CS101 Homework #4

Song Playlist Maker

Please read the homework description and follow the instructions carefully. Please be aware that this homework is **an individual task**; you can discuss the problem with your friends, but you MUST NOT implement your ideas together. **You will fail the entire course (that is, your CS101 grade will be an F) if you are found to be involved in any attempts of plagiarism, including using AI-sourced code (e.g., ChatGPT).**

Overview

In this homework, you will read and process a csv file that contains Top 600 Songs and their relevant information from 2010-2019 from Spotify. We have simplified and modified the original dataset, and you can refer to other examples of the Spotify song data (example) if you wish. The csv begins with a header consisting of column names, explained below in detail:

1. *title*: Title of the song
2. *artist*: Artist of the song
3. *top_genre*: Genre of the track
4. *year*: Song's year on the Billboard
5. *bpm*: Tempo of the song (in Beats Per Minute)
6. *energy*: Energy of the song (higher means more energetic)
7. *danceability*: Danceability of the song (higher means the song is easier to dance to)
8. *db*: Loudness of the song (higher means louder)
9. *valence*: Positivity of the song (higher means happier and more positive)

Here, we provide examples directly extracted from the data.

```
(...)
“Hey, Soul Sister”, “Train”, “neo mellow”, 2010, 97, 89, 67, -4, 80
“Someone Like You”, “Adele”, “british soul”, 2010, 135, 33, 56, -8, 28
(...)
```

You should write a program that reads the csv file **[Task 1]**, stores the data in a well-organized object form **[Task 2]**, creates a playlist of the song objects **[Task 3]**, and calculates the mood to create a mood-based playlist **[Task 4]**.

Here's an overview of the score breakdown for this homework:

Task 1 [5 pts] (Page 3)
- Task 1.1 [2 pts]
- Task 1.2 [3 pts]

Task 2 [10 pts] (Page 4~5)
- Task 2.1 [2 pts]
- Task 2.2 [4 pts]
- Task 2.3 [4 pts]

Task 3 [15 pts] (Page 6~7)
- Task 3.1 [5 pts]
- Task 3.2 [5 pts]
- Task 3.3 [5 pts]

Task 4 [20 pts] (Page 8~9)
- Task 4.1 [6 pts]
- Task 4.2 [4 pts]
- Task 4.3 [10 pts]

You can use the Python csv module in this homework. Please refer to the following link to understand how to use the module: https://docs.python.org/3/library/csv.html.

Task 1: Parse Song Data [5 pts]

In this task, you will read, process, and return specific information from the provided csv file.

Task 1.1 [2 pts]: Implement the `get_columns_in_csv(file_path)` method to obtain all the names of all the columns from the header.

```
Arg:
     file_path (str): Path to the target csv file.
Return:
     columns (list[str]): Name of all the columns.
```

Task 1.2 [3 pts]: Implement the `get_song_info_of_nth_row(file_path, n)` method to obtain the song title, artist, and year written in the *n*-th row.

```
Arg:
     file_path (str): Path to the target csv file.
     n (int): Index of the row to get the song information.
     The index begins at 1, where the counting excludes the
     header.
Return:
     song_info (Tuple(str, str, int)): Tuple of song title,
     artist, and year in the n-th row.
Exception:
     If n is not in a valid range of row numbers (1~600),
     return the string 'No song found in this row'
```

[Example Run]

```
print(get_columns_in_csv('top_songs.csv'))
>>> ['title', 'artist', 'top_genre', 'year', 'bpm', 'energy',
'danceability', 'db', 'valence']
print(get_song_info_of_nth_row('top_songs.csv', 1))
>>> ("Hey, Soul Sister", "Train", 2010)
print(get_song_info_of_nth_row('top_songs.csv', -200))
>>> 'No song found in this row'
```

Task 2: Make Song object [10 pts]

In this task, you have to define the constructor of the `Song` object created from each row of the csv file. **Figure 1** shows the attributes of the `Song` object.

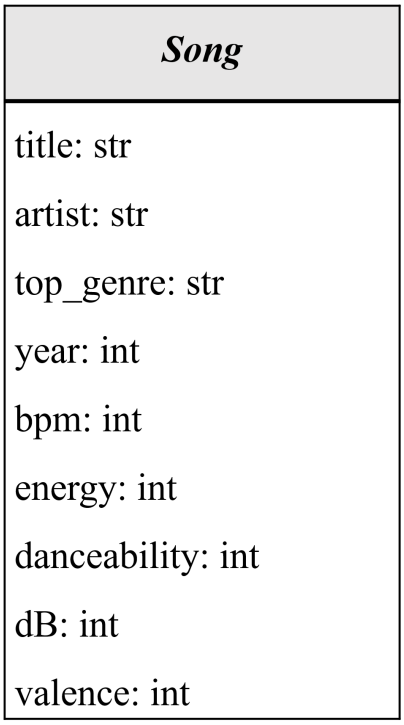

Figure 1. class `Song` specification.

Task 2.1 [2 pts]: Implement the constructor of the `Song` object `__init__(self, title, artist, top_genre, year, bpm, energy, danceability, dB, valence)` to create the `Song` object. `info` is a tuple that consists of detailed information of a song. You may refer to the details about the `info` argument described below. **Note that every attribute must be specified in your implementation. You cannot get any score even if you miss one of the attributes described in Figure 1.**

```
Arg:
    title (str), artist (str), top_genre (str), year (int),
    bpm (int), energy (int), danceability (int), dB (int),
    valence (int)
```

Task 2.2 [2pts]: Implement the get_song_of_nth_row(n) function that extracts the information of a song in n-th row of the csv file and returns corresponding `Song` object. You may copy-paste your codes in Task 1.2 and modify it to implement `get_song_of_nth_row`.

```
Arg:
    file_path (str): Path to the target csv file.
    n (int): Index of the row to get the song information.
    The index begins at 1, where the counting excludes the
    header.
Return:
    song_obj (Song): Song object corresponds to the n-th
    row of the csv file.
Exception:
    If n is not in a valid range of row numbers (1~600),
    return the string 'No song found in this row'
```

Task 2.3 [3 pts]: Implement `__str__(self)` that returns a brief description about the `Song` object. The description must have a type of string. **Note that you must keep the format of a string as described below**.

```
Return:
    description (string): a brief description about the
    Song object which has a format of "{title}: a(n)
    {top_genre} song released in {released_year} by
    {artist}."
```

Task 2.4 [3 pts]: Implement `__eq__(self, another)` that indicates equality between two `Song` objects. The `Song` objects are defined to be equal **if and only if they have the same title, artist, genre, and released year.**

```
Arg:
    another (Song): another Song object to be compared.
Return:
    equality (bool): True if two objects have the same
    title, artist, genre, and released year, False
    otherwise.
```

[Example Run]

```
song_tuple1 = ("Only Girl (In the World)", "Rihanna", "barbadian
pop", 2010, 126, 72, 79, -4, 61)
```

```
song_tuple2 = (“Take It Off”, “Kesha”, “dance pop”, 2010. 120.
61. 83. -4, 76)
song_rihanna, song_kesha = Song(song_tuple1), Song(song_tuple2)
print(song_kesha)
```
>>> “Your Love Is My Drug: a(n) dance pop song released in 2010 by Kesha.”
```
song_rihanna == song_kesha
```
>>> False
```
song_rihanna == song_rihanna
```
>>> True

Task 3: Connection of two classes [15 pts]

In this task, you have to define a `PlayList` object which can contain one or more `Song` objects, and simple class member methods dealing with them. As described in **Figure 2**, `Songs`, which is a member of PlayList object, is a list of `Song` object(s).

Figure 2. class `PlayList` specification.

Task 3.1 [5 pts]: Implement `__init__(self, song_list)` that initializes `PlayList` object.

```
Arg:
     song_list (List(Song)): a list of Song objects which
     will make up a single PlayList.
```

Task 3.2 [5 pts]: Implement `genre_set(self)` which analyzes the genre of `Song` objects contained in `PlayList` and represents them as a list without any duplicates. For example, given a `PlayList` object that contains two `'detroit hip hop'` songs and one `'pop'` song, `genre_set` will return `['detroit hip hop', 'pop']`. **Note that the order of `genres` will not affect the score**.

```
Return:
     genres (List(string)): the list of genres without
     duplicates.
```

Task 3.3 [5 pts]: Implement `most_energetic_genre(self)` that classifies the `Song` objects by genre, and returns the genre with the highest sum of energies. In this task, `genre_set(self)` in Task 3.2 will be helpful for your implementation.

```
Return:
     most_energetic_genre (string): the genre with the
     highest sum of energies in the PlayList.
```

[Example Run]

```
song_list = []
for i in range(10):
     song_elem = Song(get_song_of_nth_row(i))
     song_list.append(song_elem)
plays = PlayList(song_list)
plays.genre_set()
>>> [‘neo mellow’, ‘detroit hip hop’, ‘dance pop’, ‘canadian pop’, ‘barbadian pop’]
plays.most_energetic_genre()
>>> ‘dance pop’
```

Task 4: Mood-based Playlist Creation [20 pts]

In this task, you will create the mood-based `PlayList` objects that searches the song list for songs that fit the mood requested by the user. The mood of each song is calculated with the four characteristics of the songs (`energy, danceability, db,` and `valence`) given by Spotify.

Task 4.1 [6 pts]: Implement `calculate_song_mood(self)` that calculates the mood of the `Song` object. The four possible moods are: **sad, angry, happy,** and **calm. Note that each song can only have one mood.**

The mood is calculated with the following equation:

$$score = [((energy + dance + valence) * 0.01) - (db/60)]/4$$

The following ranges indicate the mood of each song:

$$Sad : 0.0 <= score < 0.30$$
$$Angry : 0.30 <= score < 0.45$$
$$Happy : 0.45 <= score < 0.60$$
$$Calm : 0.60 <= score <= 1.0$$

```
Return:
    song_mood (Tuple(str, float)): The calculated mood of
    the song in the form of a Tuple of (mood, score). Note
    that the mood must be in lower case.
Exception:
    If the final score of the song is less than 0.0 or
    greater than 1.0, return ('None', -1.0).
```

Task 4.2 [4 pts]: Implement `create_playlist_by_mood(song_list, mood)` that returns a list of all songs that fits the given mood. **Note that the final list of the songs must be in the order of the original file.** In this task, `calculate_song_mood` in Task 4.1 will be helpful for your implementation.

```
Arg:
    song_list (List[Song]): a list of Songs.
```

```
        mood (str): one of four possible (sad, angry, happy,
        calm) moods given
    Return:
        mood_playlist (PlayList): The PlayList object
        containing all songs from the given list that fits the
        given mood. Note that the final list of the songs must
        be in the order of the original file.
```

Task 4.3 [10 pts]: Implement `calculate_average_bpm_by_mood(song_list)` that calculates the average bpm of all the songs in each mood. Note that songs that fall under 'None' mood do not have to be calculated. **Also, note that the list should be in the descending order of the highest to the lowest bpm.**

```
    Arg:
        song_list (List[Song]): a list of Songs.
    Return:
        bpm_by_mood (List[Tuple(str, float)]): A list of Tuple
        objects that contain (mood, average_bpm) for each of
        the four moods. Note that the list should be in the
        order of the highest to lowest bpm.
```

[Example Run]

```
song_list = []
for i in range(10):
    song_elem = Song(get_song_of_nth_row(i))
    song_list.append(song_elem)
print(song_list[0].calculate_song_mood())
>>> ('calm', 0.6066666666666667)
mood_playlist = create_playlist_by_mood(song_list, 'calm')
print(mood_playlist.songs[0])
>>> Hey, Soul Sister: a(n) neo mellow song released in 2010 by
Train.
calculate_average_bpm_by_mood(song_list)
>>> [('angry', 148.0), ('happy', 107.8), ('calm',
101.33333333333333), ('sad', 93.0)]
```