HW 10 Music Recommender+

November 11, 2020

Overview

In this assignment, you will create a music recommender system similar to the one described in chapter 5 of the textbook, so read the chapter before reading the rest of this assignment. The learning objectives of this assignment are:

- Practice with imperative programming (while-and for-loops) and mutable data types (lists and dictionaries).
- Practice designing a program that combines several features and needs to be implemented using several functions.

Important:

- Do exactly as instructed in this assignment to receive full marks. You may import from the cs115 module if needed.
- All text in this document may be considered a requirement. At no point should your program crash.
- If you would like to make your program do different things, go for it! But include a fancy-mode switch, like the 'debug' variable in the tictactoe game (slide 15 of Lecture 9). Use it in your program so if it's false, the program behaves exactly as it is shown in the spec document. If fancy-mode is true, then your program can do whatever cool stuff you like. When you submit, make sure fancy-mode is switched off. Write a comment in Canvas to let the CAs know that you did something super cool so we can check it out.
- Again, if it hasn't been said enough already, make sure your program behaves EXACTLY as shown in the Example Interaction section of this document. There are two reasons the program should behave exactly as specified. First, it helps us grade more automatically (and hence consistently). Second, in real projects several people collaborate on different components; those components have to fit together exactly or the application won't work. So get used to following specifications precisely. If you have any questions about this behavior, feel free to reach out to a CA to make sure you are implementing the print statements correctly :)
- Of course, don't cheat. This exercise has been used before so it wouldn't be hard to find solutions online. Unlike some of the previous exercises, this one is more involved so it is extremely unlikely you would come up with a solution very similar to someone else's. So it's very likely that cheating will be detected.

Notes and Suggestions

- This is the most challenging assignment yet, so start early, and read the requirements section of this document carefully. It provides exact specifications that you must follow in order to receive full marks on this assignment. And read the textbook, which provides a lot of code that can help you, even though our version has a few extra features.
- The requirements section is broken down into the exact functionality your assignment must provide, and the output which it must produce. However, you must decide on the implementation details yourself.
- As this assignment is meant to help you focus on while loops, for loops, lists, and dictionaries, you should think about how you might use these tools for each task you complete. Reviewing your ideas and thinking about alternatives and improvements before proceeding with your plan can help you create better code.
- Plan for unit testing: Early on, make notes on how to test the main functions. Define the test cases so the functions can be tested separately, before combining them. These can be assert statements, print outs, pyunit tests, or whatever you feel most comfortable with, but remember to comment out these tests before submitting so that your program is able to be graded by the auto-grader.

Requirements

The Basics

1. Name your source file musicrecplus.py.
2. When the program starts, it loads the database from the file named `musicrecplus.txt`, if it exists. Otherwise it creates the file. The file stores the database using the following format:

   ```
   UserName:Artist1,Artist2,Artist3,...
   ```

   Have a look at the sample database files provided with the assignment, and note the following.

   - The artist list must in sorted order, according to the Python comparison order for strings, and not have duplicates. Artist names are standardized using "title case", i.e., capitalized (see the `.title()` function used in the textbook). For example, if the user entered artists "eminem", "TMBG", and "Kerkylas of Andros", the program will convert those to "Eminem", "Tmbg", and "Kerkylas Of Andros". So they will be stored in the file, and printed, in the converted form.
   - The lines of the file should be in order by user name, with no duplicate users.
   - The user's name is allowed to end with the character `$`, which indicates that the user prefers their information to be kept private. This is called a *private mode user* and it will be discussed in some of the features described later. (But we aren't going to implement anything that protects access to the database file by other programs.)
   - If the file `musicrecplus.txt` does not exist upon starting the program, create it.

- Notice that no spaces should appear between commas and artist names or the colon and the last name of the user and the first artist. Please follow this format very carefully and closely.

3. The user should be prompted for their name:
   - If the user is a new user (not already in the `musicrecplus.txt` file), they should be prompted to enter their initial preferences before they can move on to the menu.
   - If the user is not new, they should not be asked their preferences and should immediately be shown the menu.
   - The user should be prompted for their name with the phrase

     ```
     Enter your name (put a $ symbol after your name if you wish your preferences to remain private):
     ```

   - If the user adds a `$` as the last character of their name, they are in private mode, which affects the recommendations phase.
   - There should be a space after the colon in the prompt.

The Menu

- The menu should display EXACTLY as follows:

  ```
  Enter a letter to choose an option:
  e - Enter preferences
  r - Get recommendations
  p - Show most popular artists
  h - How popular is the most popular
  m - Which user has the most likes
  q - Save and quit
  ```

- When the menu prints, it should receive user input.
- If no option is chosen, or an invalid option is chosen, the menu should reprint and request input until the user enters a valid option.
- The main program should be structured as a while-loop that repeatedly offers the user the choice to do one of the menu options. When he or she quits, it should halt after saving the file.

Functionality

Enter Preferences: The user enters their preferences (artists they like) one at a time, similar to the basic program in the textbook:

- This will replace the preferences already saved in the database under that user.
- The user should be prompted by:

  ```
  Enter an artist that you like (Enter to finish):
  ```

  The program should keep asking the user for preferences until the user enters the empty string (i.e. until they press enter).

- After the user finishes entering preferences, the database should be updated immediately so that the other functions proceed using the new preferences. (In memory, not saved to a file.)

After the user finishes entering preferences, they should be returned to the menu, which should be awaiting their next choice. It should be the same after each of the operations (except Quit), so we won't repeat this point below.

Get Recommendations: This option will also work similarly to the basic program in the textbook. However, it is a separate menu option and must follow these requirements:

- The recommendations should come from the user with the most similarity to the current user.
- But, if another client's preferences are the same as, or a subset of, the user, then ignore that client's preferences. In other words, the "best match" should be one that overlaps the most while also having at least one different artist to recommend.
- If there is a tie for the best match, your program should just pick one (for example, the first one it finds). (We will avoid testing what happens with ties.)
- Users in private mode should be excluded from these calculations.
- The recommendations should appear one per line.
- Artists' names should not be included more than once in a single list of recommendations.
- The user should not be recommended any artists they already have entered a preference for.
- Recommendations should be sorted.
- If there are no preferences available for recommendation, the user should be told:

  ```
  No recommendations available at this time.
  ```

Show Most Popular Artist: Print the artists that are liked by the most users.

- The top 3 artists should be printed, one per line, starting with the most popular.
- Users in private mode should be excluded from these calculations.
- If there are no artists found by this operation, display to the user

  ```
  Sorry, no artists found.
  ```

- If there is a tie for most popular, either choose one or print them all. (We won't test for this situation.)

How Popular Is The Most Popular Artist: Print the number of likes the most popular artist received.

- Print only the number. Do NOT print the artist!
- Users in private mode should be excluded from these calculations.
- If there are no artists found by this operation, display to the user

  ```
  Sorry, no artists found.
  ```

- In the event of a tie, still only print this number once.
- Note the similarity between this and ShowMostPopularArtists. Consider helper functions that help with the functionality of both of these functions at the same time.

Which User Likes The Most Artists: Print the full name(s) of the user(s) who like(s) the most artists.

- Print only one user name per line, in sorted order.
- If there are no artists found by this operation, display to the user

  ```
  Sorry, no user found.
  ```

- Users in private mode should be excluded from these calculations.
- If there is a tie, you can pick one of the users, or print them all. (We won't test for this situation.)
- The current user should be included in this computation.

Save And Quit: When the user chooses to quit, the current database should be written to the `musicrecplus.txt`, replacing old contents (if any).

- The name of the file saved to must be `musicrecplus.txt`.
- You must save according to the file format described above.
- If the file exists, overwrite the contents.
- If the file does not exist, you should create it with the proper contents.
- After this saving occurs, the program should exit (without crashing, of course).

Example Interactions

Example 1: Before this interaction, `musicrecplus.txt` does not exist. See `example1.py` to see how the program should interact with the user. Make sure your program works exactly the same way! See `musicrecplus_ex1.txt` to see the `musicrecplus.txt` file after this interaction.

Example 2: See `musicrecplus_ex2_a.txt` to see what should be in `musicrecplus.txt` BEFORE executing this example. See `example2.py` to see how the program should interact with the user. Make sure your program works exactly the same way! See `musicrecplus_ex2_b.txt` to see the `musicrecplus.txt` file after this interaction. Good luck!