Project Description: Reading and Writing CSV Files

Project Description: Reading and Writing CSV Files

This week's practice project investigated using a list to represent a row of a CSV file. In this project, we will use dictionaries to represent a row of a CSV file. This dictionaries will then be organized using either a list or a dictionary.

Preliminaries: Working on the Project

Coding Style

In this class, you will be asked to strictly follow a set of coding style guidelines. Good programmers not only get their code to work, but they also write it in a way that enables others to easily read and understand their code. Please read the style guidelines carefully and get into the habit of following them right from the start. A portion of your grade on the project will be based upon coding style.

Testing

You should always test your code as you write it. Do not try to solve the entire project before running it! If you do this, you will have lots of errors that interact in unexpected ways making your program very hard to debug. Instead, as you write each function, make sure you test it to ensure that it is working properly before moving on to the next function.

Throughout this course, we will be using a machine grader (OwlTest) to help you assess your code. You can submit your code to this Owltest page to receive a preliminary grade and feedback on your project. The OwlTest page has a pale yellow background and does **not** submit your project to Coursera. OwlTest is just meant to allow you to test your project automatically. Note that trying to debug your project using the tests in OwlTest can be very tedious since they are slow and give limited feedback. Instead, we *strongly* suggest that you first test your program using your own tests. Also, note that each OwlTest link is specific to a particular project. You need to come back to this page and click the link above to ensure that you are running the tests for this project.

When you are ready to submit your code to be graded formally, submit your code to the assignment page for this project. You will be prompted to open a tool which will take you to the Coursera LTITest page. Note that the Coursera LTITest page looks similar to the OwlTest page, but they are *not* the same! The CourseraLTI Test page has a **white** background and does submit your grade to Coursera.

Note: Due to recent changes in Google App Engine, we have updated the machine grader for this course. If you experience any unexpected issues with OwlTest or CourseraTest (e.g; the grader crashes or doesn't record your grade), feel free to email *interactivepython@rice.edu* with a short description of your issue and a copy of your code.

Project: Reading and Writing CSV Files

We suggest you start by reviewing the Python documentation on the csv module. Your code will rely in the DictReader() and DictWriter() methods from this module. We have provided the following template that you can use to get you started. Note this project **should be implemented in desktop Python** since CodeSkulptor3 does not implement the `csv` module.

This template includes the signatures (name, parameters, and docstrings) for all of the functions that you will need to write. The code however, simply returns some arbitrary value no matter what the inputs are, so you will need to modify the body of the function to work correctly. You should not change the signature of any of the functions in the template, but you may add any code that you need to. You can also download all of the files used by OwlTest when testing your code as a zip file.

Problem 1: Reading the field names from a CSV file

First, you will write a function called `read_csv_fieldnames` that takes the name of a CSV file and returns a list of the field names from that file. This function assumes that the first row of the CSV file contains the field names. As CSV files can use different separator characters and quote characters, this function takes those characters as input and uses them to properly parse the CSV file.

Here is the signature of the `read_csv_fieldnames` function:

```
def read_csv_fieldnames(filename, separator, quote):
    """
    Inputs:
      filename  - name of CSV file
      separator - character that separates fields
      quote     - character used to optionally quote fields
    Output:
      A list of strings corresponding to the field names in
      the given CSV file.
    """
```

You need to write the code that implements this function.

Hints:

1. You do not actually need to read any of the rows of the CSV file to obtain the fieldnames.
2. If you are having trouble implementing this function, we suggest that you review this piece of documentation for the csv module.

Problem 2: Reading a CSV file into a list of dictionaries

Next, you will write a function called `read_csv_as_list_dict` that takes the name of a CSV file and returns the data within the file as a list of dictionaries. Each item in the list corresponds to a row in the CSV file. The dictionaries within the list map the field names to the column values for that row. As CSV files can use different separator characters and quote characters, this function takes those characters as input and uses them to properly parse the CSV file.

Here is the signature of the `read_csv_as_list_dict` function:

```
def read_csv_as_list_dict(filename, separator, quote):
    """
    Inputs:
      filename  - name of CSV file
      separator - character that separates fields
      quote     - character used to optionally quote fields
    Output:
      Returns a list of dictionaries where each item in the list
      corresponds to a row in the CSV file.  The dictionaries in the
      list map the field names to the field values for that row.
    """
```

You need to write the code that implements this function.

Problem 3: Reading a CSV file into a dictionary of dictionaries

Next, you will write a function called `read_csv_as_nested_dict` that takes the name of a CSV file and returns the data within the file as a dictionary of dictionaries. Each key-value pair in the outer dictionary corresponds to a row in the CSV file. The keys in that dictionary are the values of a header column in the table. The function takes the name of that header column as the input `keyfield`. If a key appears multiple times in column corresponding to `keyfield`, the last row containing the key is used to create the dictionary used as the corresponding value.The inner dictionaries within the outer dictionary map the field names to the column values for that row. As CSV files can use different separator characters and quote characters, this function takes those characters as input and uses them to properly parse the CSV file. Note that the key-value pair for `keyfield` should be in the inner dictionaries, even though its value is used as the key in the outer dictionary.

Here is the signature of the `read_csv_as_nested_dict` function:

```
def read_csv_as_nested_dict(filename, keyfield, separator, quote):
    """
    Inputs:
      filename  - name of CSV file
      keyfield  - field to use as key for rows
      separator - character that separates fields
      quote     - character used to optionally quote fields
    Output:
      Returns a dictionary of dictionaries where the outer dictionary
      maps the value in the key_field to the corresponding row in the
      CSV file.  The inner dictionaries map the field names to the
      field values for that row.
    """
```

You need to write the code that implements this function.

Problem 4: Writing a list of dictionaries to a CSV file

Finally, you will write a function called `write_csv_from_list_dict`. This function takes a table structured as a list of dictionaries (as if read by `read_csv_as_list_dict`) and writes it to the file named by the `filename` input. The function also takes a list of field names, `fieldnames`, as input in order to make sure the fields appear in the appropriate order (as specified by the order of the list `fieldnames`) in the CSV file. The function also takes a separator character and a quote character that should be used when writing the file. All non-numeric fields should be quoted using the specified quote character.

Here is the signature of the `write_csv_from_list_dict` function:

```
def write_csv_from_list_dict(filename, table, fieldnames, separator, quote):
    """
    Inputs:
      filename   - name of CSV file
      table      - list of dictionaries containing the table to write
      fieldnames - list of strings corresponding to the field names in order
      separator  - character that separates fields
      quote      - character used to optionally quote fields
    Output:
      Writes the table to a CSV file with the name filename, using the
      given fieldnames.  The CSV file should use the given separator and
      quote characters.  All non-numeric fields will be quoted.
    """
```

You need to write the code that implements this function.

Hints:

1. Do not forget to open the CSV file for writing.
2. Be sure to review the csv module documentation to learn how to write a CSV file.
3. Many of the options that you can use for reading CSV files are also valid when writing CSV files.