Theory: Working with CSV

20 minutes 0 / 5 problems solved

In previous topics, you have learned how to work with a text file. It is a useful skill to master. Right now, imagine that you were hired as a programmer at a well-known company. During your first day, you were asked to save information about every crime in your city. It includes the date, the district, exact time, etc. You are horrified because 100000 crimes have been committed during this month!

It is inconvenient to store this much data as a file with plain text. Your information is going to be very hard to read. Don't get upset! You can use **CSV format** to save all your information. In this topic, we will discuss what a CSV file is and how to work with it.

§1. What is CSV?

CSV stands for comma-separated values. A CSV file allows you to store your data in a tabular format. In fact, it is a simple text file with numerous lines. Each line of the file is a **data entry**. Each line contains elements separated by commas. The CSV data can be easily exported to other spreadsheets or databases. Let's have a look at an example of CSV:

	Crime	Time	District
2	Murder	10:03	1
3	Shop-lifting	21:24	2
4	Murder	13:15	2
5	Shop-lifting	16:17	3
6	Arson	15:59	5
7	Pickpocketing	22:22	4
8	Murder	23:43	5

```
Crime,Time,District
Murder,10:03,1
Shop-lifting,21:24,2
Murder,13:15,2
Shop-lifting,16:17,3
Arson,15:59,5
Pickpocketing,22:22,4
Murder,23:43,5
```

Our file is called *City_crimes.csv*, it can be opened with Microsoft Excel (left picture). The right picture presents the same file opened in Notepad. It contains information about different crimes registered in an imaginary city. The first line is a line with column titles separated by commas: Crime, Time, and District. There are other lines (entries) with certain values that characterize a crime. They are also separated by commas.

If you don't like commas, you can use any other separator instead, but when reading your CSV-file, you need to know in advance which separator is used.

Let's see how we can work with CSV in Python.

§2. Reading CSV

We can read a CSV file like any other text file in Python. Take a look at the code below. We are about to open the *City_crimes.csv* file from the previous section.

```
crimes = open('City_crimes.csv', 'r')
for line in crimes:
    print(line, end='')
# Crime,Time,District
# Murder,10:03,1
# Shop-lifting,21:24,2
# Murder,13:15,2
# Shop-lifting,16:17,3
# Arson,15:59,5

# Pickpocketing,22:22,4

# Murder,23:43,5
```

There is nothing extra about reading lines from files, but it may be not very inconvenient to obtain particular information from the lines. We can use the `line.split()` and pass a comma as a separator. Now let's have a look at the

Current topic:
Working with CSV

Topic depends on:
String formatting Stage 1 7
Indexes Stage 3 7
Dictionary Stage 1 6
Else statement Stage 1 15
Split and join Stage 1 8
Context manager Stage 2

Table of contents:
↑ Working with CSV
§1. What is CSV?
§2. Reading CSV
§3. Writing to CSV files
§4. CSV library
§5. CSV writer
§6. DictReader VS DictWriter
§7. Conclusion
Feedback & Comments

updated snippet.

```
for line in crimes:
    print(line.split(","))
# ['Crime', 'Time', 'District\n']
# ['Murder', '10:03', '1\n']
# ['Shop-lifting', '21:24', '2\n']
# ['Murder', '13:15', '2\n']
# ['Shop-lifting', '16:17', '3\n']
# ['Arson', '15:59', '5\n']
# ['Pickpocketing', '22:22', '4\n']

# ['Murder', '23:43', '5\n']
```

We have the resulting data lists. Note that the last element in each list has the `\n` symbol of a new line, it is recognized as **the end of a data entry**. Now our results are easy to process, and we can extract any information we need. When you don't need this file anymore, don't forget to close it.

```
crimes.close()
```

In the following section, we will discuss how to write to CSV files.

§3. Writing to CSV files

As you can guess, the idea of writing to a CSV file doesn't differ from the idea of writing something to any text file. Imagine your colleague brought you a new report in which you saw a crime that isn't stored in your CSV file. So, you decided to append it to the end of the file. What do you do? You create a list in which you write down all the necessary information about the crime, then you join the elements of the list with a comma, write the data into the file. Finally, you need to close the file.

```
crimes = open("City_crimes.csv", "a")
add_data = ['Hijacking', '17:49', '5']
crimes.write(','.join(add_data) + '\n')
crimes.close()
```

What happens to our file?

	Crime	Time	District
2	Murder	10:03	1
3	Shop-lifting	21:24	2
4	Murder	13:15	2
5	Shop-lifting	16:17	3
6	Arson	15:59	5
7	Pickpocketing	22:22	4
8	Murder	23:43	5
9	Hijacking	17:49	5

As you can see, our data entry is saved at the end of the file.

All ways of working with a CSV file described above are probably familiar to you. However, there's another feature that can be useful: the special CSV library.

§4. CSV library

The **CSV library** is the main tool for working with CSV files. It is built-in, so you can just import it.

```
import csv
```

To read data from a file, you should create a **reader** object. In the example below, the `csv.reader` returns a reader object that will iterate over lines in the given CSV file.

```
with open("City_crimes.csv", newline='') as crime:

file_reader = csv.reader(crime, delimiter=",")  # Create a reader object
    for line in file_reader:                        # Read each line
        print(line)
# ['Crime', 'Time', 'District']
# ['Murder', '10:03', '1']
# ['Shop-lifting', '21:24', '2']
# ['Murder', '13:15', '2']
# ['Shop-lifting', '16:17', '3']

# ['Arson', '15:59', '5']

# ['Pickpocketing', '22:22', '4']

# ['Murder', '23:43', '5']
```

As you can see, the result is the same as before. Each line returned by the reader is a list of string elements with the data.

When we open a CSV file, we need to specify `newline=''`. It's better to do it this way because if the `newline` is not mentioned, the older versions of Python will add the line break after the last element.

Moving on with our example, let's say your boss asked you to print some of the results: the crime and its time. She also asked to make them readable for other people. Below is an example of how we can improve our code to do that.

```
with open("City_crimes.csv") as crime:
    file_reader = csv.reader(crime, delimiter=",")
    count = 0
    for line in file_reader:
        if count == 0:
            print(f'Column names are {", ".join(line)}', '\n')
            count += 1
        else:
            print('The crime is', line[0])

            print('The time of the crime is', line[1], '\n')

            count += 1

# Column names are Crime, Time, District

#

# The crime is Murder

# The time of the crime is 10:03

#

# The crime is Shop-lifting

# The time of the crime is 21:24

# ...
```

What have we changed there? First of all, we added the line counter. If it is equal to 0, it means that we are reading the first line, so we'll print the names of the columns. If it is not 0, then we print all the necessary information by extracting it from the lists with the help of indexing. Don't forget that the indexing of lists starts with 0.

§5. CSV writer

Another object that is widely used in the CSV library is a **writer** object. As the name suggests, it helps to write information to a file. Let's say that policemen have found some people who had committed those crimes. Now you have to create a new file with their names, age, and height. So it would be:

```
with open("criminals.csv", "w", encoding='utf-8') as w_file:
    file_writer = csv.writer(w_file, delimiter=",", lineterminator="\n")
    file_writer.writerow(["Name", "Age", "Height"])
    file_writer.writerow(["Alex", "23", "184"])
    file_writer.writerow(["Karla", "35", "170"])
    file_writer.writerow(["Tim", "21", "178"])
```

In this example, we created a new file and then used the `file_writer.writerow()` method to write new information. The `lineterminator` parameter is the separator between the data entries. The first data entry contains names of columns, all the others contain the perpetrators. Now we can also have a look at the file we created.

	A	B	C
1	Name	Age	Height
2	Alex	23	184
3	Tom	35	170
4	Tim	21	178

You see, it works well!

§6. DictReader VS DictWriter

Apart from all mentioned above, the CSV library also has two magical classes: `csv.DictReader()` and `csv.DictWriter()`. They represent each data entry as **a dictionary**. The dictionary keys are the names of our columns, and values are corresponding data. As it is a dictionary, you can print particular information using keys. Mind the following code with `csv.DictReader()`.

```
with open("City_crimes.csv") as crime:
    file_reader = csv.DictReader(crime, delimiter=",")
    for line in file_reader:
        print(line['Crime'], line['Time'], line['District'])
# Murder 10:03 1
# Shop-lifting 21:24 2
# Murder 13:15 2
# ...
```

A similar tool for writing information is the `csv.DictWriter()`. Let's use our *criminals.csv* file and write something about criminals again to that file.

```
with open("criminals.csv", "w", encoding='utf-8') as w_file:
    names = ["Name", "Age", "Height"]

 file_writer = csv.DictWriter(w_file, delimiter=",", lineterminator="\n", field
names=names)
    file_writer.writeheader()
    file_writer.writerows([{"Name": "Alex", "Age": "23", "Height": "184"},
                           {"Name": "Tom", "Age": "35", "Height": "170"},
                           {"Name": "Tim", "Age": "21", "Height": "178"}])
```

First, we created a list of column titles that will be given as an argument to the **fieldnames** parameter. This list is a sequence of dictionary keys, it identifies the order in which values will be written to the *criminals.csv*. Then, we stored the titles using the `writeheader()` and the data using the `writerows()`. The CSV file will look the same as we have shown in one of the previous sections.

§7. Conclusion

Working at a well-known company is a tough job but we hope that it will be easier for you as you learned how to deal with CSV files. So far, you know:

- that CSV stands for comma-separated value, this file format is used for storing tabular data;
- how to read data manually from the file and write information to it;
- that Python has the built-in CSV library with useful `csv.reader()` and `csv.writer()` methods;
- that `csv.DictReader()` and `csv.DictWriter()` help you represent data as dictionaries.

Of course, we can't cover all the aspects. If you strive to learn more, read the official documentation and PEP 305. Now let's proceed to the tasks to check your knowledge.

Report a typo

16 users liked this theory. **0** didn't like it. **What about you?**

Start practicing

This content was created about 1 month ago and updated about 9 hours ago. Share your feedback below in comments to help us improve it!

Comments (0) Hints (0) Useful links (0) Show discussion