Spring, 2015 CIT 590

Programming Languages and Techniques

Homework 4

This homework deals with the following topics

* dictionaries, sets and lists

* databases using dictionaries (not too far from how they really work!)

* test driven development (TDD)

General problem specification

Websites like IMDB (Internet Movie Data Base) maintain all the information about movies, directors, actors, actresses etc. I'll use the word actor in these specifications to mean actor or actress. They also do maintain a review score of the movie, but to make this assignment more interesting we will use rotten tomatoes for our source of movie rankings.

We will use 2 dictionaries. The first corresponds to information about an actor and all the movies that they have acted in. Hence this will be keyed on the actor. The second corresponds to information about the critics score and the audience score from rottentomatoes about the movies.

Given that information we then want to answer some typical movie trivia questions.

Starter code

We have provided data about the movies in the form of two files. One called movies.txt and the other called moviescores.csv. Download both these files from the HW4 folder and put them in the same folder as the Python program you are going to write.

Call your Python file **movie_trivia.py**

Since we have not seen files yet (we will next week possibly), we have provided some starter code in a file with that same name that can be found in the HW4 folder.

As noted in the comments in that file. Please remove the print statements that we have before you submit your file.

In the interest of keeping all this fun, please do add some movies of your own to both files. They do not even have to be English movies. Just names of movies and names of actors.

Utility functions

The first step is to create some utility functions for handling the database. Do all of this as closely as possible in the TDD paradigm i.e. test first.

You are definitely allowed to modify the movies.txt and moviescores.csv file. Feel free to add more movies, actors, ratings etc. As you can see, we insist that the dictionaries get passed to the individual functions (an alternative would be to use global variables which we decided not to do in this assignment).

So it does not have to be the same dictionary as the one we initially create. In fact, you can be guaranteed that the unit tests we write for checking your code will use different data (it will be similarly formatted though).

Here is the list of basic functions that we want you to write

1. **def insert_actor_info(actor, movies, movie_Db)** - where movie_Db is the dictionary to be inserted into/updated. actor is a string. movies is a list of movies that the actor has acted in.

   Note that while this function is called insert, it should actually do an insert/update. If the actor is already present, it should go and append these new movies to their list.

   This function does not return anything. It will work by just modifying the movie_Db passed to it.

2. **def insert_rating(movie, ratings, ratings_Db)** - where ratings_Db that is to be inserted into/updated. movie is the movie name as a string. ratings is a **tuple** like (critics rating, audience rating).

   It also does not print anything. It works by modifying the ratings_Db that is passed to it. It is fine to print while you are debugging but do not submit extra print statements in your final version.

3. **def delete_movie(movie, movie_Db, ratings_Db)** - delete all information from the database that corresponds to this movie. movie is a string. movie_Db is the actor dictionary. ratings_Db is the ratings dictionary. Remember there is movie information in both of these dictionaries.

4. **def select_where_actor_is(actorName, movie_Db)** - given an actor, return the list of all movies. This one is rather trivial, but I want you to write it anyway. For those curious about the naming of this function, it comes from syntax you see when you are actually using a database.

5. **def select_where_movie_is(movieName, movie_Db)** - given a movie, return the list of all actors

6. **def select_where_rating_is(targeted_rating, comparison, is_critic, ratings_Db)** - This useful function returns a list of movies that satisfy an inequality or equality based

on the comparison argument and the targeted rating argument. The comparison argument is either =, > or < and is passed in as a string. The is_critic argument is a boolean that represents whether we are interested in the critics rating or the audience rating.

The targeted_rating is an integer.

Some examples of expected output are

select_where_rating_is(0, '>', True, ratings_Db) should return every single movie.

select_where_rating_is(65, '=', False, ratings_Db) should return every movie that has a rating of 65 as per the audience.

select_where_rating_is(30, '<', True, ratings_Db) should return every movie that has a rating less than 30. Basically the ones the critics hated!

User Questions

Outside of these utility functions we would like to be able to answer the following questions when interacting with a user

- Given an actor's name, find all the actors with whom he/she has acted.

  Define a function called **def get_co_actors(actorName, moviedb)** - returns list of all actors that the actor has ever worked with in any movie.

- Find out the movies in which both actors are present.

  For this call the function **def get_common_movie(actor1, actor2, moviedb)** - goes through the database and returns the movies where both actors were cast. In cases where the two actors have never worked together, it returns just the empty list.

- Who is the most critically acclaimed actor.

  def critics_darling(movie_Db, ratings_Db) - given the two dictionaries, we are interested in finding the actor whose movies have the highest average rotten tomatoes rating, as per the critics. This returns a list of actors (to take care of potential ties).

- **def audience_darling(movie_Db, ratings_Db)** - given the two dictionaries, we are interested in finding the actor whose movies have the highest average rotten tomatoes rating, as per the audience. This returns a list of actors (to take care of potential ties).

- **def good_movies(ratings_Db)** - this function returns the **set** of movies that both critics and the audience have rated above 85 (greater than equal to 85).

  Hint: Think about this using set theory and try and reuse a function that you have written before.

  Remember to return a set in this case.

- Given a pair of movies, return a list of actors that acted in both.

  Use **def get_common_actors(movie1, movie2, movies_Db).**

Remember that now that you have seen TDD, you have to write unit tests for each of the functions. While it might seem annoying at first, I highly recommend doing this in the proper test driven manner. First write the unit tests and then write code to make them pass.

Also, now that we have seen the concept of docStrings (the triply quoted comments at the start for a function), you are required to write a docString for each function you are creating.

Main Function

So far we have not mentioned the main function and that is because I would like you to come up with a design by yourself there.

The general idea is that you want to print some kind of welcome message and tell the user what your database contains - actor info and movie ratings. Then provide the user with options for the questions that you are able to answer. For instance something like 'press 1 for finding out the top rated actor' etc.

Depending upon the user's choice you might have to ask him/her for more input.

So for instance if they pick the co-actors option you just need to ask them for a single actor's name. But if they pick the common movie option, you want to ask them for two actors' names.

If an actor or a movie is not present in the database, simply print out 'not present' in that case. We do want the name of the actor or movie to match the name that we have in the database. The only leniency we will have is case sensitivity. So 'tom HANKS' is the same as 'Tom Hanks'. Use the **lower** function for this.

Remember to put this code at the end.

```
if __name__ == '__main__':
    main()
```

Extra credit - 2 points

Extra credit - Get an actor's Bacon number. Kevin Bacon is known for working on large ensemble cast movies. An actor's Bacon number is the number of links that have to be followed to get to Kevin Bacon. Kevin Bacon has a bacon number of 0. For instance, Kevin Costner has a Bacon number of 1 because he acted in JFK with Bacon. However, someone like Shirley Maclaine has a Bacon number of 2. I'll let you find the link.

def get_bacon(actor, movieDb).

Possible inconsistencies

If there are inconsistencies in the files that we have given you, please do not change them. We have tried to give you relatively 'clean' data, but data scraping and data cleaning is not the goal of this assignment. So please leave the files untouched.

What to submit

There are 3 things to submit here.

1. movie_trivia.py - actual code
2. movie_trivia_tests.py - a unit test file
3. Also if your unit tests are relying on any data, please supply that data to us.

How will this be evaluated

- Unit tests - 7 points. Make sure you write a bunch of test cases for each function. Also if you introduce helper functions, you need to write unit tests for those.
- Ensuring that the code works - 9 points. Make sure your database has all the functionality that we require. Implement each function. If there are common things that you are doing, please feel free to make more functions than the ones that we asked for.
- Style - 5 points. The usual things about modularity, function names, variable names. Also, every function that you write has to have unit tests, unless it is doing just printing or inputting.
- Main function (usability of your program) - 3 points. You have to make sure that your program is usable by someone who just wants to figure out something about movies and does not have any understanding about the actual structure of each function.