Music Player

Loops - Extra

1. Shuffle Playlist

- **Task:** Write a function that shuffles the order of songs in a playlist. It should return a new array with the songs in a random order.
- **Inputs:** An array of song objects.
- **Example:**
  - Input:
    ```
    [{ name: "Song 1", artist: "Artist 1" }, { name: "Song 2", artist: "Artist 2" }, { name: "Song 3", artist: "Artist 3" }]
    ```
  - Output:
    ```
    [{ name: "Song 2", artist: "Artist 2" }, { name: "Song 1", artist: "Artist 1" }, { name: "Song 3", artist: "Artist 3" }]
    ```
- **Tip:** You might want to use the Fisher-Yates (aka Knuth) shuffle algorithm for optimal performance and a truly unbiased shuffle.

2. Most Frequent Artist

- **Task:** Write a function that determines which artist appears most frequently in a playlist. If there's a tie, return any of the top artists.
- **Inputs:** An array of song objects (each object includes an "artist" string).
- **Example:**
  - Input:
    ```
    [{ name: "Song 1", artist: "Artist 1" }, { name: "Song 2", artist: "Artist 2" }, { name: "Song 3", artist: "Artist 1" }]
    ```
  - Output: `"Artist 1"`
- **Tip:** Consider using a frequency counter pattern with a loop to determine which artist appears most frequently.

3. Longest Common Sequence of Songs

- **Task:** Write a function that determines the longest common sequence of songs in two different playlists. A sequence does not have to be consecutive but the order of the songs matters.

- **Inputs:** Two arrays of song objects.
- **Example:**
  - Input:
    ```
    playlist1 = [{ name: "Song 1", artist: "Artist 1" }, { name: "Song 2", artist: "Artist 2" }, { name: "Song 3", artist: "Artist 3" }]
    ```
    and
    ```
    playlist2 = [{ name: "Song 4", artist: "Artist 4" }, { name: "Song 2", artist: "Artist 2" }, { name: "Song 5", artist: "Artist 5" }, { name: "Song 3", artist: "Artist 3" }]
    ```
  - Output:
    ```
    [{ name: "Song 2", artist: "Artist 2" }, { name: "Song 3", artist: "Artist 3" }]
    ```
- **Tip:** This problem can be solved using a dynamic programming approach where a 2D array is filled with the lengths of common sequences found so far.

4. Generate Playlist from Genre

- **Task:** Write a function that generates a playlist of a given length with songs from a specific genre. Assume each song object has a "genre" property. The genre frequency should mirror its occurrence in the song pool as closely as possible.
- **Inputs:** An array of song objects, the desired playlist length (integer), and the genre (string).
- **Example:**
  - Input:
    ```
    [{ name: "Song 1", artist: "Artist 1", genre: "Rock" }, { name: "Song 2", artist: "Artist 2", genre: "Pop" }, { name: "Song 3", artist: "Artist 3", genre: "Rock" }]
    ```
    , playlist length: 2, genre: "Rock"
  - Output:
    ```
    [{ name: "Song 1", artist: "Artist 1", genre: "Rock" }, { name: "Song 3", artist: "Artist 3", genre: "Rock" }]
    ```
- **Tip:** This problem requires you to first filter songs by genre, and then implement a random selection algorithm to pick songs based on the given genre frequency.

5. Rate Playlist

- **Task:** Write a function that assigns a rating to a playlist. Assume each song has a rating property. The rating of a playlist is based on the average of the song ratings, but the highest and lowest rated songs only contribute half of their rating to the average.

- **Inputs:** An array of song objects (each object includes a "rating" integer).
- **Example:**
  - Input:
    ```
    [{ name: "Song 1", artist: "Artist 1", rating: 4 }, { name: "Song 2", artist: "Artist 2", rating: 5 }, { name: "Song 3", artist: "Artist 3", rating: 3 }]
    ```
  - Output: `3.75`
- **Tip:** Sort the songs by rating, treat the first and last songs differently when calculating the sum and the average. Remember to use floating-point numbers for accurate results.