Homework 7 - File I/O & CSV

CS 1301 - Intro to Computing - Fall 2020

Important

- Due Date: **Tuesday, October 20th, 11:59 PM**.
- This is an individual assignment. High-level collaboration is encouraged, **but your submission must be uniquely yours.**
- Resources:
  - TA Helpdesk
  - Email TA's or use class Piazza
  - How to Think Like a Computer Scientist
  - CS 1301 YouTube Channel
- Comment out or delete all function calls. Only import statements, global variables, and comments are okay to be outside of your functions.
- **Read the entire document before starting this assignment.**

The goal of this homework is for you to enhance your understanding of File I/O and practice reading and writing to files. The homework will consist of 6 functions for you to implement. You have been given `HW07.py` skeleton file to fill out. However, below you will find more detailed information to complete your assignment. Read it thoroughly before you begin.

Hidden Test Cases: In an effort to encourage debugging and writing robust code, we will be including hidden test cases on Gradescope for some functions. You will not be able to see the input or output to these cases. Below is an example output from a failed hidden test case:

```
Test failed: False is not true
```

Written by Will Hunnicutt (whunnicutt3@gatech.edu) & Josh Tabb (jtabb6@gatech.edu)

Part 1: File I/O

For the Part 1 File I/O functions, the `.txt` file being read from will be formatted as seen below. Be sure to download the provided file to the same folder as your `HW07.py` file. To open the `.txt` file, you can use the built-in Notepad for Windows, TextEdit for Mac, or any other text editor of your choice.

You will be working with a text file that contains restaurant data. The data will contain a restaunt's name, cuisine type, and whether it's fast food or sit-down. Each restaurant's data will be separated by a newline. Below shows how the text file will be formatted. See the provided `restaurants.txt` file as an example.

```
Restaurant #1 Name
Restaurant #1 Cuisine Type
Restaurant #1 Fast Food or Sit-down

Restaurant #2 Name
Restaurant #2 Cuisine Type
Restaurant #2 Fast Food or Sit-down

.
.
.
```

Find Cuisine

Function Name: findCuisine()
Parameters: filename (`str`), cuisine (`str`)
Returns: list of restaurants (`list`)
Description: You want to go eat with some friends and are in the mood for a specific type of food. To make it easier to decide, you want to get a list of all the restaurants that match the type of food you're in the mood for. Given a filename to a file that contains restaurant data, return a list of restaurants that have the same cuisine type as the desired cuisine. If no restaurants match the given cuisine or no restaurants are in the file, return an empty list.

```
>>> findCuisine('restaurants.txt', 'American')
['Cookout', "McDonald's", "Wendy's", 'Chickfila']
```

```
>>> findCuisine('restaurants.txt', 'NotACuisine')
[]
```

Restaurant Filter

Function Name: restaurantFilter()
Parameters: filename (`str`)
Returns: dictionary that maps cuisine type (`str`) to a list of restaurants of the same cuisine type (`list`)
Description: You want to easily be able to see restaurants with the same cuisine type, so you decide to create a dictionary. This dictionary will allow fast and easy look-up of restaurants for the next time you want a certain type of food. Given a filename to a file that contains restaurant data, return a dictionary that maps each cuisine type to a list of restaurant names that have that cuisine type. If no restaurants are in the file, return an empty dictionary.

```
>>> restaurantFilter('restaurants.txt')
{
    'Mexican': ['Taco Bell', 'La Parilla'],
    'American': ['Cookout', "McDonald's", "Wendy's", 'Chickfila'],
    'Italian': ['Olive Garden', 'Twisted Kitchen'],
    'Greek': ['Gyro Bros'],
    'Chinese': ['Panda Express'],
    'Japanese': ['Momonoki']
}
```

Create Directory

Function Name: createDirectory()
Parameters: filename (`str`), output filename (`str`)
Returns: None (`NoneType`)
Description: A local mall wants you to create a food directory file for all the restaurants in the mall. The file should split the restaurants into **Fast Food** and **Sit-down** groups and list them in **alphabetical order** in their respective groups. The restaurant lists should be numbered, and each restaurant should have its cuisine type listed as well (separated by a hyphen). Given a filename to a file that contains the mall's restaurant data, write a file (with the given output filename) that has the following format:

```
Restaurant Directory

Fast Food
1. Fast Food Restaurant #1 - Cuisine Type
...

Sit-down
1. Sit-down Restaurant #1 - Cuisine Type
...
```

```
>>> createDirectory('restaurants.txt', 'directory.txt')
```

The resulting file (`directory.txt`) should **NOT** have an extra '\n' at the end:

```
Restaurant Directory

Fast Food
1. Chickfila - American
2. Cookout - American
3. Gyro Bros - Greek
4. McDonald's - American
5. Panda Express - Chinese
6. Taco Bell - Mexican
7. Wendy's - American

Sit-down
1. La Parilla - Mexican
2. Momonoki - Japanese
3. Olive Garden - Italian
4. Twisted Kitchen - Italian
```

Part 2: CSV

For the Part 2 CSV functions, the `.csv` file being read from will be formatted as seen below. As stated previously, be sure to download the provided file into the same folder as your `HW07.py` file. By default, your computer will likely use Excel on Windows, or Sheets on Mac, to open the `.csv` file, but don't be alarmed: reading values from a `.csv` file is no different than a `.txt` file; the only difference lies in the formatting of data.

You will be working with COVID-19 data (keep in mind that these are not actual COVID statistics). The `.csv` file will contain a country name, total population, and infected population, separated by commas (hence the name **Comma Separated Values** file). Each country's data will be separated by a newline. Below shows how the `.csv` file will be formatted. See the provided `covid.csv` file as an example. Note that the first row contains the titles for each of the columns, and does not contain any actual data.

```
Country,TotalPopulation,TotalInfected
Country1,Population,InfectedPopulation
Country2,Population,InfectedPopulation
Country3,Population,InfectedPopulation
.
.
.
```

Infected Percentage

Function Name: infectedPercentage()
Parameters: country list(`list`), filename(`str`)
Returns: country and percentage (`tuple`)
Description: You've just been contacted by the World Health Organization concerning the COVID pandemic. Given a list of countries, create a function that returns a tuple of the country with the greatest percentage of infected individuals, along with this percentage. Remember, we're looking for the country that has the greatest percentage of infected persons relative to the total population, **NOT** the greatest number of infected people. The final percentage should be rounded to 2 decimal places. If given an empty list, return **None**.

```
>>> infectedPercentage(["Belgium", "Belarus", "Bermuda"], 'covid.csv')
("Bermuda", 94.24)
```

```
>>> infectedPercentage(["Sweden", "Turkey", "Ukraine"], 'covid.csv')
("Turkey", 75.58)
```

COVID-19 Country Status

Function Name: countryStatus()
Parameters: country list (`list`), filename(`str`)
Returns: risk dictionary (`dict`)
Description: After helping create the previous function, the World Health Organization has assigned another task: identifying which countries need the most help and outside resources. When given a list of countries, your function should a return a dictionary, where the keys are different levels of risk of a country, and the values are a list of countries with that level of risk. We will define 3 different risk levels: "Low risk", "Medium risk", and "High risk". A country should be labeled as "Low risk" if the infected number is less than or equal to 25% of the total population, "Medium risk" if the infected number is greater than 25% of the toal population and less than or equal to 65% of the total population, and "High risk" if the infected number is greater than 65% of the total population.

Note: The countries in your list should appear in the same order as they do in the csv file.

Note: The test cases for this problem are on the following page

```
>>> countryList = ["United States", "Tonga", "Poland", "New Zealand", "Norway"]
>>> countryStatus(countryList, 'covid.csv')
{
    'Low risk': [],
    'Medium risk': [],
    'High risk': ['Norway', 'New Zealand', 'Poland', 'Tonga', 'United States']
}
```

```
>>> countryList = ["Belgium", "Bangladesh", "Belarus", "Bermuda"]
>>> countryStatus(countryList, 'covid.csv')
{
    'Low risk': ['Bangladesh', 'Belarus'],
    'Medium risk': ['Belgium'],
    'High risk': ['Bermuda']
}
```

Comparative Risk

Function Name: compareRisk()
Parameters: country to compare (`str`), country list (`list`), filename(`str`)
Returns: compared countries (`list`)
Description: The World Health Organization has been very impressed by your work so far, and has decided to give you one more mission. Your mission, should you choose to accept it, is to create a function that takes in a base country to compare to, along with a list of other countries. This function should return a list of countries from the list passed in with a smaller population than the base country, but with an infected population that is greater than the country we are comparing to. If there are no countries that fit these requirements, return **"No countries"**.

Note: The countries in your list should appear in the same order as they do in the csv file.

```
>>> countryList = ["Belgium", "Bangladesh", "Belarus", "Bermuda"]
>>> compareRisk("Tunisia", countryList, 'covid.csv')
"No countries"
```

```
>>> countryList = ["Turkmenistan", "Norway", "Netherlands", "Philippines"]
>>> compareRisk("Tuvalu", countryList, 'covid.csv')
["Netherlands", "Norway", "Turkmenistan"]
```

Grading Rubric

Function	Points
findCuisine()	15
restaurantFilter()	15
createDirectory()	20
infectedPercentage()	15
countryStatus()	15
compareRisk()	20
Total	**100**

Provided

The `HW07.py` skeleton file has been provided to you. This is the file you will edit and implement. All instructions for what the functions should do are in this skeleton and this document. You have also been provided `restaurants.txt` and `covid.csv` as examples for the data input you will be receiving.

Submission Process

For this homework, we will be using Gradescope for submissions and automatic grading. When you submit your `HW07.py` file to the appropriate assignment on Gradescope, the autograder will run automatically. The grade you see on Gradescope will be the grade you get, unless your grading TA sees signs of you trying to defeat the system in your code. You can re-submit this assignment an unlimited number of times until the deadline; just click the "Resubmit" button at the lower right-hand corner of Gradescope. You do not need to submit your `HW07.py` on Canvas.