Project 1: Listen to the Music

Objectives

This laboratory project aims to:

- Create a custom queue data structure,
- Modify a queue's behavior in dealing data based on program requirements,
- Store objects and heterogenic data into a data structure,
- Utilize various data structures and their respective algorithms to develop a program with optimal solutions based on functionality requirements.

Overview

Spotify, YouTube, Google Play Music, you name it. Each of these application software always has playlists and queued music that makes our music life convenient and stylish.

In this Project, you will be creating a relatively simple Music Playlist utilizing the knowledge you have understood in your Data Structure and Algorithms (ITCC47), and Object-Oriented Programming (ITCC45) courses.

You will start from the scratch and develop a program on your own in this project. This is to enhance and further establish your programming skills so that you will understand creating projects on your own as it will be part of your skillsets later.

Note: Note that the examples shown below are just placeholders and you can alter the final output in your project as you wish as long as the content remains the same.

Project Requirements

In this project, your program must be able to do the following:

1. Create tracks, add them to the music library, and save the data permanently.
2. Create playlists and save their data permanently.
3. Initiate and establish music queue either from the music library or a particular playlist, then save its data if it is not cleared.
4. Search for tracks from the music library and add them to a playlist.

Functionalities

Music Library

The music library is composed of ***tracks,*** where each track has a title, an artist, additional artists (if multiple artists collaborate on the same track), an album, and a duration. The duration should be outputted in "mm:ss". Note that the duration is in string format but must be flexible like a numeric for addition operations later.

A track example is provided below.

Track Example

Title: Gangnam Style
Artist: PSY
Album: Psy 6 (Six Rules), Part 1
Duration: 03:39

Each track in the library is added through user input. They must be permanently saved and remains accessible even if the program is terminated. It is up to you how to store the data of each track.

All tracks added to the music library should be sorted based on their track title by default. If two tracks share the same track title, they should be sorted in the following hierarchy:

1. Artist name
2. Album
3. Duration

Playlists

Playlists are customized music libraries that has a playlist name and is composed of many distinct tracks. This means that no track should be inserted into the playlist if it has already been added. Furthermore, the playlist should summarize the total duration of the playlist. In addition, no two or more playlists should share the same playlist name.

Playlist Example

Playlist Name: My Playlist
Total Duration: 12 min 17 sec
Tracks:
Gangnam Style – Psy (03:39)
GAS GAS GAS EXTENDED MIX – Manuel (04:21)
Counting Stars – OneRepublic (04:17)

We can add tracks to the playlist either by going to the music library and selecting each song individually. We may also add new tracks to the playlist inside the playlist itself, and search for a track title. If it exists, then it should display all the tracks (if multiple tracks share the same name), and the user can select which of the tracks should be added to the playlist.

If more than 10 playlists exist, then a pagination should be implemented on the list of playlist menu interface. Refer to the example below:

List of Playlist Example

[1] My Playlist
[2] Tempo
[3] NewLove
[4] K-PaPop
[5] JPop
[6] Anime
[7] G(OLD)

[8] Pinoy Classic
[9] HeartBr3@k
[10] HearBre@k2

<Page 2 of 3>

[11] Previous Page
[12] Next Page

The playlist data should remain accessible even if the program is terminated.

Queues

When playing from the playlist or music library, a queue should be created of all the tracks present. Note that queues can either be shuffled or repeated (but not both as a general requirement). Whenever queues are unshuffled, it should return to its original arrangement.

For repeats, only consider an all-repeat state.

Whenever queues have repeat turned off, then when we skip the last song, then it should stop playing and display no more tracks left. However, if the play button is entered, then it should return to the first track of the queue. This should be done in $O(1)$ constant time.

Whenever queues have repeat turned on, then when we skip the last song, it should immediately return to the first track of the queue and keep on playing. In addition, if we are on the first track of the queue and we enter the previous music button, then it should proceed to the last track of the queue. All these operations should be done in $O(1)$ / constant time.

Regardless of the state of the queue, it should not lose its reference of the track currently being played. This means that even if the entire queue is shuffled, unshuffled, repeated, and non-repeated, then the track currently being played should remain the same.

When the queue plays / skips / previously goes to another song, the operation should be done in $O(1)$ / constant time.

Note that the user should be able to exit the music queue interface at any time, but it should remain on background so that if we revisit it again, then it should contain the same content. In addition, we can add additional tracks to the queue either by adding tracks one by one, or by adding an entire playlist.

To display the content of the queue, only the first 10 tracks should be shown. There should also be a counter at the bottom to determine the current ***page*** of the queue and how far it is to the end. Refer to the example below.

Queue Interface Example

Total Duration: 5 hr 24 min
Shuffled: No
Repeat: Yes
Tracks:

Currently Playing (Paused):

Feel This Moment – Pitbull, Christina Aguilera (03:50)

Next:

(1) Call Me Maybe – Carly Rae Jepsen (03:13)
(2) Timber – Kesha (03:24)
(3) Roar – Katy Perry (03:44)
(4) I Knew You Were Trouble – Taylor Swift (03:40)
(5) Just Dance – Lady Gaga, Colby O'Donis (04:02)
(6) Domino – Jessie J (03:52)
(7) Payphone – Maroon 5, Wiz Khalifa (03:51)
(8) Blinded in Chains – Avenged Sevenfold (06:34)
(9) Counting Stars – OneRepublic (04:17)
(10) Bad Romance – Lady Gaga (04:55)

<Page 1 of 9>

[1] Play
[2] Next
[3] Previous
[4] Turn off repeat
[5] Turn on shuffle
[6] Clear queue
[7] Exit

The total duration changes based on the duration of the entire queue. Note that even though the first 10 are only displayed, the rest of the songs in the queue should be present and is calculated using the ***pagination*** that is at the bottom of the queue interface.

The queue interface should have a menu below where the user can interact with it. On the example above, the queue is currently on pause.

If the queue is not cleared, then the queue's data should be accessible even if the program is terminated, or the queue interface is exited.

Data Storage

Note that most, if not all, data in the project should be permanently save. You can either use JSON to store the information, or you can utilize Comma Separated Values (CSV) files to store the information. You are not required nor allowed to utilize databases as they are too complicated for the current course.

Challenges

There are multiple challenges in this project, and you may optionally achieve them. Note that when a challenge is being achieved by the group, then it will be counted as part of the completion. As such, you should not attempt to finish the challenges if you are not capable of finishing them on time since it will result to point deductions and may even result to having no bonus points received.

All bonus points received will be distributed evenly between the final term exam scores of the two subjects, ITCC45 and ITCC47.

Shuffle and Repeat

In the general requirement, it is not required for the queue to be on the state of being shuffled and repeated at the same time. However, you can adapt to implement both states (shuffle and repeat) on the queue without compromising the queue of tracks being played. There are various variations of the shuffle and repeat states and regardless of the sequence that these two are activated on the queue, the original composition of tracks in the queue should not be altered, nor should it deviate in any way.

Completing this challenge will reward your group with 8 bonus points in total, which is then halved between the two

Album Tracking

Whenever a track is added, an album is specified on the track itself. As such, for every track created, then the program should automatically create a new album from that track. In addition, any succeeding tracks to be created under the same album should be added to that same album automatically.

Completing this challenge will reward your group with 6 bonus points.

Importing Tracks

Tracks can be imported to add a variation in the music library. They can either be in the form of JSON, CSV, or any similar file types. The user can opt to import new tracks, while also having the functionality to auto-create albums based on those tracks.

Completing this challenge will reward your group with 12 bonus points.

Sorter

Sorting is a classic programming problem, and so does to people with OCD. In this challenge, playlists should be sortable either by the following:

1. Date and Time created (default),
2. Playlist Name, or
3. Total Duration.

In addition, the playlists' tracks can also be sorted either by the following:

1. Date and Time added (default),
2. Track Title
3. Artist Name (if multiple artists, consider the main artists)
4. Duration

If two tracks share the same value of a particular attribute (aside from the date and time), then the tie breaker is determined strictly on the order:

1. Track title
2. Artist Name
3. Album
4. Duration
5. Date and Time

Completing this challenge will award your group with 20 bonus points.

Limitations and Warnings

You are forbidden in using most built-in Python classes. The only exception is when you are implementing code for reading / writing data storage permanently. All classes must be defined by you and your group. Violating this note will result in significant sanctions imposed by your instructor.

For any student with similar code submissions, ignoring variable/function/class name changes, white spacing, etc., your score will be divided upon how many other students is found to share such code similarity. Since you decide to submit the same output, then it is reasonable that the score will be divided upon each and everyone involved in such practice.

Submission

Your group must create a GitHub repository for the said project. Make sure to include your instructors as collaborators in the project repository. Your instructors will require you to put your final outputs in either master or main branches.