country. We could have added the field capital if we wanted. The result is as follows:



```
E:\csv>python readerDict.py
France
Italy
Spain
Russia

E:\csv>
```

Figure 10

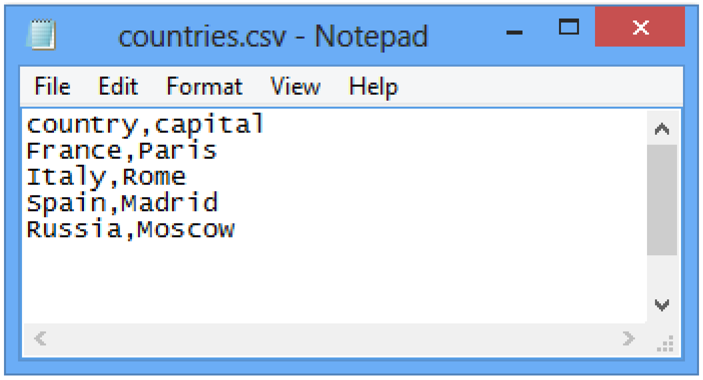
Writing to a file

We can also create a CSV file using dictionaries. In the code below, we create a dictionary with the country and capital fields.

Then, we tell the computer to create a writer object that refers to our countries.csv file, and that has the set of fields previously defined with the list myFields.

Following it, we ask the computer to write the fields with the writeheader() method, and the pairs of values using the writerow method. The result is shown in figure 11 below.

```
import csv
myFile = open('countries.csv', 'w')
with myFile:
   myFields = ['country', 'capital']
   writer = csv.DictWriter(myFile, fieldnames=myFields)
   writer.writeheader()
   writer.writerow({'country' : 'France', 'capital': 'Paris'})
   writer.writerow({'country' : 'Italy', 'capital': 'Rome'})
   writer.writerow({'country' : 'Spain', 'capital': 'Madrid'})
   writer.writerow({'country' : 'Russia', 'capital': 'Moscow'})
```



```
country,capital
France,Paris
Italy,Rome
Spain,Madrid
Russia,Moscow
```

Figure 11

Final words

CSV files are a handy file storage format: they are small, easy to manage and widely used. Phyton has a dedicated module for them that provides tools for managing CSV files in a straightforward and efficient manner.