Homework 8 - CSV Files/APIs

CS 1301 - Intro to Computing - Fall 2023

Important

- Due Date: **Tuesday, October 31st, 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 to extend your knowledge of File I/O to be able to work with important and frequently used data exchange formats like CSV and JSON. We'll be pulling data from the web with APIs, analyzing and writing data from and to CSV files, and converting one data exchange format to the other! This homework will require you to implement 5 functions. You have been given the HW08.py skeleton file to aid you with your responses.

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 Ramya Subramaniam (ramyasub@gatech.edu) & Muhammad Hamzah Zahid (mzahid30@gatech.edu) & Muhammad Shayan Waqar (mwaqar9@gatech.edu) & Jane Crowley (jcrowley38@gatech.edu)

PART 1 - CSV Files

For Part 1, the `.csv` file being read from will be formatted as shown below. Be sure to download the provided file from Canvas, and **move it into the same folder as your `HW08.py` file**. Recall that if you are using a Mac, do not place HW08.py and starWars.csv in your Downloads or Desktop folder.

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 the data.

The `starWars.csv` file (and other `.csv` files like it) will contain a character's name, their Midi-Chlorians, their type (or species), and the number of the first movie they appeared in.

A character's Midi-Chlorian score directly corresponds to the character's ability to wield the Force in the Star Wars universe. While Midi-Chlorians are normally positive, we have made them negative for you if the character is a Sith (evil). Thus, a powerful Jedi (good character) has a large positive Midi-Chlorian score (like 95), and a powerful Sith (evil character) has a large negative Midi-Chlorian score (like -90). Each data entry will be separated by a newline. Below is an example of how the `.csv` file will be formatted. Note that the first row contains the headers for each of the columns; it does not contain any actual data.

Format of `starWars.csv`:

```
name,midi-chlorians,type,firstAppearance
name1,midi-chlorians1,type1,firstAppearance1
name2,midi-chlorians2,type2,firstAppearance2
...
```

New Recruits

Function Name: newRecruits()
Parameters: movieNum (`int`)
Returns: winner (`str`)
Description: Given a `movieNum` that corresponds to the potential characters' first appearance, this function should figure out whether the Jedi (good) or Sith (evil) characters for that specific movie number would win if they were to fight (all the characters on their corresponding side will fight together). Jedi and Sith characters are determined using their Midi-Chlorians in the `starWars.csv` file. Remember, a negative Midi-Chlorian score corresponds to the Sith side, while positive Midi-Chlorians correspond to the Jedi side.

After totaling the Midi-Chlorians of both the sides, if the Jedis will triumph, return `"The Jedis have won by {numPoints} points!'` , where numPoints represents the amount of Midi-Chlorians the good recruits have won by. If the Siths win, return `"Oh no, the Siths are better!"` . If there's a movie with no new appearances by the characters, return `"No new recruits!"`

Note: There won't be any ties.

```
>>> newRecruits(1)
"Oh no, the Siths are better!"
```

```
>>> newRecruits(4)
"The Jedis have won by 350 points!"
```

The Best of Each

Function Name: orgByType()
Parameters: minChlorian (`float`), maxChlorian (`float`)
Returns: organizedDict (`dict`)
Description: After watching Star Wars for the first time, you are still unsure what characters belong to the Jedi side, Sith side, or somewhere in-between. Give a minimum and maximum Midi-Chlorian score, create a dictionary mapping the character type to a list of characters of that type, if their Midi-Chlorian score falls within the range passed in, inclusive. The lists in the dictionary should be sorted alphabetically by the character name. If there are no characters of any type within the range, return an empty dictionary.

```
>>> orgByType(0, 80)
{'Protocol Droid': ['C-3PO'], 'Wookiee': ['Chewbacca'],
 'Human': ['Boba Fett', 'Lando Calrissian', 'Rey']}
```

```
>>> orgByType(-75, -10)
{'Human': ['Anakin Skywalker / Darth Vader'], 'Assassin Droid': ['IG-88']}
```

PART 2 - APIs

For Part 2, we will be using the Starwars API. Please read through the documentation, as it will be extremely helpful for completing this assignment.

If you make a valid request with the URL: https://swapi.dev/people/1, you will receive the following JSON formatted response:

```
{
        "name": "Luke Skywalker",
        "height": "172",
        "mass": "77",
        "hair_color": "blond",
        "skin_color": "fair",
        "eye_color": "blue",
        "birth_year": "19BBY",
        "gender": "male",
        "homeworld": "https://swapi.dev/api/planets/1/",
        ...
}
```

If you make a request with an invalid URL, you will receive the following response:

```
{"detail": "Not found"}
```

While the official endpoints of the Starwars API are numbers that correspond to the planets, people, starships, etc., you can search for a specific planet, person, or starship by **querying**.

API Query parameters are optional key-value pairs that appear after a question mark in a URL. They are extensions of the URL that are used to retrieve specific content or perform specific actions. All endpoints in the Starwars API support a search parameter that filters the set of resources returned. This allows you to make requests like:

```
https://swapi.dev/api/planets/?search=Tatooine
```

to retrieve planet information of Tatooine.

You will have to use this form of querying to retrieve specific information for all of the remaining questions.

Space Break

Function Name: possiblePlanets()
Parameters: favPlanets (`list`)
Returns: destinationDict (`dict`)
Description: Thanksgiving break is almost here! You have decided to go with your friends to explore one of your favorite planets. However, you all agreed to only visit a planet with a temperate climate. You can find a planet's temperature information in the Star Wars API, explained above. Given a list of names of your favorite planets, write a function that returns a dictionary that maps the planets in the passed in list to their climate, but only if "temperate" is one of the given climates of that planet. If none of your favorite planets have the following criteria, then return an empty dictionary.

Note: Invalid planets should not be included in `destinationDict`. Consider using a `try/except` block to deal with this.
Hint: Planets may have more than one climate.

```
>>> favPlanets = ["Tatooine", "Alderaan", "Yavin IV", "Hoth", "Bespin"]
>>> possiblePlanets(favPlanets)
{'Alderaan': 'temperate', 'Yavin IV': 'temperate, tropical', 'Bespin': 'temperate'}
```

```
>>> favPlanets = ["Endor", "Naboo", "Uranus", "Neptune", "Coruscant"]
>>> possiblePlanets(favPlanets)
{'Endor': 'temperate', 'Naboo': 'temperate', 'Coruscant': 'temperate'}
```

Resident Heights

Function Name: residentHeights()
Parameters: planet (`str`)
Returns: heights (`list`)
Description: Sightseeing is on your bucket list this Thanksgiving, and in order to experience unobstructed views, you want to travel to planets with shorter people. Write a function that takes in the name of a planet and returns the names of all residents of that planet and their respective heights, formatted as a list of tuples. The first element of each tuple should be the height of the resident from the passed-in country, and the second element should be their name. Sort the residents in your list in ascending order **based on their heights**. If the planet is invalid, return an empty list.

Note: Beru Lars' last name appears lowercase in the API, so don't be alarmed that it is lowercase in the test case or when you run your code.

```
>>> residentHeights("Tatooine")
[(97, 'R5-D4'), (163, 'Shmi Skywalker'), (165, 'Beru Whitesun lars'),
 (167, 'C-3PO'), (172, 'Luke Skywalker'), (178, 'Owen Lars'),
 (183, 'Biggs Darklighter'), (183, 'Cliegg Lars'), (188 'Anakin Skywalker'),
 (202, 'Darth Vader')]
```

```
>>> residentHeights("Alderaan")
[(150, 'Leia Organa'), (188, 'Raymus Antilles'), (191, 'Bail Prestor Organa')]
```

Starship Information

Function Name: starshipInfo()
Parameters: starships (`list`)
Returns: None (`NoneType`)
Description: CSV files are easier to work with than APIs, right? You'll need data of important starships before you travel the universe, so write a function that takes in a list containing the names of some starships. Your function should create a CSV file called `starships.csv` that follows the format below:

```
Starship Name,Model,Starship Class,Passengers
starshipName1,model1,starshipClass1,passengers1
starshipName2,model2,starshipClass2,passengers2
...
```

If the starship is invalid, don't add it to your file. You should always create the CSV file even if it only contains the header. Ensure there is no extra `"\n"` at the end of the file.

```
>>> starshipInfo(["Millennium Falcon", "Rebel transport", "Star Destroyer"])
```

Output: `starships.csv`

```
Starship Name,Model,Starship Class,Passengers
Millennium Falcon,YT-1300 light freighter,Light freighter,6
Rebel transport,GR-75 medium transport,Medium transport,90
Star Destroyer,Imperial I-class Star Destroyer,Star Destroyer,n/a
```

```
>>> starshipInfo(["Imperial shuttle", "Calamari Cruiser", "Republic Cruiser"])
```

Output: `starships.csv`

```
Starship Name,Model,Starship Class,Passengers
Imperial shuttle,Lambda-class T-4a shuttle,Armed government transport,20
Calamari Cruiser,MC80 Liberty type Star Cruiser,Star Cruiser,1200
Republic Cruiser,Consular-class cruiser,Space cruiser,16
```

Grading Rubric

Function	Points
newRecruits()	20
orgByType()	20
possiblePlanets()	20
residentHeights()	20
starshipInfo()	20
Total	**100**

Provided

The `HW08.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 `HW08.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 `HW08.py` on Canvas.