Homework 7 - File I/O

CS 1301 - Intro to Computing - Summer 2022

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 5 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 Cynthia Medlej (cmedlej3@gatech.edu) & Alice Heranval (aheranval3@gatech.edu)

Part 1: My International CookBook

For Part 1 , the `cookBook.txt` file being read from will be formatted as shown below. Be sure to download the provided file from Canvas, and move the 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 data about recipes. The data will contain a recipe name, cooking time, and country of origin. Each Recipes's data will be separated by a newline. The code below shows how the text file will be formatted. See the provided `cookBook.txt` file as an example.

```
Recipes

Recipe #1 Name
Recipe #1 PrepTime
Recipe #1 Country

Recipe #2 Name
Recipe #2 PrepTime
Recipe #2 Country

.
.
.
```

Lazy Meal

Function Name: lazyMeal()
Parameters: maxPrepTime (`int`)
Returns: recipe (`list`)
Description: You're so overwhelemed this week and you've spent way too much money on take-out. It's time for you to cook for yourself! Write a function which will use the `cookBook.txt` file (with the format shown above), and take in the maximum time you can spend cooking as a parameter. This function should return a list (sorted in **alphabetical order**) of all the names of all the recipes from the `.txt` file which you can make within that time limit (inclusive). If there are no recipes that you can make within the time limit, return the string `"Waffle House it is!"`.

```
>>> lazyMeal(25)
["Hummus","Pizza","Shakshuka"]
```

```
>>> lazyMeal(5)
"Waffle House it is!"
```

International Guests

Function Name: groupCountries()
Parameters: countries(`list`)
Returns: recipeDict (`dict`)
Description: Your international friends are coming over and you want to satisfy their cravings, but you keep forgetting who likes what. Write a function which uses the `cookBook.txt` file to return a dictionary with names of your friends as keys of this dictionary, mapping to a sorted list of recipes from their home countries. This function will take in a list of tuples containing both the name and country of each friend in a tuple. Use the country name given in the coutries list to find all the recipies from the `.txt` file.

Note: If you can't find at least one national recipe for each of your international friends, return the string `"Throw the book or get new friends!"`

```
>>> groupCountries([("Marco", "Italy"),("Isabelle", "Lebanon")])
{"Marco": ["Pasta","Pizza"], "Isabelle": ["Hummus"]}
```

```
>>> groupCountries([("Sarra", "Tunisia")])
{"Sarra":["Shakshuka"]}
```

Favorite Recipes

Function Name: favRecipes()
Parameters: recipeList (`list`)
Returns: None (`NoneType`)
Description: Inspired by reading through the `cookBook.txt` file, you decide that you want to write your own `.txt` file, called `favRecipes.txt`. Write a function that takes in a list of recipes (recipeList) and uses the information provided by the `cookBook.txt` file to write to a new file called `favRecipes.txt`. The file `favRecipes.txt` should be formatted in the exact same way as `cookBook.txt`, but it should only include information about recipes that are contained in recipeList. If there are no recipes or none of the recipes are in the file, `favRecipes.txt` should contain the string `"No information found."`.

Note: Be sure that no extra newline (`\n`) character is included at the end of your `favRecipes.txt` file. The order of movies in `favRecipes.txt` should match the order they appear in the `cookBook.txt`. Also, be sure that no extra newline (`\n`) character is included at the end of your `candyOrder.txt` file.

```
>>> favRecipes(["Sushi", "Mofongo"])
```

Output of `favRecipes.txt`:

```
Recipes

Sushi
70
Japan

Mofongo
40
Puerto Rico
```

```
>>> favRecipes(["Shiro"])
```

Output of `favRecipes.txt`:

```
No information found.
```

Part 2: Halloween Party

For Part 2 the `.txt` files being read from will be formatted as shown below. As stated previously, be sure to download the provided files from Canvas, and move them into the same folder as your `HW07.py` file.

You will be working with data from a file containing your friends' preferences, and data from a file containing a list of your favorite Halloween movies. The `friends.txt` file will contain your friend's name, a rating from 1-10 of how scary they want their Halloween movies to be, and their votes for top 2 favorite candies for the party. The `movies.txt` file will contain the movie name and how scary it is (on a scale of 1-10). Each data will be separated by a newline. Below shows how the `.txt` files will be formatted. See the provided `friends.txt` and `movies.txt` files as an example. **You can assume that the files will never be empty** (and will always be in the right format).

`friends.txt`:

```
Friend #1 Name
Friend #1 MaxRating
Friend #1 FavCandy1
Friend #1 FavCandy2

Friend #2 Name
Friend #2 MaxRating
Friend #2 FavCandy1
Friend #2 FavCandy2

.
.
.
```

`movies.txt`:

```
Movie #1 Name
Movie #1 Rating

Movie #2 Name
Movie #2 Rating

.
.
.
```

Order Candy

Function Name: orderCandy()
Parameters: prices (`dict`), maxCost (`float`)
Returns: None (`NoneType`)
Description: You want to buy candy for your Halloween party, but you want to make sure you don't buy too much of the wrong kind. Write a function that uses the `friends.txt` file to create a new file called `candyOrder.txt` that contains the candies that cost less than the maxCost and is wanted by 3 or more friends. This function should take in a dictionary mapping candy names to their prices, in order to determine which candies are within the budget. If their prices in the dictionary are less than or equal to the maxCost, write them to the `candyOrder.txt` file in the format `candyName: numVotes`, where numVotes refers to how many times the candy was written as a favorite in `friends.txt`. Each candy should be written on a new line.

Note: You can assume that all candies in the file will be in the dictionary! If there are no candies that fit the criteria to be written to the file, write in a string that says `"Ask again."`. Also, be sure that no extra newline (`\n`) character is included at the end of your candyOrder.txt file. The order of candies in `candyOrder.txt` should match the order they appear in the friends file.

```
>>> prices = {'KitKats': 2.0, 'M&Ms': 3.0, 'Hersheys': 3.5, 'Reeses': 1.5,
              'Starburst': 2.0, 'Twix': 3.2, 'Jolly Ranchers': 2.6}
>>> orderCandy(prices, 3.0)
```

Output of `candyOrder.txt`:

```
KitKats: 3
M&Ms: 3
Reeses: 4
```

```
>>> prices = {'KitKats': 2.0, 'M&Ms': 3.0, 'Hersheys': 3.5, 'Reeses': 1.5,
              'Starburst': 2.0, 'Twix': 3.2, 'Jolly Ranchers': 2.6}
>>> orderCandy(prices, 1.0)
```

Output of `candyOrder.txt`:

```
Ask again.
```

Movie Picker

Function Name: moviePicker()
Parameters: potentialMovie (`str`)
Returns: isOption(`bool`)
Description: Finally, you need to choose which movie to see during the Halloween party! You were given a recommendation from a friend, but you want to make sure it's good enough for everyone invited. Write a function that uses `.txt` files: the `friends.txt` file and the `movies.txt` file, and takes in the movie recommendation (`potentialMovie`) to figure out which scary movies you can watch with your friends. Use the data from the two files to make sure that the potentialMovie does not have a higher rating than the lowest horror movie rating in `friends.txt`, so that no one is too scared during the movie night! Return `True` if the movie is good to watch (i.e. has been given a rating in the file less than or equal to the lowest movie rating among your friends) and `False` if another should be chosen instead.

```
>>> moviePicker("Midsommar")
False
```

```
>>> moviePicker("The Nightmare Before Christmas")
True
```

Grading Rubric

Function	Points
lazyMeal()	20
groupCountries()	20
favRecipes()	20
orderCandy()	20
moviePicker()	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.

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.