CSc 110, Spring 2017
Programming Assignment #10: Recommendation System
Due Wednesday, April 11, 2017, 7:00 PM

thanks to Michelle Craig of the University of Toronto and Marty Stepp of Stanford

This program focuses on using `sets` and `dictionaries`. Turn in a file named `recommender.py` on the Homework section of the course web site. You will need `review.py` from the web site; place it in the same folder as your program.

Background:

If you've ever bought a book online, the bookseller's website has probably told you what other books you might like. This is handy for customers, but also very important for business. In 2010, online movie-rental company Netflix awarded one million dollars to the winners of the Netflix Prize. The competition simply asked for an algorithm that would perform 10% better than their own algorithm. Making good predictions about people's preferences was that important to this company. It is also a very current area of research in machine learning, which is part of the area of computer science called artificial intelligence.

What you will do:

So how might we write a program to make recommendations for books? Consider a user named Carlos. How is it that the program should predict books Carlos might like? The simplest approach would be to make almost the same prediction for every customer. In this case the program would simply calculate the average rating for all the books in the dataset and display the highest rated book. With this simple approach no information about Carlos is used.

We could make a better prediction about what Carlos might like by considering his actual ratings in the past and how these ratings compare to the ratings given by other customers. Consider how you decide on movie recommendations from friends. If a friend tells you about a number of movies that s(he) enjoyed and you also enjoyed them, then when your friend recommends another movie that you have never seen, you probably are willing to go see it. On the other hand, if you and a different friend always tend to disagree about movies, you are not likely to go to see a movie this friend recommends.

A program can calculate how similar two users are by calculating the distance between their ratings. Calculate the distance by finding the sum of the products of their ratings for each book that they both have rated. For example, suppose we have the following data, Carlos: `Harry Potter=5, Animal Farm=1,1984=-5`, Suelyn: `Harry Potter:1,1984=5, Animal Farm=-3`, Bob: `Lord of the Rings=5, Harry Potter=-3, Cats=5`, and Kalid: `Animal Farm=-3, Harry Potter=3, Watership Down=5`. The similarity between Carlos and Bob is calculated as: 5 x -3 = -15. The similarity between Carlos and Suelyn is: (5 x 1) + (-3 x 1) + (5 x -5) = 5 + -3 + -25 = -23. The similarity between Carlos and Kalid is (-3 x 1) + (3 x 5) = -3 + 15 = 12. We see that if both people like a book (rating it with a positive number) it increases their similarity and if both people dislike a book (both giving it a negative number) it also increases their similarity.

Once you have calculated the pair-wise similarity between Carlos and every other customer, you can then identify whose ratings are most similar to Carlos's. In this case Kalid is most similar to Carlos, so we would recommend to Carlos the books from Kalid's ratings that Carlos hadn't already rated.

You may choose either option if there is a tie between closest users. Recommendations may be printed in any order.

Your task for this project is to write a program that takes people's book ratings and makes book recommendations to them using both techniques described here. We will refer to the people who use the program as "users".

Implementation Details:

You will write one program; `recommender.py`. The file contains sets of four lines containing the username, title, author and rating in that order. You must use Python's `dictionaries` and `sets` to complete this assignment. When your program starts it should load all the ratings from `ratings.txt`. Your program should prompt the user with options to add a rating, get recommendations, get the best rated book or quit. Your program should keep prompting the user for a next task until the user types "`quit`". When this happens the program should halt.

It must be possible for the user to choose options multiple times in any order and get the correct results each time.

You are required to store your rating data in a dictionary mapping a name to set of review objects.

`Review Class` *(provided by the instructor):*

We have provided you with a class named `Review` that your `recommender` program will use. A `Review` object represents a single review of a book. It keeps track of the name of the book, the book's author and the rating a user gave it.

The `Review` class provides the following constructors and methods that you should use in your program.

Method	Description
`Review(`**`title`**`, `**`author`**`, `**`rating`**`)`	Constructs a new `Review` object.
`get_title(), get_author(), get_rating()`	Returns the state of the review as passed to the constructor.
`__str__()`	Returns a text representation of the review.

You can look at the contents of the provided `review.py` to answer any further questions about how it works.

You will need to place `review.py` in the same directory as your program and import it.

Your program should match the expected output files provided on the course web page exactly.

Development Strategy and Hints:

We suggest the following development strategy for solving this program:

1. Write a function to read the input file and create the initial dictionary mapping username to sets of books rated.
2. Write a main that prompts the user for what they want to do next. You may need to prompt for additional information depending on what they want to do. See sample outputs for details.
3. Implement the add functionality so that values can be added to your dictionary
4. Write code to find the best book overall. To do this, create another dictionary mapping the title of each book to a tuple containing the sum of all of its ratings and the number of ratings. Then, loop over the dictionary to find the book with the highest average review (calculate average by dividing sum of all ratings by number of ratings).
5. Calculate recommendations for a particular user. To do this, start by creating a map of users to distances form the user you are trying to make a recommendation for. To fill this, go through your dictionary that maps users to their reviews and for each user go through all of their review. For each of those reviews go through all of the reviews of the user you are trying to make a recommendation for and if the book titles are the same, multiply the ratings of the two users and add them to the similarity score.

   Once the dictionary is filled, go through it and find the person with the minimum distance score. Output all of the books that the person with the minimum distance has read and rated a positive number that the person you are trying to make a recommendation for hasn't.

You can test the output of your program by running it on the provided input files as well as creating an input file of your own. You can also use our Output Comparison Tool to see that your outputs match what is expected.

Style Guidelines and Grading:

Part of your grade will come from appropriately using dictionaries and sets. You may not use any other structures beside dictionaries, sets and tuples (only where described in the development strategy).

Redundancy is always a major grading focus; avoid redundancy and repeated logic as much as possible in your code. Divide your code into a set of functions that captures the structure of the program.

Follow good general style guidelines such as: appropriately using control structures like loops and `if`/`else` statements; avoiding redundancy using techniques such as functions, loops, and `if`/`else` factoring; good variable names, and naming conventions; and not having any lines of code longer than 100 characters. You may have no global variables and you may not nest functions inside other functions

Comment descriptively at the top of your program, each function, and on complex sections of your code. Comments should explain each function's behavior, parameters and returns.