Music Player

Introduction

For this project implementation we have chosen to mimic a music player system. To keep the scope minimum we are not going to actually play any music, rather focus will be to build an application that should exhibit the similar behavior. Users will be allowed to create multiple playlists based on their mood. Users can pick songs from any of the created playlist and will be allowed to play the music file they have chosen.

Playlist will be created from the list of songs that is available. There will not be any playlist available for the user by default. Once the user logs in into the system, they can create their own playlist by adding available songs from the default list. Users are allowed to add or remove songs from their playlist at any point of time. Some of the feature of the playlist is mentioned below:

- Creation of playlists
- Adding/removing a song in the playlist
- Shuffle option in the music player
- Play the playlist
- Sorting playlist based on the song name

Housekeeping points

- This is a minimal example and may not follow some standard practices
- The focus is on the main flow, with minimal error handling. Errors in validation logic should be handled appropriately though.

Program Organization

This section will contain the information related to the different files available in the project.

1. C02_P01_Music_Player.py:
    a. **MusicPlayer Class:** This is the class structure and constructor is defined in it to save the information. This class will contain the implementation pertaining to the music player with features such as choosing a playlist to play, playing a song, shuffling a song. Adding songs in a playlist, removing songs from the playlist. This file will have following methods to implement:
        i. *create_playlist:* In this method, users will be allowed to create a playlist by providing the name of the playlist.
        ii. *play_playlist:* This will have a feature to play songs from the chosen and available playlist.

iii. *insert_song_in_playlist:* This method will enable the feature to add songs in an existing playlist.
iv. *delete_song_from_playlist:* This method will enable the feature to remove songs from a playlist.
v. *sorting_playlist:* This method will enable sorting the sorting feature on the songs available in the playlist.
vi. *shuffle_songs:* This method will enable the shuffling feature on the music player. Choosing this option means that instead of playing the music sequentially, any random music from the playlist will be played.
vii. list_all_playlists: This method will print all the playlists that are available
viii. search_playlist_by_name: Printing the playlist after searching it in the created playlists till now.
ix. delete_playlist: Deleting the playlist altogether from the list of available playlists.

b. LinkedList class: In this file there will be a structure to implement the creation of linked list, node insertion, updation and deletion. All these nodes will be designed to hold the data for a song. Each node will contain (*song_name, artist, year, length*). This file will contain following methods:

i. ***node_creation****:* This method will create the nodes based on the data that was provided. Please note that insertion and deletion of the node is different. This will merely create the node based on the received data.
ii. ***traversal****:* In this method we will be traversing through the linked list using a temporary pointer.
iii. ***insertion****:* In this method, you need to implement the feature to insert the node in the linked list.
    1. insert_at_start: This method will insert the song in the beginning of the linked list.
    2. insert_after: This method will insert a song after finding an specific song available in the playlist, if the song that we are looking for is not available insertion will not take place.
    3. insert_before: This method will insert a song before a node after finding an specific song available in the playlist, if the song that we are looking for is not available insertion will not take place.
iv. ***deletion****:* In this method, based on the requirement if you want to delete any specific node from the linked list, you should delete the node from the linked list.
v. ***sort_list***: This method will perform sorting on the available song in the playlist. The idea here is to perform sorting on the basis song_name that is available with each song node.
vi. ***shuffle_song***: This method will be used to play the song by choosing a random song that is available in the playlist. You can use the random library to pick a song from the list of songs that are available to choose.

2. ***app_data.csv:*** This file contains the list of so that are available for you to push in the linkedlist and for playlist creation. Please do not make any changes into this file. This file does not require any changes.
3. ***output.txt:*** This is a sample output file. Once your code is in working state and all the methods are implemented properly, your output should appear like this.

Problem Statement

The program structure is already set and there are specific methods that you are expected to implement. Please also read the comments in the code, especially in the methods to be implemented. You can modify main.py to add further demonstration calls.

1. ***deletion****:* In this method, based on the requirement if you want to delete any specific node from the linked list, this operation will help you in performing that operation.
    a. This method will be invoked by music player class, that will look for a specific song in the playlist and remove that from that particular playlist.
    b. Please note that deletion of a song means you are updating the playlist that was created in the main function.

2. ***sort_list***: This method will perform sorting on the available song in the playlist. The idea here is to perform sorting on the basis song_name that is available with each song node.
    a. This method will be called again using the music player class and this will sort all the songs available in the linked list using the song name.

3. ***shuffle_song***: This method will be used to play the song by choosing a random song that is available in the playlist. You can use the random library to pick a song from the list of songs that are available to choose.
    a. This method will pick a song randomly from the chosen playlist.
    b. Every time you are calling this method a different song should be chosen and played.

Evaluation Rubric

Total Project Points: **20**

- Basic compilation without errors (**10%**) : **2 Points**
- Correctness:
    - Problem statement - 1 (**60%**) : **12 Points**
    - Problem statement - 2 (**20%**) : **3 Points**
    - Problem statement - 3 (**20%**) : **3 Points**

Program Instructions

1. Download the zipped folder named **C02-P01-Music-Player-System.zip**, and unzip it on your local machine. Go into the directory named **C02-P01-Music-Player-System**

2. Make sure that you have Python 3.6, or higher, installed. At your command prompt, run:
   ```
   $ python --version
   Python 3.7.3
   ```

   If not installed, install the latest available version of Python 3.

3. To run the code in the source code folder, run the following command:
   ```
   $ python3 C02_P01_Music_Player.py (On many Linux/Mac platforms)
             OR
   $ python C02_P01_Music_Player.py            (On      Windows/Mac
   platforms)
   ```

   In any case, one of these two commands should work.

4. Alternatively, you could install a popular Python IDE, such as PyCharm or Visual Studio Code, and select a command to build the project from there.

5. You will be making very frequent changes into the C02_P01_Music_Player.py python files. The idea is to complete both the scripts so that it satisfies all the requirements. Once you have completed the changes in the code and it is executed without any error. Zip the folder as **C02-P01-Music-Player-System.zip** & now it is ready for submission.