

CS 2316 - Homework 8 - Data Merge

Due: Friday, October 24th, 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 assignment** in case you run into problems.
-

Introduction

The city of Shamalamadingdong is in a bit of a pickle—it needs to verify data for its employees, but there seems to be a few gaps.

You have been hired by the city of Shamalamadingdong to write a program that downloads employee data from the internet and compares it to a CSV the city of Shamalamadingdong provides to you with all of the information they currently have. The city is quite large and its government has many employees, however it hasn't been very organized lately, so the some data is missing. It's up to you to help Shamalamadingdong merge and organize all the data on its employees.

The CSV file contains data in the following form:

Last Name, First Name, Department, Salary

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

The website contains data in the following form:

Full Name, Department, Salary

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 government of Shamalamadingdong has the following departments in the data:
 - a. Transportation
 - b. Human Resources
 - c. Environment
 - d. Education
 - e. Public Works
 - f. Parks and Recreation
2. The city of Shamalamadingdong is a little disorganized, so for some employees, salary data is only available in the CSV or only available on the website. If this happens, include whatever information is available. Fortunately there won't be a case where the salary for one employee is different between the CSV and the website.
3. If neither the website nor the CSV file has salary data for an employee, the final CSV should have a "-" in place of the salary. All employees will have department data.
4. Some employees that appear in the CSV file do not appear in the website, and vice versa. All employees that appear in either source should be included in the output CSV file.
5. Recently, the Parks and Recreation department and the Environment department have been merged. Any employee whose department is "Parks and Recreation" should have their department changed to "Environment" in the output CSV file.

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:

<i>Name</i>	<i>Salary</i>	<i>Department</i>
<i>Last Name, First Name</i>	<i>Salary</i>	<i>Name of Department</i>

***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).**

In order to make the Shamalamadingdong employee data more organized, you must write out this CSV data in a particular order.

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

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

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 be 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 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 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 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 department and the number of employees in each department, as well as a total count of all employees. 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.

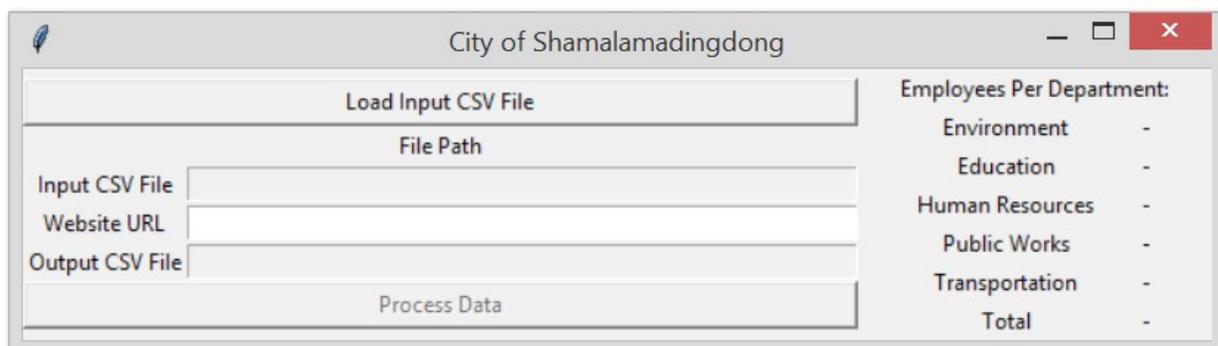
Note that if you use a class for your GUI, each function should have the self parameter.

__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 departments (Parks and Recreation does not need a label because it will be merged with Environment) and a total count of employees. The labels will be updated when the data is processed. It is highly recommended that you use the **grid** layout manager!

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

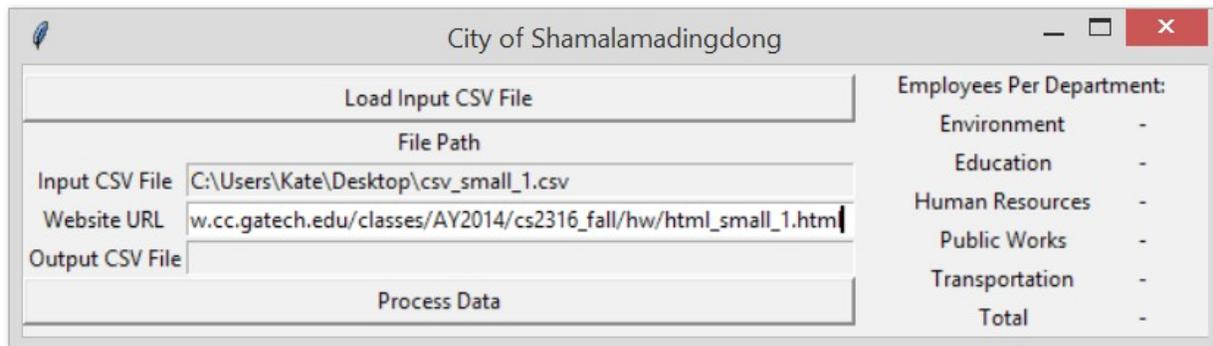


loadCSVclicked

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`.

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, department) 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:

```
[ ['Name', 'Salary', 'Department'], ['Leslie Knope', '139000', 'Parks and Recreation'], ['Bob Loblaw', '50000', 'Human Resources'], ... ]
```

You must convert this data into the following form:

[['Last Name', 'First Name', 'Department', 'Salary'], ['Knope', 'Leslie', 'Parks and Recreation', '139000'], ['Loblaw', 'Bob', 'Human Resources', '50000'], ...]

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 employee 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 employees 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.

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 employees in each department in the merged data. It will also add up the total number of employees in the government of Shamalamadingdong. The numbers of employees per department and the total number of employees 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 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
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
Employees per department and total employees are correct	10