Polytechnic University of the Philippines - QUEZON CITY

Project in COMP 006 Data Structures and Algorithms

Semester and Academic Year: 1st Sem, A.Y. 2023-2024

Music Playlist Shuffle

Submitted By:

Dela Cruz, John Roodge

Magtangob, Mark Regie

Solano, Cedric Mark

Tubigan, Dynar

Year and Section: BSIT 2-2

Project Outline and Content:

1. Introduction

- **Briefly describe the project's purpose and scope.**
- **State the specific problem addressed by the data structure and algorithm.**
- **Outline the key data structure(s) and algorithm(s) implemented.**

The purpose of the "Music Playlist Shuffle" is to manage a music playlist. Users can create a playlist, add songs to it, shuffle the playlist randomly, and view information such as the current playlist and the number of times it has been shuffled. The scope includes basic features like creating a playlist, adding and removing songs, shuffling the playlist randomly, and checking the number of shuffle operations.

The specific problem addressed by the data structure and algorithm in this code is managing and manipulating a music playlist. The code uses a class named MusicPlaylist to represent a playlist, and it provides functions for adding, deleting songs, shuffling, and displaying the playlist. The playlist itself is stored as a list (self.playlist), and the shuffling is performed using the random.shuffle function.

The program uses a MusicPlaylist class to manage a list of songs. The playlist class has methods to add a song, delete a song, shuffle the playlist, and display the playlist and shuffle counter. The playlist is stored as a list, and the shuffle counter is an integer tracking how many times the playlist has been shuffled. The shuffle operation uses the random.shuffle() function to randomly rearrange the songs.

2. Problem Statement

- **Clearly define the problem to be solved, including inputs, expected outputs, and any constraints.**

The problem to be solved is to create a program that shuffles the song in a playlist and count the number of shuffles and displays it upon request.The program should enable users to perform various operations on a playlist, including creating the playlist, adding songs to it, shuffling the playlist randomly, and keeping track of the number of shuffle operations.

Inputs:

1. User input choices from the menu **(1 to 6)** to perform specific actions.
2. **When adding songs (1)**, the user inputs song names until they type '0' to stop.
3. **When deleting a song (2)**, the user inputs the name of the song to be deleted.
4. **Shuffle Playlist option (3)**, This option shuffles the order of songs in the playlist randomly.
5. **Display playlist (4)**, This option displays the current playlist.
6. **Display Shuffle Counter** (5), Display the number of Shuffle operations performed on the playlist.

7. **Exit (6)**, Allows users to exit the program.

Expected Outputs:

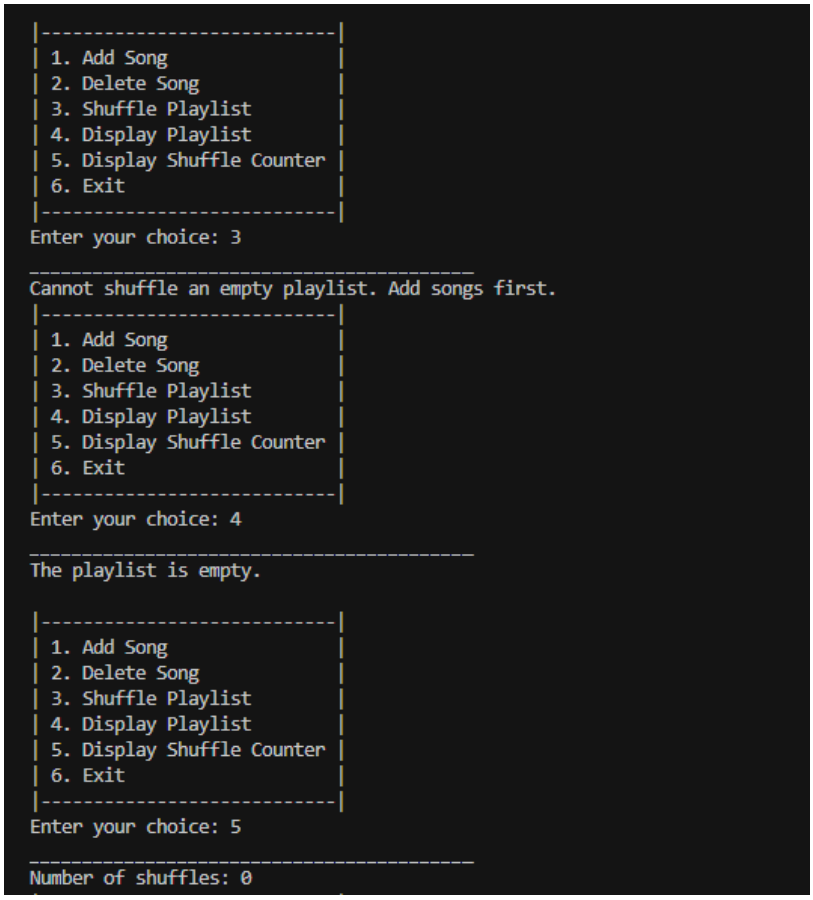
1. **Add songs (1)**: If the user adds a song successfully, the program outputs: "Songs have been added to the playlist."
2. **Delete song (2)**: If the song to be deleted is found in the playlist, the program output: (Name of Song) Has been removed from the playlist.
3. **Shuffle playlist (3)**: When the playlist is shuffled, the program gives you a message: " the playlist has been successfully shuffled. If you shuffle an empty playlist the program gives an error message: "Cannot shuffle an empty playlist."
4. **Display playlist (4):** If the Playlist is not empty, The program displays the current song in the playlist. If its empty the program gives you an error message: "The playlist is empty"
5. **Display Shuffle Counter** (5): Display the number of shuffle operations.
6. **Exit (6)**: Allows users to exit the program.

Constraints:

1. Song names must not be blank when adding to the playlist.
2. The playlist can be shuffled only if it is not empty.
3. You can't delete a song if it's not in the playlist.
4. The program runs in a loop until the user chooses to exit (option 6).

- **Provide examples to illustrate the problem.**

- **Error message when you shuffle and display an empty playlist.**



- **Adding songs In the playlist**

```
|--------------------------|
| 1. Add Song              |
| 2. Delete Song           |
| 3. Shuffle Playlist      |
| 4. Display Playlist      |
| 5. Display Shuffle Counter |
| 6. Exit                  |
|--------------------------|
Enter your choice: 1

----------------------------------------
Enter the song name type 0 to stop: "live Well" - Palace
Enter the song name type 0 to stop: "One of the Girls" - The weeknd
Enter the song name type 0 to stop: "Sunset" - Andy Shauf
Enter the song name type 0 to stop: "The Opposite of Afternoon" - Unknown Mortal Orchestra
Enter the song name type 0 to stop: "In The Meantime" - Space Hog
Enter the song name type 0 to stop: 0
Songs have been added to the playlist.
```

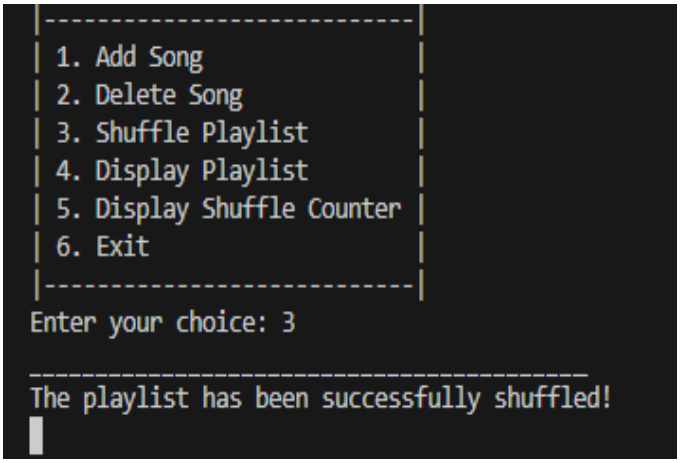
- **Deleting a song in the playlist**

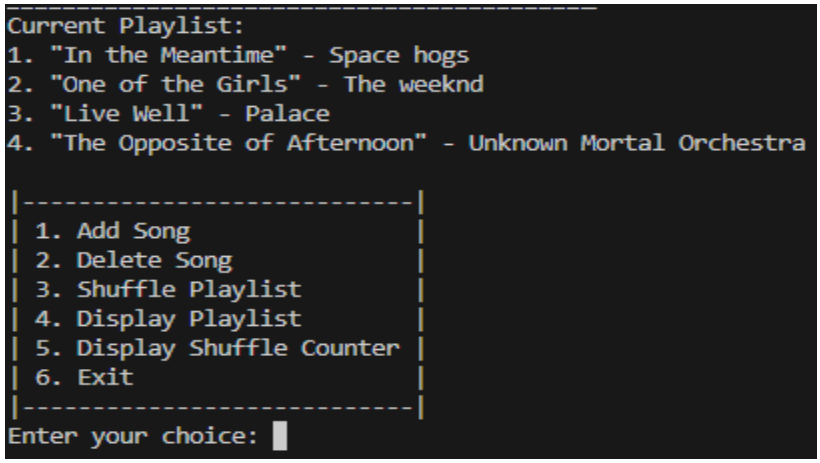
```
| 1. Add Song              |
| 2. Delete Song           |
| 3. Shuffle Playlist      |
| 4. Display Playlist      |
| 5. Display Shuffle Counter |
| 6. Exit                  |
|--------------------------|
Enter your choice: 2

----------------------------------------
Enter the song name to delete: "Sunset" - Andy Shauf
"Sunset" - Andy Shauf has been removed from the playlist.
```

- **Display playlist**

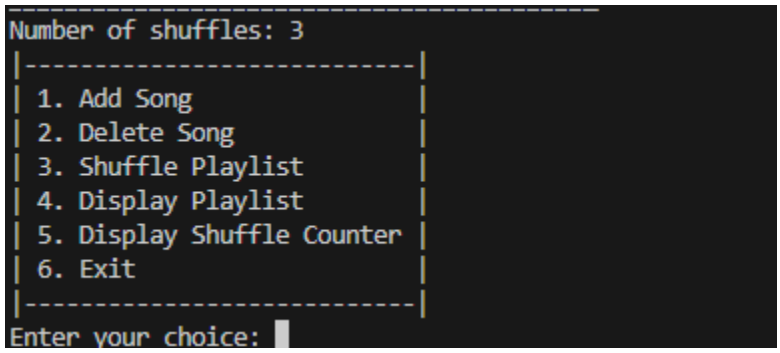
```
----------------------------------------
Current Playlist:
1. "Live Well" - Palace
2. "One of the Girls" - The weeknd
3. "The Opposite of Afternoon" - Unknown Mortal Orchestra
4. "In the Meantime" - Space hogs

|--------------------------|
| 1. Add Song              |
| 2. Delete Song           |
| 3. Shuffle Playlist      |
| 4. Display Playlist      |
| 5. Display Shuffle Counter |
| 6. Exit                  |
|--------------------------|
Enter your choice:
```

- **Shuffling the Playlist**





- **Display Shuffle Counter**



3. Data Structures and Algorithms

- **Describe the chosen data structure(s) and algorithm(s) in detail.**

Data Structure:

- **List (self.playlist):** The main data structure used in this program is a list. The playlist is represented as a list of songs. We can easily add, remove, and shuffle songs in this list. It's the main structure that holds all the songs we manage in the program.
- **Add songs (add_song):** The add_song checks if the input song name is not blank before appending it to the playlist. It ensures that only valid, non-blank song names are added.
- **Delete Song (delete_song):** This method removes the specified song from the playlist if it exists.
- **Shuffle Playlist (shuffle_playlist):** The shuffle_playlist shuffles the songs in the playlist using the Random-shuffle algorithm implemented by the random.shuffle() function.

- **Display Playlist (display_playlist):** The display_playlist prints the current songs in the playlist. It checks if the playlist is empty before displaying, and it uses a for loop to enumerate and print each song with its index.
- **Display Shuffle Counter (display_shuffle_counter)**: prints the number of times the playlist has been shuffled. Every time the **shuffle_playlist** method is called (when the user chooses to shuffle the playlist), the shuffle counter (**self**.**shuffle_counter**) is incremented by 1.

Algorithm:

In our case we use the **Random Shuffle Algorithm** in our problem.The random shuffle in this problem is implemented using the random.shuffle() function. It is used within the shuffle_playlist method of the MusicPlaylist class.The random shuffle process relies on the random.shuffle() function to randomize the order of songs in the playlist, providing users with an unpredictable sequence of songs when they choose to shuffle the playlist. **Linear search** are also used in this program because, when we want to delete a song from the playlist, we go through each song, starting from the beginning. We look at each song one after another until we find the one we want to delete or we check all the songs in the playlist. Once we find the song we want to delete, we remove it from the playlist. If the song isn't in the playlist, we still have to check every song to make sure.

- **Explain their functionality and how they are suited to the problem.**

  - **Add songs (add_song):** This function adds songs in the playlist and ensures that only non-blank song names are added to the playlist. If the input is blank, it prints "Invalid Input!" to notify the user. This functionality prevents the addition of songs with blank names, maintaining data integrity in the playlist.
  - **Delete Song (delete_song):** Removes a specified song from the playlist if it is present. Prints a corresponding message whether the song is successfully removed or not. Enables users to manage the playlist by selectively removing songs, providing feedback on the deletion process.
  - **Shuffle Playlist (shuffle_playlist):** Shuffles the order of songs in the playlist using the random.shuffle function. Increments the shuffle counter and provides feedback on the successful shuffle. It introduces randomness to the playlist order, enhancing user experience. The shuffle counter helps track the number of shuffle operations.
  - **Display Playlist (display_playlist):** Checks if the playlist is empty and either displays the songs with their indices or prints a message indicating an empty playlist. Provides users with a clear view of the current playlist, adapting the display based on whether there are songs in the playlist or not.
  - **Display Shuffle Counter (display_shuffle_counter)**: Simply displays the number of shuffle operations that have occurred. It offers transparency to users by providing information on the number of times the playlist has been shuffled.

- **Discuss their time and space complexities.**

1. **Add songs (add_song):**
    - **Time Complexity:** O(1) The time complexity is constant because adding a song involves appending it to the end of the list, which is an O(1) operation.
    - **Space Complexity:** O(1) The space complexity is constant as well since it doesn't depend on the size of the playlist.
2. **Delete Song (delete_song):**
    - **Time Complexity:** O(n) The remove method in Python lists has a time complexity of O(n) in the worst case, where n is the length of the playlist. This is because it may need to shift elements after the removed one.
    - **Space Complexity:** O(1) The space complexity is constant as the operation doesn't require additional space proportional to the size of the playlist.
3. **Shuffle Playlist (shuffle_playlist):**
    - **Time Complexity:** O(n) The random.shuffle function has a time complexity of O(n), where n is the length of the playlist.
    - **Space Complexity:** O(1) The space complexity is constant as the operation is performed in-place on the existing playlist.
4. **Display Playlist (display_playlist):**
    - **Time Complexity:** O(n) The time complexity is linear, where n is the length of the playlist, as it involves iterating over each song to display them.
    - **Space Complexity:** O(1)The space complexity is constant as it doesn't use additional space proportional to the size of the playlist.
5. **Display Shuffle Counter (display_shuffle_counter)**
    - **Time Complexity**: O(1) Displaying the shuffle counter involves a constant-time operation.
    - **Space Complexity**: O(1) The space complexity is constant as it doesn't use additional space proportional to the size of the playlist.

4. Testing and Results

- **Explain the testing approach and methodology used.**

    - Since the project is intended to be run in a terminal environment, manual testing is likely a significant part of the testing approach. Testers or users can manually input commands through the terminal and observe the program's responses. This includes testing various menu options, adding and deleting songs, shuffling the playlist, and checking the shuffle counter.Given that the problem involves terminal-based user interactions, testing the clarity and correctness of the user interface is Important. Testers can assess whether the displayed menus, prompts, and messages are user-friendly and provide clear instructions.

- **Present test cases and their corresponding results.**

1. **Add songs in the playlist and its error message when you put blank in the playlist**

```
|--------------------------|
| 1. Add Song              |
| 2. Delete Song           |
| 3. Shuffle Playlist      |
| 4. Display Playlist      |
| 5. Display Shuffle Counter |
| 6. Exit                  |
|--------------------------|
Enter your choice: 1

_____________________________________
Enter the song name type 0 to stop: "live Well" - Palace
Enter the song name type 0 to stop: "One of the Girls" - The weeknd
Enter the song name type 0 to stop: "Sunset" - Andy Shauf
Enter the song name type 0 to stop: "The Opposite of Afternoon" - Unknown Mortal Orchestra
Enter the song name type 0 to stop: "In The Meantime" - Space Hog
Enter the song name type 0 to stop: 0
Songs have been added to the playlist.
```

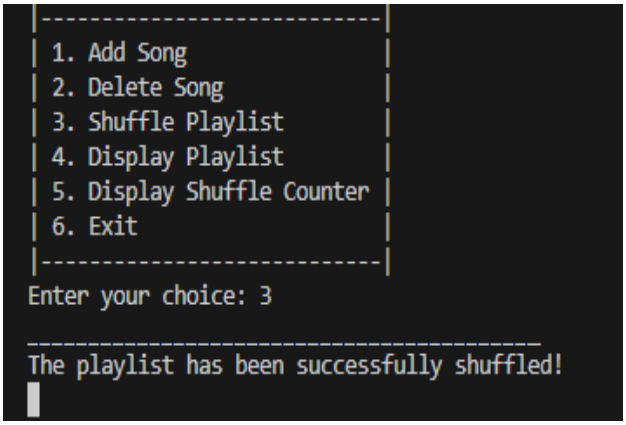
```
|--------------------------|
| 1. Add Song              |
| 2. Delete Song           |
| 3. Shuffle Playlist      |
| 4. Display Playlist      |
| 5. Display Shuffle Counter |
| 6. Exit                  |
|--------------------------|
Enter your choice: 1

_____________________________________
Enter the song name type 0 to stop:
Invalid Input!
```

2. **Deleting a song in the playlist and its error message when you delete a song that does not exist in the playlist**

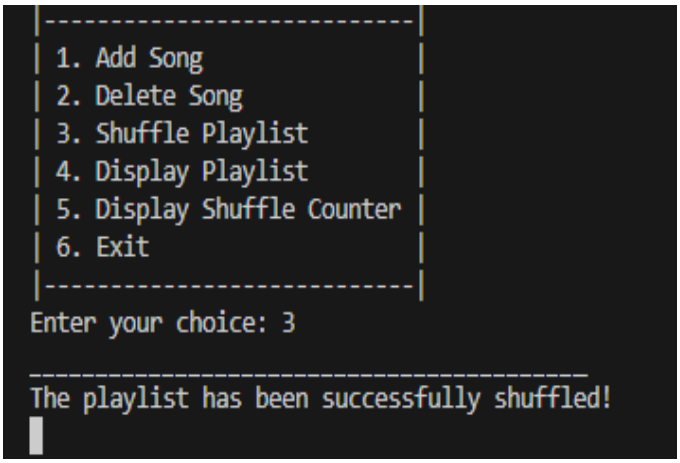
```
| 1. Add Song              |
| 2. Delete Song           |
| 3. Shuffle Playlist      |
| 4. Display Playlist      |
| 5. Display Shuffle Counter |
| 6. Exit                  |
|--------------------------|
Enter your choice: 2

_____________________________________
Enter the song name to delete: "Sunset" - Andy Shauf
"Sunset" - Andy Shauf has been removed from the playlist.
```

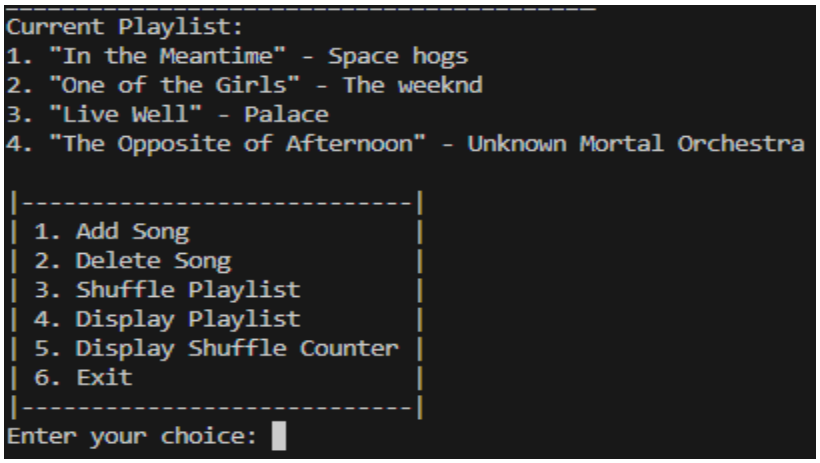
```
|--------------------------|
| 1. Add Song              |
| 2. Delete Song           |
| 3. Shuffle Playlist      |
| 4. Display Playlist      |
| 5. Display Shuffle Counter |
| 6. Exit                  |
|--------------------------|
Enter your choice: 2

_____________________________________
Enter the song name to delete: "Pusong bato" -  Ed Sheeran
"Pusong bato" -  Ed Sheeran is not in the playlist.
```

3. **Shuffle the Playlist and its message when the shuffling is successful**

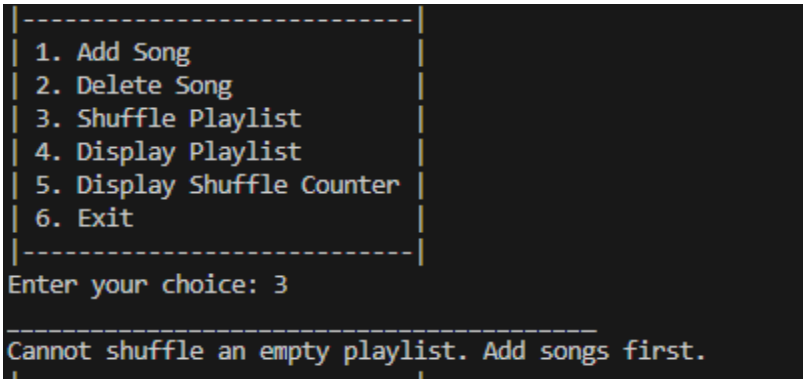


Error message when you shuffle an empty playlist:



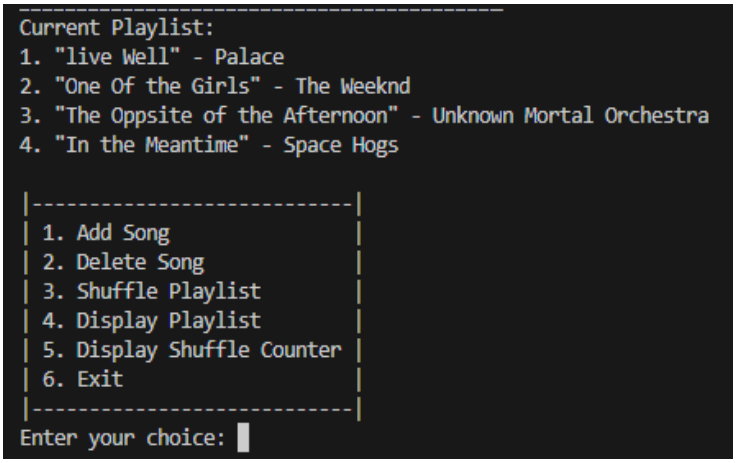


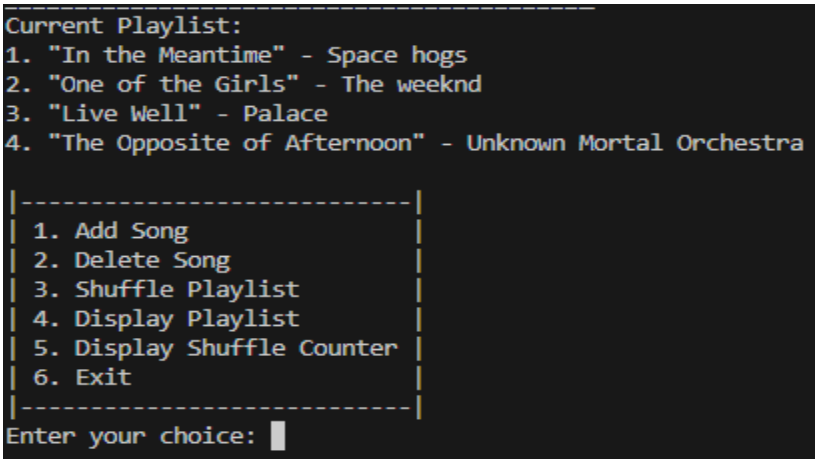
Message that indicating the playlist is successfully shuffled



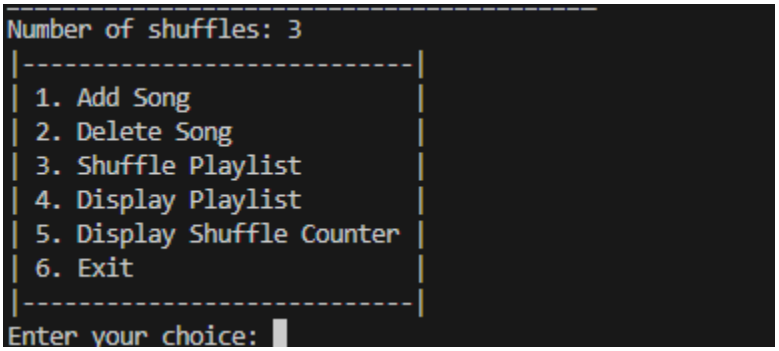
4. **Display playlist before shuffling and after shuffling**

Before **After**





5. **Display Shuffle Counter**



In this case we shuffled the playlist three (3) times to show that the shuffle counter is working properly.

Note: The terminal clears after 2.5 seconds when you add, delete, shuffle as well as after the error messages to improve the visual appeal and readability of the output program.

- **Analyze the results, discussing performance and correctness.**

  - The overall functionality testing of the program indicates that it works properly in terms of managing a music playlist. The key features, such as adding and deleting songs, shuffling the playlist, and displaying shuffle counter and everything worked as expected.The program did what it was supposed to do, gave the right feedback, and handled different situations correctly, even when there were mistakes. It also ran smoothly when doing things like shuffling songs. Overall, the testing tells us that the program is doing its job properly and is easy for users to use.

5. Discussion

- **Reflect on the project's strengths and weaknesses.**

Project Strengths:

The project has a user-friendly text-based menu, making it easy for users to manage their music playlist. It allows users to add, delete, shuffle songs, and view the playlist and shuffle counter. The program also handles errors, like preventing blank song names and notifying users if they try to shuffle an empty playlist. Messages are clear, and the display is easy to understand. Additionally, there's a delay and terminal clearing after specific operations, improving the overall visual appeal and readability of the console application.

Project Weaknesses:

The project doesn't save the playlist between uses, so when the program is closed, the playlist and shuffle counter reset. It's designed for a single user; handling multiple users or interactions might be important for real-world use. The program assumes each song has a unique name. If multiple songs share a name, deleting a specific one might not work correctly. Using unique identifiers or indices could help with this issue.

- **Discuss any challenges encountered and how they were addressed.**

1. **User Input Validation**:

   Challenges:

   - Users might input unexpected or invalid values, leading to errors or undesired behavior.

   How we address:

   - The program could incorporate robust input validation mechanisms. For example, using try-except blocks to catch errors, checking if inputs are within expected ranges, and providing clear error messages to guide users.

2. **Terminal is too messy**

   Challenges:

   - When we test the program, things get messy in the terminal after we type something. The menu and messages pile up, making it hard to read and use.

   How we address:

   - We added a clear_terminal function to fix this. Now, after you type something, the program cleans up the terminal, so it looks neat and organized for the next use. This makes testing and using the program easier for everyone.

- **Suggest potential improvements or extensions for future work.**

1. **Undo Feature:** Implement an undo feature that allows users to revert the last operation, such as undoing the deletion of a song or reverting a shuffle.
2. **Sorting Options**: Introduce sorting options for the playlist, allowing users to sort by song name, artist, or other criteria.
3. **Importing Real Songs:** Introduce a feature that allows users to import real songs into the program by providing file paths

6. Conclusion

- **Summarize the project's key findings and contributions.**

  - The music playlist shuffle program helps users manage their playlists by adding, deleting, shuffling songs and also the Shuffle counter. Key findings from testing show that the program works well, responding correctly to user inputs and maintaining a user-friendly interface. The implementation of the random shuffle algorithm ensures a random playlist shuffling experience.

- **Restate the significance of the implemented data structure(s) and algorithm(s) in solving the problem.**

  - The use of a list as the primary data structure in the music playlist shuffle program allows for flexible and dynamic management of songs. This simple structure enables easy addition, deletion, shuffling of songs, and the shuffle counter contributing to the effective organization of a playlist. The implementation of the Random Shuffle Algorithm ensures that the shuffling process is random and unbiased, enhancing the user experience by introducing variety to the playlist order. These chosen data structures and algorithms play an important role in efficiently solving the problem of managing a music playlist, providing users with a straightforward and functional tool for organizing and enjoying their collection of songs.

Music Playlist Shuffle:

- Create a playlist of songs using arrays or lists.
- Implement a random shuffle algorithm to reorder the playlist.
- Use a counter to track the number of shuffles and displays it upon request.

FINAL CODE:

```
import random
import os
import time

class MusicPlaylist:
    def __init__(self):
        # Initialize an instance of the MusicPlaylist class with an empty
playlist,
        # shuffle counter set to zero, and a flag to indicate if the
playlist is shuffled.
        self.playlist = []
        self.shuffle_counter = 0
        self.is_shuffled = False

    def add_song(self, song):
        #  to add a song to the playlist
        if song.strip():  # Check if the song name is not blank
            self.playlist.append(song)
        else:
            print("Invalid Input!")

    def delete_song(self, song):
        #  to delete a song from the playlist
        if song in self.playlist:
            # If the song is found in the playlist, remove it and display
a message.
            self.playlist.remove(song)
            print(f"{song} has been removed from the playlist.")
        else:
            # If the song is not in the playlist, display a message.
            print(f"{song} is not in the playlist.")
        time.sleep(2.5)  # Introduce a 2.5-second delay
        clear_terminal()  # Clear the terminal screen after deletion.
```

```
    def shuffle_playlist(self):
        #  to shuffle the playlist
        if not self.playlist:
            # If the playlist is empty, inform the user and clear the
terminal.
            print("Cannot shuffle an empty playlist. Add songs first.")
            time.sleep(2.5)  # Introduce a 2.5-second delay
            clear_terminal()
        else:
            # If the playlist is not empty, shuffle it using
random.shuffle.
            random.shuffle(self.playlist)
            self.shuffle_counter += 1
            self.is_shuffled = True
            # Display a success message, introduce a delay, and clear the
terminal.
            print("The playlist has been successfully shuffled!")
            time.sleep(2.5)  # Introduce a 2.5-second delay
            clear_terminal()

    def display_playlist(self):
        #  to display the current playlist
        if not self.playlist:
            # If the playlist is empty, inform the user, introduce a
delay, and clear the terminal.
            print("The playlist is empty.")
            time.sleep(2.5)  # Introduce a 2.5-second delay
            clear_terminal()
        else:
            # If the playlist is not empty, display each song with its
index.
            print("Current Playlist:")
            for i, song in enumerate(self.playlist, start=1):
                print(f"{i}. {song}")
        print()  # Print an extra line for better readability.

    def display_shuffle_counter(self):
        #  to display the number of shuffles
        print(f"Number of shuffles: {self.shuffle_counter}")

def clear_terminal():
    # Function to clear the terminal screen based on the operating system.
    if os.name == 'nt':  # For Windows
```

```
        os.system('cls')
    else:  # For Unix-like systems (Linux, macOS)
        os.system('clear')


if __name__ == "__main__":
    # Instantiate the MusicPlaylist class
    playlist = MusicPlaylist()

    while True:
        # Display the menu options to the user
        print("|----------------------------|")
        print("| 1. Add Song                |")
        print("| 2. Delete Song             |")
        print("| 3. Shuffle Playlist        |")
        print("| 4. Display Playlist        |")
        print("| 5. Display Shuffle Counter |")
        print("| 6. Exit                    |")
        print("|----------------------------|")

        # Prompt the user for their choice
        choice = input("Enter your choice: ")
        print("_____________________________________________")

        # Process the user's choice
        if choice == "1":
            # Allow the user to add songs until they enter '0' to stop
            while True:
                song_name = input("Enter the song name type 0 to stop: ")
                if song_name.lower() == "0":
                    break
                playlist.add_song(song_name)
            # Display a message and clear the terminal after adding songs.
            print("Songs have been added to the playlist.")
            time.sleep(2.5)  # Introduce a 2.5-second delay
            clear_terminal()
        elif choice == "2":
            # Prompt the user to enter the song name to delete and perform
the deletion.
            song_to_delete = input("Enter the song name to delete: ")
            playlist.delete_song(song_to_delete)
        elif choice == "3":
```

```
            # Shuffle the playlist, display it if not empty, and clear the
terminal.
            playlist.shuffle_playlist()
            if playlist.playlist:
                playlist.display_playlist()
        elif choice == "4":
            # Display the current playlist.
            playlist.display_playlist()
        elif choice == "5":
            # Display the number of shuffles.
            playlist.display_shuffle_counter()
        elif choice == "6":
            # Exit the program.
            break
        else:
            # Inform the user if an invalid choice is entered.
            print("Invalid choice")
```