CS 101 – Problem Solving & Programming I
Program 5 – Netflix Ratings
Algorithm Due Sunday night, 10/7
Program Due Sunday night, 10/14

Many online businesses have a system to recommend products; for example, Amazon's "People who bought this book also bought these books" feature. Netflix has a recommendation system as well; if you rate movies you've liked (or didn't like), the service attempts to recommend other movies you're likely to enjoy as well. Netflix recently offered a $1,000,000 prize for an algorithm that improved recommendations at least 10% over their current system.

So how do we make recommendations? A simple approach might be to just look at the average rating a movie has, and make the same recommendation for every customer. For example, we might find the 5 movies that have the highest average rating, and recommend them to everyone. We could 'personalize' the rating by removing any movies the person's already seen, and recommend the 5 highest rated movies from what's left. In this case, the only information we need is whether or not someone has seen a particular movie.

A more sophisticated approach would be to consider the user's actual ratings in the past, and see how those ratings compare with other customers'. After all, if a friend tells you they really liked several movies, and you also liked them, you're more likely to accept their recommendation regarding a movie you haven't seen. On the other hand, if your friend really liked several movies that you hated, and hated several movies that you really liked, you probably won't take their recommendation as readily.

One way to calculate how similar two people's preferences are is to treat each of their ratings as a vector, and calculate the dot product of the vectors. In this case, a vector is just a list of integers. The dot product is just the sum of the products of corresponding terms. Here's an example. Suppose we use this rating system:

-5	Hated it!
-3	Didn't like it
1	Neutral, so-so, not really hot or cold about it
3	Liked it
5	Loved it!
0	Didn't see it

Now suppose we have 3 movies in our database, and
Alfred rated them [-3, 5, 0]
Bruce rated them [-5, 0, 3]
Chuck rated them [1, 5, 3]
David rated them [5, -3, -5]

We want to make a recommendation for Alfred.
Alfred's similarity score to Bruce is: (-3*-5) + (5*0) + (0*3) = 15
Alfred's similarity to Chuck is: (-3*1) + (5*5) + (0*3) = 22
Alfred's similarity to David is: (-3*5) + (5*-3) + (0*-5) = -30

We see that Alfred's ratings are most similar to Chuck's. We'd look at Chuck's score and note that Chuck liked movie #3, which Alfred hasn't seen, so we'd recommend that Alfred should consider seeing that movie.

Note that this system scores two users as similar if they both liked a movie, or if they both disliked it. If one person liked a movie and the other person didn't, that decreases the similarity. (Take a look at Alfred and David above. Each liked a movie the other strongly disliked, so their similarity score is sharply negative.)

Your task is to write a program that takes people's movie ratings and makes recommendations to them using this technique.

Specification:
You are given two data files. One contains a list of movie names, one name per line. These are listed in the same order they are rated. The other data file contains the actual ratings. Each line of this data file contains the ID number and the ratings of one member. Ratings are integers, separated by commas, and there are as many ratings in each as there are movies listed (there is no missing data).

Note that data will be read from the file as strings, one line per string, and ratings will need to be converted to a list of integers. Store the ratings in a dictionary, indexed by the user's ID number.

Your main program will ask the user for their ID number. Iterate through the ratings, and find the set of ratings that is most similar to that users' ratings. Obviously, you should not calculate the similarity of users with themselves. Once you have found the member with the highest similarity to the user, scan that other member's list. Identify any movies that they rated positively (3 or higher) that the user has not seen, and print the titles of these movies to the screen, one title per line.

Extra credit! There are two extra-credit components of this assignment. You can do either, or both.

- For **5 points** extra credit, also add recommendations about movies to avoid. This is done by finding the most similar person and reporting on movies they *didn't* like.
- For **10 points** extra credit, modify your recommendation system to find the *five* members most similar to the user, and recommend all movies that were seen by at least 2 of those members (but not by the user) and have an average rating of at least 3.0

Development notes:

- This program seems much more complex than it is. Careful planning will pay handsome dividends. Start early, but start by thinking!
- All data from the file will be read in as strings. You will need to convert some of the data to integers.
- Reading from a file can be done in one of two ways. Here's one:
  `FileContents = open('ratings.txt').readlines()`
  This reads the contents of the file as a list of strings, one line per string. You can then iterate through the list as you would any other.

- Here's another way:
  `for line in open('ratings.txt'):`
  This will iterate through the contents of the file, one line at a time. Again, each line is a string.
- Dictionary keys must be immutable. So they can be either integers or strings. Use whichever seems best.
- If two or more members have identical similarity ratings to the user, break ties however you like (the first found, the last found, random selection, etc).

Here's a transcript of a sample run, run in WING 101.

```
Python 3.2 (r32:88445, Feb 20 2011, 21:29:02) [MSC v.1500 32 bit (Intel)]
Type "help", "copyright", "credits" or "license" for more information.
[evaluate Program05.py]
What is your user number? 34
You are most similar to user # 30
You may want to consider seeing:
The Hunger Games
Iron Man 2
Inception
Despicable Me
How To Train Your Dragon

Likewise, you may wish to avoid:
Ted
>>>
```