CS 2316 - **Homework 8 – Data Merge**

Due: Friday, March 28th, 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 TA's 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 city of Shamalamadingdong is in a bit of a pickle—it needs to verify salary data, but there seems to be a few gaps.

You have been hired by the city of Shamalamadingdong to write a program that downloads salary data from the internet and compare it to a CSV the city of Shamalamadingdong provides to you with all of the information they currently have.

The CSV file contains data in the following form:

Last Name, First Name, Salary for 2000, Salary for 2012

Keep in mind that there is a header so peoples' salary data starts in the second row.

The website contains data in the following form:

Full Name, Salary for 2000, Salary for 2012

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. CSV Data
    a. The CSV data has two columns for salaries: one column for the 2000 year and one column for the 2012 year.
    b. The CSV may not have data for a certain person for a specific year (i.e. John Doe might have salary data for 2000, but not for 2012 or vis versa).

If this is the case, there will be a "-"(hyphen or dash) in place of the salary.

2. HTML Data
    a. The HTML data has two columns for salaries: one column for the 2000 year and one column for the 2012 year.
    b. The HTML data may not have data for a certain person for a specific person (i.e. John Doe might have salary data for 2000, but not for 2012 or vis versa).

If this is the case, there will be a "-" (hyphen or dash) in place of the salary.

3. Unfortunately, the city of Shamalamadingdong is not very organized. Some people that appear in the CSV file do not appear in the HTML data and vis versa.

Because the data for this homework is "fake" and auto-generated, the salary data for one person in a specific year will not be different in the CSV and HTML file. (If this were real data we couldn't make things this easy for you!) For example, if John Doe has 13000 as his salary in year 2000 in the CSV file, the HTML data will have either 13000 or "-" for John Doe in year 2000.

Objective

You will have to read in the data from the provided CSV file and also webscrape 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	***2000***	***2012***
Last Name, First Name	*Salary for 2000*	*Salary for 2012*

***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).**

Because there are some people that only have salary data for one year, you must write out this CSV data in a particular order.

1. You must first write out all of the people who have salary data for both years in alphabetical order; alphabetize by last name and if two or more people have the same last name, sort those people by their first names.

2. You must then write out all of the people who have salary data for year 2000, but not 2012, in alphabetical order; alphabetize by last name and if two or more people have the same last name, sort those people by their first names.
3. Finally, you must write out all of the people who do not have salary data for year 2000, but do have data for year 2012 in alphabetical order; alphabetize by last name and if two or more people have the same last name, sort those people by their first names.

Note: If a person does not have salary data for a specific year, remember to include a hyphen “-” to denote that there is no data for that year.

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 file. 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 ten (10) minutes running on a TA's laptop.

Web Table 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, write your own data extractor, or make use of the HTML table parser demonstrated in class and posted on the class notes webpage:

http://www.summet.com/dmsi/html/readingTheWeb.html#parsing-html-tables

If you use this HTML table parser, you must indicate the source of the function with comments in your code. If you modify the function but still use the foundation of it, be sure to comment on which parts you modified and why.

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 as an input). The GUI should consist of two buttons and three entry boxes: one entry for the CSV file name, one entry for the output CSV file name, and one entry for the website’s URL. It should look similar to the following:

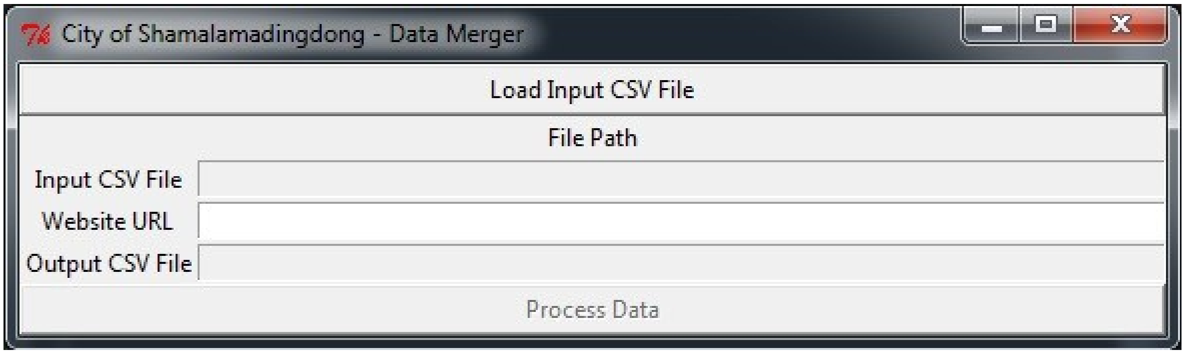
The “Input CSV File” entry box should be populated with the complete path of the input CSV file, the “Website URL” entry box will contain the website URL that is specified by the user as input, and the “Output CSV File” entry box should be populated with the output CSV file’s complete path. Notice that **the input and output file path entry boxes** 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.

__init__

This function will set up the GUI . The gui should have 3 labeled entries (Input CSV Filename, Output CSV Filename, 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 CSV File and Process Data). The Load 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.

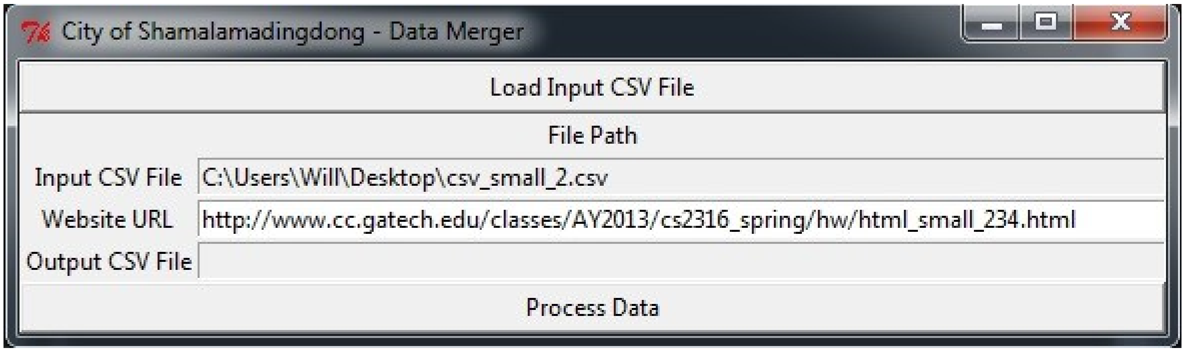


loadCSVclicked

Input: None

Output: None

This function should be called when the user clicks the “Load 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!



Input CSV File C:\Users\Will\Desktop\csv_small_2.csv
Website URL http://www.cc.gatech.edu/classes/AY2013/cs2316_spring/hw/html_small_234.html
Output CSV File

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 file name

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 downloadSalaryData 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 which 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 ***saveSalaryData*** to save the output data to a CSV file. Again, because the button which “called” PDclicked doesn't care about the return value, you may return None.

downloadSalaryData

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, string.find() or the parse table method from the course notes 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, salary for 2000, salary for 2012) and return it in the CSV data format (last name, first name, salary for 2000, salary for 2012). You may find the use of regular expressions extremely helpful, although not required.

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

[[‘Name’, ‘2000’, ‘2012’], [‘Chris Hemsworth’, ‘139000’, ‘-‘], [‘Tony Stark’, ‘10000’, ‘500000’], …]

You must convert this data into the following form:

[[‘Last Name’, ‘First Name’, ‘2000’, ‘2012’], [‘Hemsworth’, ‘Chris’, ‘139000’, ‘-‘], [‘Stark’, ‘Tony’, ‘10000’, ‘500000’], …]

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 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 records 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. Be sure to NEVER replease valid salary data (e.g. 13000) with a dash or hyphen!

For example, if the entry in the CSV and converted HTML data looked like the following:

CSV

[[‘Hemsworth’, ‘Chris’, ‘139000’, ‘-‘], [‘Stark’, ‘Tony’, ‘10000’, ‘500000’],

[‘America’, ‘Captain’, ‘-‘, ‘5000’] …]

Converted HTML

[[‘Hemsworth’, ‘Chris’, ‘-‘, ‘7000’], [‘Stark’, ‘Tony’, ‘-’, ‘500000’],

[‘America’, ‘Captain’, ‘-‘, ‘5000’] …]

Your final dictionary would have entries that look like the following:

{(‘Hemsworth’, ‘Chris’): (‘139000’,’7000’), (‘Stark’, ‘Tony’: (‘10000’, ‘500000’),

(‘America’, ‘Captain’):(‘-‘, ‘5000’)}

Notice that for Chris Hemsworth, he is missing salary data for 2012 in the CSV and salary data for 2000 in the converted HTML data, but in the dictionary entry, he has values for both. In other words, the hyphen “-“ is just replaced if there is salary data in the other data list. You may use helper functions that this method calls if you wish.

saveSalaryData

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.

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 table 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**
Information about the input and output file names is displayed correctly in the GUI	**10**

Matching Accuracy: 40 points

Dictionary of merged data is completely correct	**20**
Output CSV file is completely correct	**20**