1.) **Write a program that prints out the top-5 highest-rated movies for each genre.**

Usage: python3 highest_ranked.py

To do this, I considered the different ways that this app would define highest-rated movies. I chose the literal definition, taking the movies with the highest average rating but I imagine that in other rating systems, the highest rated movies might be weighted by the number of people that rated them (I.e a movie with an average rating of 4.5 with 10,000 ratings would be considered better than a 5.0 movie with 2 ratings)

For the final output, my program prints out the top rated movies but also includes functionality that returns the highest rated movies with the most number of user ratings in the case of a tie.

```
Output Format: [(Movie ID, Number of Ratings, Rating)] ->

Genre:  Children
[(3612, 1, 5.0), (4822, 1, 5.0), (5765, 1, 5.0), (6332, 1, 5.0), (32289, 1, 5.0)]

Genre:  Adventure
[(7087, 2, 5.0), (53, 1, 5.0), (2880, 1, 5.0), (3612, 1, 5.0), (4565, 1, 5.0)]

Genre:  Action
[(876, 1, 5.0), (2880, 1, 5.0), (3165, 1, 5.0), (3837, 1, 5.0), (3879, 1, 5.0)]

Genre:  Comedy
[(74727, 2, 5.0), (99764, 2, 5.0), (183, 1, 5.0), (559, 1, 5.0), (876, 1, 5.0)]

Genre:  Western
[(3737, 1, 5.0), (4428, 1, 5.0), (5301, 1, 5.0), (32515, 1, 5.0), (144976, 1, 5.0)]

Genre:  Horror
[(183, 1, 5.0), (1692, 1, 5.0), (1771, 1, 5.0), (2636, 1, 5.0), (2647, 1, 5.0)]

Genre:  Romance
[(9010, 2, 5.0), (32525, 2, 5.0), (301, 1, 5.0), (764, 1, 5.0), (1455, 1, 5.0)]

Genre:  Documentary
[(759, 2, 5.0), (6598, 2, 5.0), (1420, 1, 5.0), (1819, 1, 5.0), (2627, 1, 5.0)]

Genre:  Film-Noir
[(3656, 1, 5.0), (6273, 3, 4.833333333333333), (3966, 2, 4.75), (5017, 3, 4.666666666666667), (7126, 1, 4
.5)]

Genre:  War
[(1450, 1, 5.0), (2897, 1, 5.0), (6107, 1, 5.0), (8675, 1, 5.0), (26150, 1, 5.0)]

Genre:  Thriller
[(183, 1, 5.0), (876, 1, 5.0), (2982, 1, 5.0), (3216, 1, 5.0), (3656, 1, 5.0)]

Genre:  Fantasy
[(99764, 2, 5.0), (2086, 1, 5.0), (3216, 1, 5.0), (3612, 1, 5.0), (3837, 1, 5.0)]

Genre:  Animation
[(99764, 2, 5.0), (5765, 1, 5.0), (7355, 1, 5.0), (84414, 1, 5.0), (92210, 1, 5.0)]

Genre:  Drama
[(3038, 4, 5.0), (309, 3, 5.0), (3112, 3, 5.0), (1859, 2, 5.0), (6918, 2, 5.0)]

Genre:  Musical
[(1819, 1, 5.0), (2573, 1, 5.0), (3612, 1, 5.0), (4789, 1, 5.0), (6725, 1, 5.0)]

Genre:  Mystery
[(74727, 2, 5.0), (2945, 1, 5.0), (3656, 1, 5.0), (4522, 1, 5.0), (5062, 1, 5.0)]

Genre:  Sci-Fi
[(99764, 2, 5.0), (1692, 1, 5.0), (3837, 1, 5.0), (4626, 1, 5.0), (5062, 1, 5.0)]

Genre:  Crime
[(74727, 2, 5.0), (876, 1, 5.0), (3656, 1, 5.0), (3947, 1, 5.0), (4466, 1, 5.0)]

Genre:  (no genres listed)
[(122888, 1, 5.0), (128620, 1, 5.0), (140763, 1, 5.0), (160590, 1, 5.0), (117192, 2, 4.75)]
```

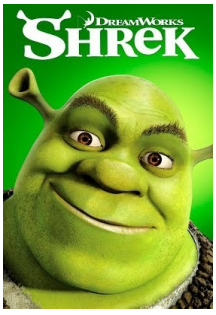
2.) **Write a program that prints out the 5 most similar movies to a given input movie, assuming that two movies are similar if people who like one movie tend to like the other movie. Write a couple of sentences justifying your choice of similarity.**

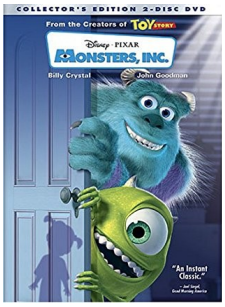
Usage: python3 recommender.py <movie id>

I believe that the most important metrics of movie similarity is 1.) User's who liked movie A also rated movie B with around the same score, 2.) Movie A and Movie B contain the same genres and 3.) Popularity, around the same number of people watched both Movie A and Movie B. For this problem, I created a python module (recommender.py) that takes movies with the same genre and outputs the top 5 movies with the best normalized-weighted-euclidean similarity.

Input Movie:

Similar Movies:





Terminal Output

```
input movie:  [1, 247, 3.8724696356275303]

 Top n Movies, (Movie Id, Similarity score): [(4306, 0.41971566469469185), (4886, 0.6724401190947795), (3
114, 0.7012881300987872), (2987, 0.8051878089034052), (2294, 1.1528989301463008)]
```

3.) **Let's say you use your similarity function as a way to recommend movies. How would you evaluate the performance of your system? Write a paragraph or so explaining your performance metric and test methodology.**

For an app in production, I would have a feedback mechanism from the users. Once a user tells our app that they enjoyed a movie, the app will recommend other movies that they might like based on other users who enjoyed the movie they just watched. If the user then clicks on a recommended movie and rates it positively, that would be a positive evaluation. If they left a negative rating or maybe even didn't rate the recommend movie, that would be a negative evaluation. For an advanced movie recommending agent, I could even implement a reinforcement learning model that rewarded our recommender system for positive evaluations and penalized the agent for negative evaluations.