Homework #5 - Debugging/Test Cases

In this homework, you are provided with a dataset of movies in CSV format. You have also been provided with code for reading the CSV and performing functions on the data in it. The code provided fails several test cases and needs to be debugged. Your goal is to debug all of the provided methods such that they work correctly, and pass all the given test cases. You should not adjust the test cases in any way. Here are descriptions of all of the classes and functions:

FileReader
Represents a generic file reader. Used to read in data from a file of the user's choice, analyze, and manipulate its data as Python objects.

FileReader.__init__(self, filename)
The constructor. Creates a new *FileReader* object based on the specified filename. For our purposes, the file should be in the same folder as HW5.py. To open a file called "mydiary.txt", you would call *FileReader('mydiary.txt').*

FileReader.open_file(self)
Opens the file in read-only mode. Stores the resulting object as an instance variable called *file_obj*.

FileReader.read_lines(self)
Reads the lines from the file object into a list where each row of the CSV is a separate element and stores it as an instance variable called *file_data*. Then closes the file object.

FileReader.strip_trailing_newlines(self)
Removes unnecessary newline characters (\n) from the end of each string in *file_data*.

CsvReader
A child class of FileReader for reading CSV files only.

CsvReader.__init__(self, csvfile)
The constructor. Overwrites the *FileReader* constructor such that in order to read a file called "mycsv.csv", one only needs to call CsvReader('mycsv'). (Since CsvReader is designed to work only with .csv files, we don't need to specify an extension.)

CsvReader.build_data_dict(self)
Accesses the data stored in the *file_data* instance variable (a list) and converts it to a dictionary, *data_dict*, where each key is a column name found in the CSV, and each value is a list of data in that column with the same order as it was in the file. Each element of the list corresponds to a single row in the CSV. For example, to access the "Name" column in the CSV, I would access *data_dict*['Name'].

CsvReader.get_name_rating(self)
From the data stored in the data_dict instance variable, returns a list of tuples containing the name of the movie and its rating (as a floating-point number) in the format (Name, Rating). The list should be sorted based on the descending order of ratings.
For example, [('Interstellar', 8.6), ('The Terminator', 8), ('Cars', 7.1)]

CsvReader.get_genre_counts(self)
From the data stored in the data_dict instance variable, returns a list of tuples in the format ('Comedy', 10) where the first element is the genre, and the second element is the number of movies with that genre. The list should be sorted in descending order by the number of movies.
For example, [('Comedy', 20), ('Drama', 15), ('Action', 12), ('Romance', 8)]

CsvReader.most_common_month(self)
Iterates through the "Release Date" column and counts the number of movies that were released in the same month (regardless of year). Finally, returns the month with the maximum number of movie releases. For example, if the most movie releases were in July, this method would return 7.
(We are using the American date format of month/date/year.)

CsvReader.get_movie_ID(self)
Returns a list of IDs for each row in the 'URL' column of the CSV by extracting the ID from the end of the URL. (i.e. the ID that would be extracted from the URL"https://www.imdb.com/title/tt1825683" would be "tt1825683") For example, given: ['https://www.imdb.com/title/tt1825683', 'https://www.imdb.com/title/tt8946378', 'https://www.imdb.com/title/tt1375666'], this method would return ['tt1825683', 'tt8946378', 'tt1375666'].

Part I: Debug the Code

Modify the code so that all of the provided test cases pass. You must do this **WITHOUT** modifying the test cases themselves or modifying the data. We will be running software that checks for modifications to the test cases or the data. You also may not hardcode answers (returning expected values directly) into any of the functions you are modifying. Violating any of the above requirements will result in an automatic 0 for the affected functions.

Part II: Add 2 Test Cases

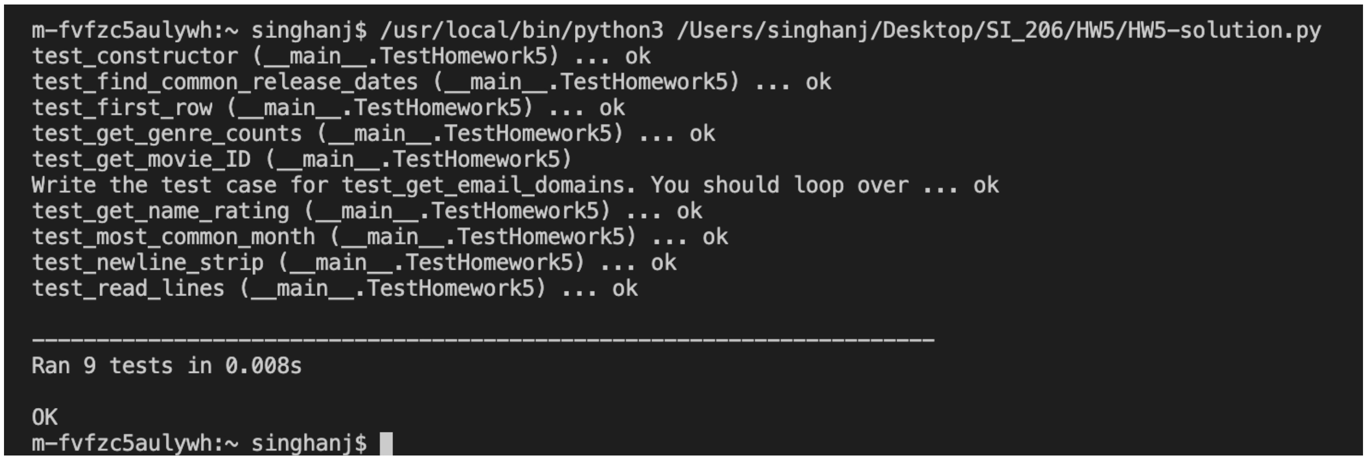
Write the test case for *get_movie_ID*. You should loop over the data returned by *CsvReader.test_get_movie_ID( )* and write two assert statements within that loop. The first assert statement should test if each ID starts with 'tt'. The second should check whether the length of each ID is 9.

We will check that your output for *CsvReader.get_movie_ID()* passes a hidden test case that meets the above two requirements. You will receive points for passing our version of this test case, and correctly implementing (and passing) your own version of this test case.

Grading Rubric (60 points):

5 points for passing *test_constructor*
5 points for passing *test_read_lines*
10 points for passing *test_get_name_rating*
10 points for passing *test_get_genre_counts*
10 points for passing *test_most_common_month*
15 points for correctly implementing and passing *test_get_movie_ID*
5 points for passing our hidden version of *test_get_movie_ID*

Sample Output:



```
m-fvfzc5aulywh:~ singhanj$ /usr/local/bin/python3 /Users/singhanj/Desktop/SI_206/HW5/HW5-solution.py
test_constructor (__main__.TestHomework5) ... ok
test_find_common_release_dates (__main__.TestHomework5) ... ok
test_first_row (__main__.TestHomework5) ... ok
test_get_genre_counts (__main__.TestHomework5) ... ok
test_get_movie_ID (__main__.TestHomework5)
Write the test case for test_get_email_domains. You should loop over ... ok
test_get_name_rating (__main__.TestHomework5) ... ok
test_most_common_month (__main__.TestHomework5) ... ok
test_newline_strip (__main__.TestHomework5) ... ok
test_read_lines (__main__.TestHomework5) ... ok

----------------------------------------------------------------------
Ran 9 tests in 0.008s

OK
m-fvfzc5aulywh:~ singhanj$
```

Extra Credit:

Part 1 (3 points)

Implement a method called *find_common_release_dates* as part of the CsvReader class. It should iterate through all of the rows of the CSV and return a list of tuples in the format (‘5/25’, 4). The first element of each tuple should be a common release date, i.e. a release date (including both day and month, regardless of year) shared by more than 1 movie. The second element should indicate the number of movies that share that release date. Finally, you should sort in decreasing order of the number of movies.
Example output: [('10/5', 5), ('11/4', 3), ('5/23', 2)]

Part 2 (3 points)

Create a non-trivial test method, *test_find_common_release_dates*, with at least two assert statements for the *find_common_release_dates* method. One assert statement should test that only release dates shared by more than one movie are included. The other assert statement should test that the tuples are sorted in descending order.