3.1 The CSV module in Python

The CSV module in Python

The CSV (Comma Separated Values) format is one of the most popular file formats used to store and transfer data between different programs. Currently, many database management tools and the popular Excel offer data import and export in this format.

The CSV file is a plain text file with the .csv extension. A typical file contains comma-separated values, but other separators such as semicolon or tab are also allowed. It should be emphasized that only one type of separator can be used in one CSV file.

Each line in the file represents a certain set of data. Optionally, in the first line we can put a header that describes this data. Let's look at a simple example of a file called `contacts.csv` that stores contacts from a phone:

```
Name,Phone
mother,222-555-101
father,222-555-102
wife,222-555-103
mother-in-law,222-555-104
```

In the above file, there are four contacts consisting of name and phone number. Note that the first line contains a header to help you interpret the data.

During this course, you'll learn how to use the `csv` module provided by the Python Standard Library. However, for commercial projects, we recommend using an excellent library called `pandas`. We encourage you to familiarize yourself with its possibilities in your free time.

The CSV format is really simple. Let's find out whether reading data from a CSV file in Python is equally simple.

Reading data from a CSV file (part 1)

The Python Standard Library offers a module called `csv` that provides functions for reading and writing data in CSV format. Reading data is done using the `reader` object, while writing is done using the `writer` object. First, we'll take a closer look at reading data using the reader object.

The `reader` function returns an object that allows you to iterate over each line in the CSV file. To create it, we need to pass a file object to the `reader` function. For this purpose, we can use a built-in function called `open`. Look at the code in the editor and run it.

```
import csv

with open('contacts.csv', newline='') as csvfile:
    reader = csv.reader(csvfile, delimiter=',')
    for row in reader:
        print(row)
```

It should produce the following output:

```
['Name', 'Phone']
['mother', '222-555-101']
```

```
['father', '222-555-102']
['wife', '222-555-103']
['mother-in-law', '222-555-104']
```

What happened? We've passed an open file named `contacts.csv` and a separator used to separate the data in the file to the `reader` function. The second argument can be omitted if our file uses the default separator, which is a comma - we've added it to show you how to specify other separators.

Reading data from a CSV file (part 2)

Finally, we read each row using the `for` loop. Note that a single line is returned as a list of strings. However, more readable results can be obtained, e.g., by using the `join` method. Look at the code in the editor and run it.

```
import csv

with open('contacts.csv', newline='') as csvfile:
    reader = csv.reader(csvfile, delimiter=',')
    for row in reader:
        print(','.join(row))
```

It should produce the following result:

```
Name,Phone
mother,222-555-101
father,222-555-102
wife,222-555-103
mother-in-law,222-555-104
```

NOTE: The `newline=''` argument added to the `open` function protects us from incorrect interpretation of the newline character on different platforms.

Reading data from a CSV file (part 3)

The `csv` module provides a more convenient way to read data, in which each line is mapped to an `OrderedDict` object. To achieve this, we must use the `DictReader` class in the way we show in the editor.

```
import csv

with open('contacts.csv', newline='') as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        print(row['Name'], ':', row['Phone'])
```

The code in the editor will produce the following output:

```
mother : 222-555-101
father : 222-555-102
wife : 222-555-103
mother-in-law : 222-555-104
```

Reading data from a CSV file (part 4)

Like the `reader` function, the `DictReader` class accepts a file object as an argument. It treats the first line of the file as a header from which to read the keys. If your file doesn't have a header, you must define it using the `fieldnames` argument. Look at the code in the editor.

```
import csv

with open('contacts.csv', newline='') as csvfile:
    fieldnames = ['Name', 'Phone']
    reader = csv.DictReader(csvfile, fieldnames=fieldnames)
    for row in reader:
        print(row['Name'], row['Phone'])
```

NOTE: If you define more column names than the values in the file, the missing values will be None.

Saving data to a CSV file (part 1)

As we mentioned before, saving data to a CSV file is done using the `writer` object provided by the `csv` module. To create it, we need to use a function called `writer`, which takes the same set of arguments as the reader function. Let's see how to save contacts to a CSV file. Look at the code in the editor.

```
import csv

with open('exported_contacts.csv', 'w', newline='') as csvfile:
    writer = csv.writer(csvfile, delimiter=',')

    writer.writerow(['Name', 'Phone'])
    writer.writerow(['mother', '222-555-101'])
    writer.writerow(['father', '222-555-102'])
    writer.writerow(['wife', '222-555-103'])
    writer.writerow(['mother-in-law', '222-555-104'])
```

In the example code, we first open the file for writing. The `'w'` mode creates a file for us if it hasn't already been created. Next, we create a `writer` object that we use to add rows using the `writerow` method. The `writerow` method takes a list of values as an argument, and then saves them as one line in a CSV file.

Saving data to a CSV file (part 2)

Imagine a situation where you add a contact containing the separator used to separate the values in the CSV file. By default, these values are quoted, but you can change this with the `quotechar` argument, which must be a single character. Look at the code in the editor and notice how we explicitly set default arguments.

```
import csv

with open('exported_contacts.csv', 'w', newline='') as csvfile:
    writer = csv.writer(csvfile, delimiter=',', quotechar='"',
quoting=csv.QUOTE_MINIMAL)

    writer.writerow(['Name', 'Phone'])
    writer.writerow(['mother', '222-555-101'])
```

```
    writer.writerow(['father', '222-555-102'])
    writer.writerow(['wife', '222-555-103'])
    writer.writerow(['mother-in-law', '222-555-104'])
    writer.writerow(['grandmother, grandfather', '222-555-105'])
```

The code will save the following data to the *exported_contacts.csv* file:

```
Name,Phone
mother,222-555-101
father,222-555-102
wife,222-555-103
"grandmother, grandfather and auntie",222-555-105
```

The last argument called `quoting`, specifies what values should be quoted. The default value `QUOTE_MINIMAL` means that only values with special characters such as separator or `quotechar` will be quoted. In our case, it's the value of «grandmother, grandfather and auntie».

Below are other constants that we can use as the value of the `quoting` argument:

- **csv.QUOTE_ALL** – quotes all values
- **csv.QUOTE_NONNUMERIC** – quotes only non-numeric values
- **csv.QUOTE_NONE** – doesn't quote any values. It's not a good idea to set this value if you have special characters that require quoting, because this will raise an error

NOTE: The quotechar and quoting parameters can also be used in the reader function. See the documentation for more information.

Saving data to a CSV file (part 3)

Do you remember how we read rows from the CSV file to `OrderedDict` objects? In the `csv` module, there is an analogous class called `DictWriter` with which we can map dictionaries to rows. Unlike the `DictReader` object, when creating the `DictWriter` object, we must define a header. Let's look at the example in the editor.

```
import csv

with open('exported_contacts.csv', 'w', newline='') as csvfile:
    fieldnames = ['Name', 'Phone']
    writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

    writer.writeheader()
    writer.writerow({'Name': 'mother', 'Phone': '222-555-101'})
    writer.writerow({'Name': 'father', 'Phone': '222-555-102'})
    writer.writerow({'Name': 'wife', 'Phone': '222-555-103'})
    writer.writerow({'Name': 'mother-in-law', 'Phone': '222-555-104'})
    writer.writerow({'Name': 'grandmother, grandfather and auntie', 'Phone':
'222-555-105'})
```

To create the `DictWriter` object, we use a file object and a list containing column names. Note that before saving the value, we first call the `writeheader` method, which adds the header to the first line of the file. After that we add rows with values by passing dictionaries to the `writerow` method.