CS 2316 - **Homework 8 – Data Merge**

Due: Wednesday, April 6th, 2016, before 11:55 PM

Files to submit:

1. **HW8.py**

This is an INDIVIDUAL assignment!

Collaboration at a reasonable level will not result in substantially similar code. Students may only collaborate with fellow students currently taking CS 2316, the TA's and the lecturer. Collaboration means talking through problems, assisting with debugging, explaining a concept, etc. You should not exchange code or write code for others.

For help:

- TA Helpdesk—schedule posted on class website
- Email TAs or use Piazza

Notes:

- **Don't forget to include the required comments and collaboration statement (as outlined on the course syllabus).**
- **Do not wait until the last minute to do this assignmen**t in case you run into problems

Introduction

The Atlanta Zoo is in a bit of a pickle—it needs to verify data for its animals, but there seems to be a few gaps.

You have been hired by the zoo to write a Python script that web scrapes data from the internet and compares it to a CSV the Atlanta Zoo provides to you with all of the information they currently have. The zoo is quite large and it has many different animals, however it hasn't been very organized lately, so some of the data is missing. It's up to you to help the Atlanta Zoo merge and organize all the data on its animals. Each animal is given a unique name and placed into one of the Zoo's exhibits and has an associated yearly cost.

The CSV file contains data in the following form:

Animal's Last Name, Animal's First Name, Yearly Cost

Keep in mind that there is a header so the animal data starts in the second row.

The website contains data in the following form:

Animal's Full Name, Species, Exhibit

NOTE: The website contains FULL names while the CSV file contains two separate columns—one for the last name and one for the first name.

Things to plan for:

1. The Zoo has the following Exhibits in the data:
    a. African Plains
    b. Aquatic
    c. Arctic
    d. Australian Plains
    e. Birds
    f. Insects
    g. Petting Zoo
    h. Rain Forest
    i. Reptiles

2. Some animals that appear in the CSV file do not appear in the website, and vice versa. All animals that appear in either source should be included in the output CSV file. If an animal does not appear in both data sources, some of their data fields will be empty, and you should place a "-" in the final output CSV file to indicate this missing data.

3. Recently, the Australian Plains Exhibit and the African Plains department have been merged into The Wonders of the Plains. Any animal whose exhibit is "Australian Plains" or "African Plains" should have their exhibit changed to "The Wonders of the Plains" in the output CSV file.

Objective

You will have to read in the data from the provided CSV file and also web scrape data off the HTML table on the given website. You must analyze both pieces of data and return a CSV file that looks like the following:

Name	***Yearly Cost***	***Species***	***Exhibit***
Last Name, First Name	*Yearly Cost*	*Species*	*Exhibit*

***Notice that the headers above are in bold. You must have the same headers in your output CSV file (but your headers will not be "bolded" in the CSV file) the "Name" column is a single data item, but the names inside it will have commas in them.**

In order to make the Atlanta Zoo's data more organized, you must write out this CSV data in a particular order.

1. You must first sort the animals **by exhibit**. (The exhibit that comes first in the alphabet should be at the top of the list.)
2. Then the animals should be sorted alphabetically by **Species.**
3. Next, you must then sort the animals by name **alphabetically within each exhibit**. Alphabetize by last name, and if two or more animals have the same last name, sort those by their first names.

Note: If an animal is missing any data, remember to include a hyphen "-" to denote that there is no data for that category. Animals without an exhibit will be sorted into the "hyphen" exhibit group.

Smaller Test Files

To test your code, you want to check your solution against a small sample file first to make sure everything seems to working correctly. If and only if you get things working smoothly on the smaller test file should you move on to the larger test files. You can find the small test file and website linked from the course homework webpage.

Larger Test Files

Once you get your code working properly for the small test files, you should check your code with the large test files to ensure that your code works with all test cases. Keep in mind that running large test files may take longer than the small test files. Your program must be able to complete the large test file in less than five (5) minutes running on a TA's laptop.

Web Scraper

You will need a function or method that will retrieve the table of data at the URLs found in the class webpage and convert it into a list of data. You may use regular expressions, or write your own data extractor using string operations.

Informational GUI

Create a simple GUI that will allow the user to choose which CSV file they want to read in. Once the file is selected, your code should download data from the website (the URL will be defined by the user in an entry). The left side of the GUI should contain two buttons, one to read in the input CSV file and another to read data from the website and process the output data. There should be entry boxes for the input CSV file name, the output CSV file name, and the website's URL.

After processing is complete, the right side of the GUI should display labels with the name of each exhibit and the number of animals in each exhibit, as well as a total count of all animals (including animals that are not in an exhibit). The labels should initially display a hyphen "-" in place of a number until both the CSV and website data have been read in. (See the suggested problem solution below for an image of what the GUI should look like. Before reading in any data, the "Input CSV File" entry box should be populated with the complete path of the input CSV file, the "Website URL" entry box will contain the website URL that is specified by the user as input, and the "Output CSV File" entry box should be populated with the output CSV file's complete path. Notice that **the input and output file path entry boxes are in readonly state**. Also note that the "Process Data" button is "grayed out" or **disabled** UNTIL the user successfully loads a CSV file! Only after the user successfully loads a CSV file should they be able to click the "Process Data" button.

Suggested Problem Solution

The following is the suggested way to solve this homework assignment. If you'd like you can use different functions and/or add helper functions.

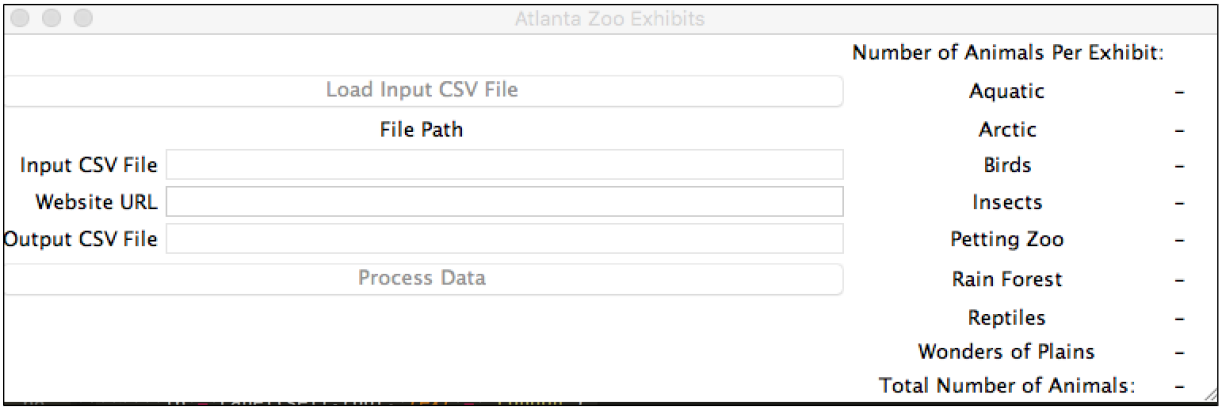
Note that if you use a class for your GUI, each function should have the self parameter.

__init__

This function will set up the GUI. The GUI should have 3 labeled entries (Input CSV File, Output CSV File, and Website URL). Only the Website URL entry should allow user input; the other two should be "readonly". The GUI should also have two buttons (Load Input CSV File and Process Data). The Load Input CSV File button should be enabled by default, but the Process Data button should be disabled (grayed out) until the user loads CSV file data.

The right side of the GUI should have labels for each of the Exhibits (Australian and African Plains does not need a label because it will be merged with Wonders of Plains) and a total count of animals. The labels will be updated when the data is processed.

Before reading in data, the GUI should look something like this:


Atlanta Zoo Exhibits
Load Input CSV File
File Path
Input CSV File
Website URL
Output CSV File
Process Data
Number of Animals Per Exhibit:
Aquatic -
Arctic -
Birds -
Insects -
Petting Zoo -
Rain Forest -
Reptiles -
Wonders of Plains -
Total Number of Animals: -

loadCSVclicked

Input: None

Output: None

This function should be called when the user clicks the “Load Input CSV File” button. Prompt the user to select a file using a file dialog (askopenfilename). Then call “loadCSVFile” giving it the name of the CSV file. If the loadCSVFile method returns a list of data, store it in an object variable of your choice. Place the name of the loaded CSV file into the correct entry in the GUI so the user can see it. Finally, enable the “Process Data” button so the user can now click it!

If instead the loadCSVfile method returned a None (indicating that the file was invalid), you should **pop up an error dialog message to the user**, and leave the input CSV file entry blank and the Process Data button disabled.

loadCSVfile

Input: A string representing a filename

Output: CSV Data as a list, or None (if the file is invalid).

This method will open the specified file and load the CSV data from it. If you are successful, you should return a list of lists, where each inner list is one row from the CSV file. If the file does not exist, or is not a valid CSV file, you should return None. The method that calls **loadCSVFile** (**loadCSVClicked**) will issue an appropriate warning to the user.

PDclicked

Input: None

Output: None

This function should be called when the user clicks the “Process Data” button. If the Process Data button is enabled, it means that the user has already successfully

loaded the contents of a CSV file into the object variable of your choice. This function should check the URL entry in the GUI and get the URL that the user has (hopefully) typed into the entry. It should take this URL and pass it as an argument to the downloadData function. If the download fails (for example, if the URL entry is blank), you should show a warning message (e.g. “URL or Data Invalid!” and return. Because buttons that call command methods do not actually use the return value, what you return is immaterial. The point of returning on an unsuccessful download is to prevent the PDclicked method from continuing to execute more code. The best return value in this case would be a None.

If, however, the download was successful, the PDclicked method should then call the **convertHTMLtoCSVFormat** function, using it to convert the data. Once the data is in the right format, pass it to the **mergeData** function. Finally, PDClicked will call **saveData** to save the output data to a CSV file. PDClicked will also call **calculate** to count the number of employees in each department and calculate the total number of employees (these numbers should all be displayed in labels in the GUI). Again, because the button which “called” PDclicked doesn't care about the return value, you may return None.

downloadData

Input: String representing the URL

Output: A list of data from the website, or None if the download/extract failed.

Given the URL, this method will attempt to download the HTML data from the specified website. You may use regular expressions, or string.find(), to extract the data. Your method should return the data (see convertHTMLtoCSVFormat for an example of the format the data should be in). If you run into a problem (website/url does not exist, etc.) return None instead.

convertHTMLtoCSVFormat

Input: Downloaded HTML data list Output:

A list of converted HTML data

This helper function will take in the HTML data (a list of lists in the following format: full name, Species, Exhibit) and return it in the CSV data format (last name, first name, department, salary). You may find the use of regular expressions extremely helpful, although not required.

For example, the downloaded HTML data will look like the following:

[[‘Animal Name’, Species", ‘Exhibit’], [‘Foxy Williams’, 'Fox', 'African Plains'],
…]
You must convert this data into the following form:

[[‘Last Name’, ‘First Name’, 'Species', ‘Exhibit,], [‘Williams', 'Foxy', ‘Fox’, ‘African Plains’,], …]

After converting the downloaded HTML data into the aforementioned format, you should return that list.

mergeData

Input: Converted HTML data list and CSV data list or none (if both are self variables)

Output: A dictionary of merged data

This method will take each animal record from the input CSV file and put it into a dictionary. Then, it will take each record from the HTML data and add it to the dictionary. Keep in mind that there may be some animals that only exist in either the CSV file or the HTML data, but not both. So when adding the HTML data, you may need to update an existing record, or you may need to add an entirely new record.

Also remember to follow the guidelines described above in the “things to plan for” section with respect to the “Plains” exhibits and missing yearly cost data.

saveData

Input: The dictionary returned from **mergeData**

Output: None

This method will use an asksaveasfilename file dialog to ask the user for a filename to save the data to. It should update the GUI by adding the file name to the correct entry, and then write out the updated records as a CSV file. Keep in mind that the output CSV file has a format different from that of the input CSV file and HTML data as specified above. The data in the CSV file needs to be should be sorted in the format mentioned previously in this document.

calculate

Input: The dictionary returned from **mergeData**

Output: None

This method will count the number of animals in each exhibit in the merged data. It will also add up the total number of animals in the zoo. The numbers of animals per exhibit and the total number of animals will be displayed in labels on the GUI when this method is called.

Grading

You will earn points as follows for each element that works correctly according to the specifications.

Basic functionality: 60 points

Code is easily readable and understandable with good comments	10
Successfully loads the input CSV file	5
Successfully downloads the webpage HTML	5
Correctly parses the HTML page to extract records	10
Correctly converts the HTML data to CSV format	10
Process Data button is disabled until CSV Data is successfully loaded.	10
Input and output file names is displayed correctly in the GUI	10

Matching Accuracy: 40 points

Dictionary of merged data is completely correct	15
Output CSV file is completely correct	15
Animals per Exhibit and total Animals are correct	10