**CIT 590: Fall 2018**

**Homework 4**

HW deadline as per Canvas.

This homework deals with the following topics:

* Dictionaries
* Sets
* Databases using dictionaries (not too far from how they really work!)
* Test driven development (TDD)

In this HW, we will deal with representing movie data using dictionaries, with the goal of answering some simple movie trivia questions. For example, “what is the name of the movie that both Tom Hanks and Leonardo DiCaprio acted in?”

We will use 2 dictionaries. The first corresponds to information about an actor and all the movies that he/she has acted in. The second corresponds to information about the critics’ score and the audience score from <https://www.rottentomatoes.com/>, about the movies.

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

**Data Structures**

There are 2 dictionaries in this HW.

The first dictionary is keyed on actor name and has a list of movies as data. For example, {‘Meryl Streep’ : [“Doubt”, “Out of Africa”, “The Post”, “Sophie’s Choice”]}

The second dictionary is keyed on movie name and has a list of two ratings - critics rating and audience rating. For example, {“Doubt”: [79, 78]}  
  
  
**Testing**

You have now seen how *Test Driven Development* works. For this homework, you are required 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, implement the functions to pass your unit tests. Be thoughtful about writing a good suite of tests to ensure proper behavior for the functions. Test typical inputs and edge cases.  
  
  
**Starter Code**

We have provided you with 3 files to get started. The python file (*movie\_trivia.py)* reads from both the text file (*moviedata.txt*) and the csv file (*movieratings.csv*) and populates the two dictionaries with a minimal amount of data.

Download all 3 of these files and store them in the same folder. Then run *movie\_trivia.py* to make sure the starter code works. If there are any issues with the starter code, let us know ASAP.

Please get rid of the print statements in the *main* function once the starter code works. Those exist right now simply for you to verify that things work.  
  
  
**Suggested Approach**

Please do this homework in the TDD manner. In particular, once you have run the starter code successfully, do not worry about writing the *main* function until the very end. Write test cases for the first function, implement the function so that it passes those test cases, and then make sure your function is stylistically ok. After you finish this process for the first function, do it again for the second function, then the third, etc.

This suggestion applies for this homework and for almost all subsequent homework assignments as well!  
  
  
**Utility Functions**

The first step is to create some basic utility functions for interacting with the database.

You are allowed to modify the *movies.txt* and *moviescores.csv* file. Feel free to add more movies, actors, ratings, etc. The files we provide are very small and simple.

As you’ll see, we insist that the dictionary variables are passed to the individual functions as parameters. It is possible that a dictionary other than the ones created by the provided *movies.txt* and *moviescores.csv* will be passed to your functions. In fact, you can be guaranteed that the unit tests we write for checking your code will use different data. Our unit tests will conform to the formatting specified, but will have different values.

Here is the list of 5 utility functions that we want you to write. Be sure to add docstrings to your functions and comments to your code.

*insert\_actor\_info(actor, movies, actordb):*

* actordb​ 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 OR an update. If the actor is already present, it should append these new movies to their list.
* This function does not return anything. It will work by just modifying the movie database/dictionary passed to it.

*insert\_rating(movie, ratings, ratingsdb):*

* ratingsdb​ is the ratings database/dictionary that is to be inserted into/updated
* movie​ is the movie name as a string
* ratings​ is a list with 2 elements: the critics’ rating and the audience rating
* Similar to *insert\_actor\_info*, this function should update the ratings for a movie if the movie is already in the database.

*select\_where\_actor\_is(actor\_name, actordb):*

* Given an actor, return the list of all movies
* actor\_name​ is the name of an actor as a string
* actordb​ is the dictionary to get the data from
* This function 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 normally see when you are using a SQL database.

*select\_where\_movie\_is(movie\_name, actordb):*

* Given a movie, return the list of all actors in that movie
* movie\_name​ is the name of a movie as a string
* actordb​ is the dictionary to get the data from
* This uses the same database as *select\_where\_actor\_is*, but is less trivial to code

*select\_where\_rating\_is(comparison, targeted\_rating, is\_critic, ratingsdb):*

* This useful function returns a list of movies that satisfy an inequality or equality, based on the comparison argument and the targeted rating argument
* comparison​ is either ‘=’, ‘>’, or ‘< ‘ and is passed in as a string
* is\_critic​ is a boolean that represents whether we are interested in the critics’ rating or the audience rating. True = critic ratings, False = audience ratings.
* targeted\_rating​ is an integer
* Some examples of expected output are:
  + *select\_where\_rating\_is(‘>’, 0, True, ratingsdb)* should return every movie in the database, assuming no movie has a critic rating equal to 0.
  + *select\_where\_rating\_is(‘=’, 65, False, ratingsdb)* should return every movie that has an audience rating of exactly 65.
  + *select\_where\_rating\_is(‘<’, 30, True, ratingsdb)* should return every movie that has a critic rating less than 30. Basically the ones the critics hated!

**More Fun Functions**

Outside of the 5 utility functions above, we would like to be able to answer some questions when interacting with a user and hence these other functions are required. Remember code reuse when you write these functions!

*get\_co\_actors(actor\_name, actor\_db):*

* actor\_name​ is the name of an actor as a string
* actor\_db​ is the database/dictionary to search through
* This function returns a list of all actors that the given actor has ever worked with in any movie

*get\_common\_movie(actor1, actor2, actor\_db):*

* actor1​ and ​actor2​ are actor names as strings
* actor\_db​ is the database/dictionary to search through
* This function returns a list of movies where both actors were cast
* In cases where the two actors have never worked together, it returns an empty list

*good\_movies(ratingsdb):*

* ratingsdb​ is the ratings database/dictionary
* This function returns the ​set​ of movies that both critics and the audience have rated above 85 (greater than or 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

*get\_common\_actors(movie1, movie2, actor\_db):*

* movie1 and movie2 are the names of movies as strings
* actor\_db is the actor database/dictionary
* Given a pair of movies, this function returns a list of actors that acted in both movies
* In cases where the movies have no actors in common, it returns an empty list

**Main Function**

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

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’.

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’.

User input should NOT be case sensitive, but spelling must be correct. This means that ‘tOm HANkS’ is the same as ’Tom Hanks’, which is the same as ‘tom hanks’. Use the *lower()* function to convert all user input the lowercase.

Always remember to put this code at the bottom of your .py file:

if \_\_name\_\_ == ‘\_\_main\_\_’:

main()

**What to Submit**

There are 2 separate files to submit here. Do not submit them in a .zip, just the individual files.

1. *movie\_trivia.py*: the program code file
2. *movie\_trivia\_test.py*: the unit test file

**Evaluation**

* Your unit tests - 10 points. Make sure you write a lot 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 (passes our unit tests) - 15 points. Make sure your database has all the functionality that we require and implement every function.
* Code Reuse - 5 points. If there are common things that you are doing, you should breeak those out and create additional functions.
* Style - 4 points. The usual things about function names, variable names, docstrings for all functions, and comments for non-trivial code.
* Usability of the program (mostly in *main* function) - 6 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.