Using .csv Files

November 30, 2020

Administrative Notes

Keep grinding away on Project 3 - it's due Friday night

Lab 12 is this week!!

Next Monday (December 7) is the last lecture. It will be a review of the course material in preparation for the Final.

- There will be a sample Final posted later this week
- The final is Monday, December 14 at (insert time here). Online; on Blackboard; like the two midterms. You will have two hours to finish.

Something about course evaluations

Lab 12

Some notes about virtual environments and Lab 12

- “Virtual” environments are software-based representations of a computer system
- Your code can operate as if it were running on a system by itself, without damaging actual system resources
- Lab 12 has you set up a virtual environment in your spaces to practice with this technology

Lab 12 also has you install packages/modules to help perform tasks

- Just like you’re doing on our Project 3!!!

Tonight’s lecture - comma-separated values (csv) files

This is the first of two “special topics” lectures. Wednesday will be the second

A .csv file consists of lines of text, with each line separated by a newline character (‘\n’) and each item on each line separated by a comma. (pop to an example of a .csv file as text)

.csv files are commonly used to transfer data between applications. It’s a generic file type. They can be created and read by applications such as Excel, Google Sheets, Apple’s Numbers,...

Reading from a .csv file

(In Project 3 we let you use some custom-written code to read in a .csv file. Now you're going to learn to do it on your own, the hard way.)

If you know that the file you're reading from is a .csv file, the best approach is:

- Step 1, use readline() or readlines() to read the file in a line at a time. You know that the lines in the file are separated by \n, so let Python do the separating for you
- Step 2, use split(“,”) to split each line into its component elements. Each line is read in as a string. Split that string on the commas. This will throw away the commas, and separate the actual elements into their proper place
- Step 3, all components are still strings. Convert elements to ints, floats, etc. as necessary.

Writing to a .csv file

Remember that you can only write one string to a file at a time.

So you have to put everything together into a string to write it.

Use the join() command to create the strings.

- Using “,”.join() automatically puts the commas into place between the components of the string - the “values” to be “separated” by “commas”
- But remember how join works - it only works on iterables (lists or dictionaries) and the components of those lists or dictionaries have to themselves be strings

Writing to .csv

An example:

```
l = [‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’]
```

Using

```
s = “,”.join(l)
```

gives you a string with elements separated by commas, which you can now write to your file

What if instead we had

```
l = [1,2,3,4,5,6]
```

What does

```
s = “,”.join(l)
```

produce?

write() vs writelines()

“write()” writes a single string to a file. Using this means you write one line at a time

“writelines()” lets you write multiple strings to a file in a single statement. If you have a list of strings, you can write them all to the file with a single use of “writelines”

- Our examples will use “write()” for simplicity

An example

```
with open ("C:\\Users\\Al\\PycharmProjects\\exam_qs\\data.csv", "w") as outfile:
    l = ['1','2','3','4','5','6']
    s = ",".join(l)
    print (s)
    outfile.write(s)
    outfile.write('\n')
    outfile.write(s)
```

This is the full path on a Windows machine. You have to escape the backslash character to make Windows understand what you want. That's why the double "\\" exist.

How to deal with commas in a csv file

Quick sidebar - what if one of your data items contains a comma? Suppose that you have a field “name” which you’ve developed to be “Lastname, Firstname, MI”. That is, the string

> Arsenault, Alfred, W.

is supposed to be a single value, not three different values. But if you treat the comma as a separator, you’d have three different values.

Cheat code: Python provides a whole module that handles csv files for you:

import csv

The short answer:

If you're writing a file, embed the comma in the string before you do the ",".join(). Then use something like quotes around the string so that the comma will later be skipped.

If you're reading the file, you might run into problems

- There's no explicit way to mark a comma in the file as "this is a separator" vs. "This is an actual data value." You hope that the person who created the file anticipated this for you.
- You're going to have do some EDA - exploratory data analysis - to see if you are going to have that problem
- If you do run into the problem, you'll have to write code to address it on a case-by-case basis.