Homework 7 - File I/O & CSV

CS 1301 - Intro to Computing - Spring 2021

Important

- Due Date: **Tuesday, October 26th, 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 Carolyn Yuan (cyuan78@gatech.edu) & Assata Quinichett (aquinichett@gatech.edu)

Part 1: File I/O

For Part 1 (File I/O functions), the `.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 music. The data will contain a song's name, its genre, and the artist. Each song's data will be separated by a newline. The code below shows how the text file will be formatted. See the provided `music.txt` file as an example.

```
Music data

Song #1 Name
Song #1 Genre
Song #1 Artist

Song #2 Name
Song #2 Genre
Song #2 Artist

.
.
.
```

Sort By Artist

Function Name: sortByArtist()
Parameters: filename (`str`), artist (`str`)
Returns: list of songs (`list`)
Description: You want to make a playlist for Fall Break. However, there are way too many songs to choose from, so you've decided to choose what songs to add based on your cousin's favorite artist. Given a filename to a file that contains music data, return a list of songs that match your cousin's favorite artist. If no songs match your cousin's favorite artist, return an empty list.

Note: If there is more than one song by the artist, the order of songs in the list should appear as they do in the music file.

```
>>> sortByArtist('music.txt', 'Shawn Mendes')
['Stitches', 'Wonder', 'Senorita', "There's Nothing Holdin' Me Back"]
```

```
>>> sortByArtist('music.txt', 'Justin Bieber')
[]
```

Genre Filter

Function Name: genreFilter()
Parameters: filename (`str`)
Returns: mapping of songs to lists of genres of that song (`dict`)
Description: You want to be able to easily see songs with the same genre, so you decide to create a dictionary. Given a filename to a file that contains music data, return a dictionary that maps each genre to a list of songs that belong to that genre.

```
>>> genreFilter('music.txt')
{
 'Pop': ['Stitches', 'When We Were Young', 'Someone Like You', 'Wonder',
         'Senorita', "There's Nothing Holdin' Me Back"],
 'Rock': ['Bohemian Rhapsody', 'Stairway to Heaven', 'We Will Rock You'],
 'Jazz': ['Giant Steps'],
 'Rap': ["Can't Hold Us", 'Say So', 'Old Town Road']
}
```

Sort By Genre

Function Name: sortByGenre()
Parameters: filename (`str`), genre (`str`), output filename (`str`)
Returns: None (`NoneType`)
Description: You've been given a data book of songs, their genres, and the artist who wrote the songs. To make things easier to find in the data book, you decide to filter the songs according to their genres. You've also decided to sort the song names alphabetically to make them even easier to find. Given a file name to a file that contains all the music data and a genre, write all the sorted data for the specified genre to a file (with the given output filename) in the following format:

```
Genre

1. Song - Artist
2. Song - Artist
.
.
.
```

```
>>> sortByGenre('music.txt', 'Rock', 'musicByGenres.txt')
```

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

```
Rock

1. Bohemian Rhapsody - Queen
2. Stairway To Heaven - Led Zeppelin
3. We Will Rock You - Queen
```

Note: If the genre does not exist in the music file, your output file should be in the following format:

```
Genre
```

Part 2: CSV

For Part 2 (CSV functions), the `.csv` file being read from will be formatted as shown below. As stated previously, be sure to download the provided file from Canvas, and move it into the same folder as your `HW07.py` file. By default, you 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.

You will be working with music artist data. The `.csv` file will contain an artist name, the number of albums they've created, the number of albums they've sold (in millions), and the number of albums they've had in the Bill Board Top 200, separated by commas (hence the name **Comma Separated Values** file). Each data will be separated by a newline. Below shows how the `.csv` file will be formatted. See the provided `artists.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.

```
artist,totalAlbumsCreated,totalAlbumsSold,totalBillBoard200Albums
artist1,albumsCreated1,albumsSold1,billBoard200albums1
artist2,albumsCreated2,albumsSold2,billBoard200albums2
artist3,albumsCreated3,albumsSold3,billBoard200albums3
.
.
.
```

Biggest Success

Function Name: biggestSuccess()
Parameters: filename (`str`), artists (`list`)
Returns: artist and percentage (`tuple`)
Description: Given a list of artists, create a function that returns a tuple of the artist with the greatest **ratio of BillBoard200Albums to AlbumsCreated**, along with this percentage. Remember, we're looking for the artist with the greatest percentage of Albums that made the Bill Board Top 200, NOT the greatest number of Bill Board Top 200 albums. The final percentage should be rounded to 2 decimal places. If given an empty list or the greatest percentage is 0, return an empty tuple.

Note: You can use try/except to catch `ZeroDivisionErrors` that might occur.

```
>>> biggestSuccess('artists.csv', ['Taylor Swift', 'KAYTRANADA', 'Blink-182',
                                   'Willow','Tame Impala', 'The Marías'])
('Taylor Swift', 81.82)
```

```
>>> biggestSuccess('artists.csv', ['Connan Mockasin', 'Yellow Days',
                                   'Arlo Parks'])
()
```

Grammy Awards

Function Name: grammyAwards()
Parameters: filename (`str`), artists (`list`)
Returns: category of artists by celebrity status (`dict`)
Description: You're hiring artists to perform at the Grammy Awards and need to identify which artists are the most successful. When given a list of artists, your function should return a dictionary, where the keys are different levels of celebrity status, and the values are the artists with that level of celebrity status. We will define 3 different levels of celebrity status: `"A-List"` , `"B-List"` , and `"C-List"` .

A `"C-List"` celebrity has a Bill Board Top 200 Albums number that is less than or equal to 30% of their Albums Created. A `"B-List"` celebrity has a Bill Board Top 200 Albums number that is greater than 30% and less than or equal to 70% of their Albums created. An `"A-List"` celebrity has a BillBoard Top 200 Album number that is greater than 70% of their albums created.

Note: You can use try/except to catch `ZeroDivisionErrors` that might occur.

```
>>> grammyAwards('artists.csv', ['Taylor Swift', 'KAYTRANADA', 'Blink-182',
                                 'Willow','Tame Impala', 'The Marías'])
{
  'A-List': ['Taylor Swift'],
  'B-List': ['Tame Impala'],
  'C-List': ['The Marías', 'KAYTRANADA', 'Blink-182', 'Willow']
}
```

```
>>> grammyAwards('artists.csv', ['Connan Mockasin', 'Yellow Days',
                                 'Arlo Parks'])
{
  'A-List': [],
  'B-List': [],
  'C-List': ['Connan Mockasin', 'Yellow Days', 'Arlo Parks']
}
```

Grading Rubric

Function	Points
sortByArtist()	20
genreFilter()	20
sortByGenre()	20
biggestSuccess()	20
grammyAwards()	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 "Re-submit" button at the lower right-hand corner of Gradescope. You do not need to submit your `HW07.py` on Canvas.