Project #5 – Movie Matchmaking

1 Objective

Implement varied selection logic in a realistic project. Create and use tuples in simple contexts.

2 Problem Overview

The company you work for is exploring the creation of a movie streaming service, Movie Me! Viewers will be asked to pick some of their favorite movies of all time so that the service can suggest other movies. They are asking you to implement their initial heuristics, matching a favorite movie with another movie to determine the probability the viewer will enjoy other movie.

3 Functions

Write these functions, implementing the scenarios described. Functions other than main should contain no user input and no display of information to the user. Please use the function names shown; my test code depends on exact naming.

Function	Purpose/Tasks	Parameters	Returns
main	Create tuples for two movies, call calcLikeProbability to determine probability, output results	none	none
calcLikeProbability	Determines the probability the viewer will like the second movie, given that the first is a favorite	Favorite movie (a tuple), another movie (a tuple)	Probability of liking, in range 0.0 (no chance) to 1.0 (certain)
calcDecadeProbability	Given two years, determine the probability the user will like other movies from the year the second movie was released	Year of favorite movie, year of another movie	Probability of liking, either 1.0 (movies in same decade), .5 (movies within adjacent decades), or 0 (movies farther apart)

4 Function Details

4.1 Global ~~Variables~~ constants

Create global variables for the weightings used to determine whether the viewer will like a movie: director ~~(.15),~~ lead actor (.20), decade (.1), and categories (.5).

4.2 main

In main, create two movies: a favorite movie and another movie to match against the favorite. Each one should contain this data, represented in a tuple:

Title, director, lead actor, year (an integer), then Boolean literals representing whether the movie falls into each of these categories: Anime, Family, Comedy, Drama, Horror, Romance, Sci-Fi, and Thriller.

It should display the two movies. It is okay to display them in tuple form.

It should then pass these two movies to calcLikeProbability, which will return a probability. That result should be shown in a message like this (note the single digit after the decimal point):

```
The probability the viewer will like the other movie is 58.3%
```

4.3 calcLikeProbability

This function is the heart of the algorithm, determining whether the viewer is likely to enjoy the second movie given their love for the first.

Movie Me! has determined that director is an important factor. If the second movie has the same director as the first movie, then the director probability mentioned above is added to the overall probability. Lead actor is also important; use the same logic here.

Call calcDecadeProbability and multiply its result by the decade probability. Then, add that to the overall probability.

Lastly, deal with categories. The goal here is to find the matches between the categories in which the favorite movie falls (e.g., Comedy, Horror) and the second movie falls (e.g., Horror, Sci-Fi). If there is an exact match in categories, you should return 1.0 to represent that. Otherwise, determine the percentage based on how many categories are in the favorite movie, e.g., in the examples in this paragraph, the favorite movie fell into two categories, but only one was matched, so the probability is 1/2 or 0.5. If there is no match, then you should return 0.0 here.

4.4 calcDecadeProbability

Movie Me! has also found that viewers tend to like movies from certain decades more than others. Compare the two decades to decide whether to return 1.0 (same decade), 0.5 (adjacent decade), or 0.0 (non-adjacent decade).

5 Code Specifications

- Place all functions in the same file; there is only one .PY file to submit for this project.
- You should need no additional functions; these should be enough.
- At the bottom of the program you should have a call to main().
- Include header comments at the top of the file. Include your name, the date, and a brief description of what the program does.
- Include comments for each section saying what's going on in the lines of code below.
- Use comments elsewhere as you think they help guide the reader. Don't overdo, though! Not every line needs a comment; think about describing a block of related code.
- Use blank lines to separate sections and provide visual "breathing room."
- Use descriptive variable names.

6 Hints

- Functions should have only **one** return statement, the last statement in the function[1].
- While turning the problem description into code, remember that your first attempt will probably not achieve the **simple elegance** the problem deserves. Think carefully about finding simpler/shorter code paths without sacrificing readability or making it hard to map the problem to the code. Also remember that just because something is mentioned first in the problem description, that doesn't mean it's the first thing in your code; you can often simplify by reorganizing.
- Functions dealing with Boolean expressions and/or returns should achieve "**Boolean Zen**;" they are often simpler and shorter than you first imagine.
- **Code duplication** is undesirable; watch for situations where it can be eliminated or minimized.

7 Testing

- Develop an appropriate number of test cases. Think about boundaries, places where behavior switches from one result to another, wrong data types (which you may not know how to fix—true—but document them and note what you need to learn for the future). Document test cases in your summary write-up.

8 Summary

At the bottom of your program, add comments that answer these questions:

- How did you approach this assignment? Where did you get stuck, and how did you get unstuck?
- What doesn't work as you'd like, perhaps things that you'd like to fix as you learn more?
- What did you learn from this assignment? What will you do differently on the next project?

9 Extra Credit: Tuple within a Tuple

We would like to break out category handling into its own function; but having to pass that data or repackage into another tuple seems onerous. So, when creating the movies, make the category list a tuple within the overall tuple. In calcLikeProbability, receive it into a single variable and pass it to a new function calcCategoryProbability where it can be unpacked and used. Move pertinent code from calcLikeProbability into the new function.

(continues…)

[1] This is not a hard and fast rule in the real world, but it is a good general habit. Remind me in class if you would like to have a discussion regarding recommendations for real-world use.

10 Grading Matrix

Area	**Pct**
Tuple creation	5
main	
tuple creation	5
display of movies, function call, display of results	10
calcLikeProbability	
tuple handling	5
function calls and returns	5
calculation correctness	20
general coding	5
calcDecadeProbability	
calculation correctness	10
general coding	5
Test cases	20
Comments, variable names, white space	5
Summary report	5
Extra Credit	5
Total	**105**