As part of an earlier course on "Algorithmic Thinking", we created infrastructure that allowed students to plot cancer-risk due to air toxics on a county-level map of the United States. The practice projects for this week and the next will focus on some of the scripting tasks necessary to bring the raw air pollution data (provided by the Environmental Protection Agency) into a form suitable for use. The practice projects for the last course in this specialization will then focus on the scripting tasks required to create an image similar to the one linked here.

This two-part project is designed to give you more substantial experience in working with tabular data (especially CSV files). You are encouraged to complete this **optional** practice project before you attempt the required project for this week. Remember that, if you are struggling, you are welcome to examine our solution for the practice project provided at the end of the description.

Practice Project: Loading Cancer-Risk Data

The data that we will process in the practice project was generated in 2005 by the Environmental Protection Agency as part of an effort to understand the affect of air toxics on human health. The specific county-level data on cancer-risk from air toxics is stored in an .xls file located here. As part of our initial processing of this data, we have downloaded and manually removed some of the extra text from the data set. This processed CSV file which will be a critical component of our remaining practice projects is available on Google Storage here.

The structure of the cancer-risk CSV file

The data file provided above for this project contains a 2D table of data whose rows are indexed by the states and counties of the United States and whose corresponding columns contain the following information:

- Column A - State name
- Column B - County name
- Column C - Unique five-digit FIPS county county code
- Column E - Population
- Column L - Lifetime cancer-risk

To familiarize yourself further with this data set, we suggest that you download the CSV file and examine its contents using an tool like Excel.

Required functions

For this week's part of the practice project, your task is to implement two functions that will read and write CSV files. These functions are:

- **`read_csv_file(file_name)`** - Given a file path specified as the string, **`file_name`** , load the associated CSV file and return a nested list whose entries are the fields in the CSV file. Each entry in the returned table should be of type **`str`**
- **`write_csv_file(csv_table, file_name)`** - Given a nested list **`csv_table`** and a file path specified as the string, **`file_name`**, write entries in the nested list as the fields of a comma-separated CSV file with the specified path.

Since the example code for this week's programming tip contains almost a full solution to this part of the practice project, we recommend that you do **not** use it as a starting point (or even refer to it). Instead, we recommend that you use the project template that is available here. The template includes a function **`print_table(table)`** that you can use to echo the loaded CSV data to the console.

Note this practice project **should be implemented in desktop Python** since CodeSkulptor3 does not implement the **`csv`** module. On a related note, remember to avoid naming your solution file with same name as a module that you import. For example, naming your solution file "csv.py" will cause Python to raise an error when then try to import the **`csv`** module in the file.

Testing your functions

As always, you should test your functions to make sure that they correctly read and write CSV files. To help you with this task, we have provided part of a function **`test_part1_code()`**. This function reads a small CSV file test_case.csv and then uses **`print_table()`** to display the results in the console. We suggest that view this small test file in Excel and then compare the result to what is displayed in the console.

Your code for writing CSV files can be tested in a similar way for small examples. However, testing your code on larger data sets by visually comparing results is problematic since it is easy to overlook small differences in the results. One strategy for checking the correctness of your read and write functions is to load the cancer-risk data into Python, write the data set to a second new CSV file, and the reload this second file. If you're reading and writing code works correctly, the loaded table and the reloaded table should be identical.

The second part of **`test_part1_code()`** includes starter code for this check. However, you will need to add code to this part of the test that compares the contents of two tables. Once you are satisfied with the correctness of your code, you are welcome to compare your solution to our solution that is available here.

Hints for the project

- Remember to use **`with`** to ensure that the files are closed correctly when you have finished working with them.
- In **`read_csv_file()`**, use the correct **`delimiter`** option with **`csv.reader()`** when reading the provided CSV files.
- In **`write_csv_file()`**, experiment with values for the **`quoting`** option inside **`csv.writer()`**. Since some of the county names include single quotes as apostrophes (e.g. **`"O&#x27;Brien"`**) , you will need to carefully consider how quotes are handled during the reading and writing of the provided CSV files.

Mark as completed