CS 1026B – Assignment 4

Due: April 5, 2022 – 6:00 PM

Overview

Write a program using objects to simulate a movie reviewing system.

Reminders

- **IMPORTANT NOTE: an SRA for A4 is only valid until 6:00 PM on April 8 (the last day of classes)!**
- Your code must be done individually.
- Your code will be graded in part by an automated system.
- Your code may be compared to other submissions using computer software.
- You can submit your code up to 48 hours late, with a deduction of 0.5% per hour (or part of an hour) that the assignment is late.

Background

You are writing code to manage a database for movie reviews. Movies are added to the system, and reviews are added for that movie. Then summaries of all reviews for movies can be displayed.

Program

Your program is designed to simulate a system for managing reviews for movies. This will be called the movie database. The movie database is controlled by reading a text file that contains the commands for the database. Reading these commands changes the state of the movie database – adding movies, adding reviews or displaying information.

Your program will create classes for both movies and movie databases. Movies will have a title, a year of release and a collection of reviews. The movie database will contain a list of movies. Then your program will prompt the user for a file containing a list of commands, read the list of commands, and process the information in each command. This will produce some output, which should be shown on the screen.

Classes

You will write two classes for your program. Note that these classes must each be in their own files, called movie.py and moviedb.py.

1. A **Movie** class – this class represents the data associated with a movie. You should store the following information as instance variables in your class:
    a. The movie **title**
    b. The **year** of release of the movie
    c. The reviews of the movie. Movies are reviewed on a 1-5 star scale. The number of each type of review (1-star, …, 5-star) needs to be saved so that these values can be displayed. You should use a data structure to store these reviews.

   All of these instance variables should be stored as private variables (names should start with two underscores).

The Movie class will also need these methods[1]:

a. A constructor (`__init__`) to allow Movie objects to be created. This should take two parameters (title and year). The constructor should create a movie with no reviews.
b. **addReview**, which accepts a single integer parameter, representing that a new review (with the number of stars indicated by the parameter) should be added for this movie. The program should assume that the input is an integer. If the integer is not between 1 and 5 (inclusive), then the review should be ignored and not added.
c. A short representation of the reviews for a movie, called **shortReview**. This method should take no parameters. This method should **return** (as a string) the year and the average reviews as shown in the following:

```
Dune (2021): 4.7/5
```

This represents a movie whose title is "Dune", whose year of release is 2021, and has an average rating of 4.7/5[2]. There is a single space before the open parenthesis and a single space after the colon. The average review is rounded to one decimal place.
d. A long representation of the reviews for the movie, called **longReview**. This method takes no parameters. This method should **return** (as a string) a more detailed review of the movie.

```
Dune (2021)
Average review: 4.7/5
*****: 2
**** : 1
***  : 0
**   : 0
*    : 0
```

Note that the colons must line up, and there is no space between the 5 stars and the colon in the third line of the review. You do not need to worry about aligning the numbers after the column – put a single space after the colon, and then the number of reviews of that type. The average review is rounded to one decimal place.
e. **Accessors** (but not mutators) for the title (called **getTitle**) and the year (called **getYear**).
f. A **private** method called **__calcAverage** that calculates the average review of a movie. The average returned by the method should be rounded to one decimal place. Note that all numbers (including integer averages) should be rounded to one decimal place, so that an average of 4 should be displayed as 4.0/5.

2. A **Movie Database** class – this class represents all the data for reviews for several movies. You will need one instance variable in this class: A data structure (list or set) of Movie objects in the database. Additionally, you should construct the following methods in the Movie Database class:
   a. A constructor (`__init__`) that creates an empty database with no movies. The constructor should take no parameters.
   b. **addMovie:** This method takes two parameters, the title and year of a movie. This method should create a new Movie object and add it to the database if it does not exist

[1] in all the descriptions of methods in this assignment, the self parameter is not explicitly mentioned, but is always present.
[2] In this assignment, you can assume that any movies with **no** reviews has an average of 0.0/5.

already. If it does exist, the method should raise a `KeyError` with an appropriate error message indicating the title of the duplicate.

c. **findMovie**: This method takes two parameters, the title and year of a movie. This method should return the **Movie object** in the database with the same name and year. If the movie does not exist in the database, the method should return None.

d. **showAll:** This method should print all short reviews for all movies in the database. The movies should be sorted by year and title. To do this, you can use the following technique, similar to how soundex encodings were sorted in Assignment 2:

```
info = []
for m in [YOUR_LIST_OF_MOVIES]:
    info.append( (m.getTitle(), m.getYear()))

info.sort()

for (t,y) in info:
     print ( [MOVIE WITH TITLE t AND YEAR y] )
```

This method does not return any value.

Input File and Commands

To run, your code should open a file and read commands (see the section called Provided Code below). There are four types of commands in the input files:

1. NEW: create a new movie for the database, if it does not already exist. If it exists, the new command should have no effect. The command has the form:

   ```
   NEW-Movie Title-YEAR
   ```

   The statement should produce no output. HINT: you may find the python command "pass" (which has no effect) helpful to handle the case of duplicate NEW commands.

2. REV: add a new review to the database, if the movie exists. If the movie does not exist in the database at the time this command is read, the command has no effect. If the number of stars in the review is not in the range 1-5, the command has no effect The command has the form:

   ```
   REV-Movie Title-YEAR-STARS
   ```

   The final part of the command is an integer representing the number of stars. The statement should produce no output.

3. SHO: show the **short** version of the reviews for **all** movies in the database. The movies should be shown in alphabetical order of their title, and by oldest-to-newest for movies with the same title. The format of the command is just SHO, with no additional information.

4. PRI: print the **long** version of the reviews for **one** movie in the database, if it exists. If the movie does not exist, the command has no effect. The command has the form.

   ```
   PRI-Movie Title-YEAR
   ```

In all the command, movie titles are sequences of letters, punctuation, numbers and spaces but do not contain the dash character (-). Years are sequences of digits 0-9.

Provided Code

You will be provided with code to read a file of commands. This code (in a file called provided_code.py) will open a file of commands and read them.

Functions

Beyond the above classes, you are required to break your program into functions. This includes these functions:

1. A function for reading commands, called readFile (largely based on the provided_code.py file).
2. A main function. The main function should prompt for the name of the file, and then call the code to execute the commands in that file. The prompt must read
   `"Enter the name of the file: "` (including the space.)

You should not use global values in your functions: all data needed by a function should be communicated with parameters and return values. Except for main, all functions can be given any suitable name. These functions must be in a **separate file from the Movie and Movie Database classes**.

Code Requirements

Your code must satisfy the following requirements in addition to producing correct output.

1. Your code must be documented appropriately. Do not document every line of code, but major portions of your code must be commented.
2. Your code must include a comment at the top of program (**the file that includes main()**) that includes your name and describes the overall function of the program. The files with the classes do not need to have this comment.
3. Your code should use appropriate structures, such as loops, lists and functions. Functions must be used based on the requirements above. You must also use a main() function.
4. You should give a docstring for each function and method you write (except for main and `__init__` for classes). See the programming standards for more details.
5. You should review the programming standards document for information on commenting, variable naming, documentation of functions, and other issues.

Submission Details

- You must submit your code to gradescope using the instructions on owl. As you need to submit multiple files, submit three separate .py files – do **not** submit them as a zip file.
- Your Movie class must be in a file called movie.py
- Your Movie Database class must be in a file called moviedb.py
- The file that contains your main must be called Assign4.py.
- You must follow the format of the output shown in the example below, including the prompt.

Assignment Marking

- Your code will be marked by an automated tool. The testing program assumes that
  - The code is saved as a file called Assign4.py

    - You are using a python3 version (recommended python3.9 or higher). If you use python2, you will likely lose marks.

  If your code has to be marked manually due to failure to follow these instructions, you will receive a deduction from your final mark.
- Your code will be evaluated by the Teaching Assistants for the code requirements described above.

Sample Output

Consider the following input file:

```
NEW-Dune-2021
REV-Dune-2021-5
REV-Dune-2021-4
NEW-Spiderman: No Way Home-2021
NEW-The Batman-2022
REV-Spiderman: No Way Home-2021-5
REV-Dune-2021-5
REV-Dune-2021-4
REV-Dune-2021-5
REV-Spiderman: No Way Home-2021-5
REV-Spiderman: No Way Home-2021-3
REV-Spiderman: No Way Home-2021-5
PRI-Dune-2021
SHO
```

As you can see, three movies are created in the database, but not all at the beginning of the file. Reviews are also added for two of the three movies. In particular, 5 reviews are added for the movie Dune (5,4,5,4,5, in that order). None of the commands except PRI and SHO produce any output.

- When PRI is processed, the long review for Dune is shown.
- When SHO is processed, the short reviews for each movie are shown, in the order described above.

The output for this file would then be:

```
Enter the name of the file: movies.txt
Dune (2021)
Average review: 4.6/5
*****: 3
**** : 2
***  : 0
**   : 0
*    : 0
Dune (2021): 4.6/5
Spiderman: No Way Home (2021): 4.5/5
The Batman (2022): 0.0/5
```

Note that in the case when a movie has no reviews, the result should be 0.0/5 and not 0/5. This is also the case if the reviews for a movie are averaged to an integer value, like 4. It should be displayed as 4.0/5, not 4/5.

Gradescope Test Cases

There are 16 gradescope tests. They are available on owl to review, along with the reason for each test case. If you are failing a test, you should review the test first to identify the problem that is being tested, to help debug your code. You can debug with the test case or a similar case that tests a similar input.