CSE 101: Computer Science Principles (Fall 2019)

Final Programming Project

Assignment Due: Thursday, December 12, 2019, by 11:59 PM

Getting Started

This assignment requires you to write Python code to solve several computational problems. To help you get started, we will give you a basic starter file for each problem with a few tests you can use to verify that your code seems to be correct (**note that these test cases are just examples; we will use different test inputs to grade your work!**). For each problem, complete its function as directed in the assignment. **Do not, under any circumstances, change the names of the functions or their parameter lists.** The automated grading system will be looking for functions with those exact names and parameter lists; if you change any function headers, the grading program will reject your solution and mark it as incorrect.

Directions

Each part of this assignment describes a single function that makes up part of the final program. For simplicity, all of these functions are located in a single starter file that combines them into a single, cohesive program. You will need to place copies of your completed `.py` files from Homework 2 (as well as your data files for testing) in the same folder/directory as this file, since its functions rely on that code in order to work. We have already set up the starter file for the project to correctly link everything together.

As with the lab assignments, our automated grading system will execute your solution to each part several times, using different input values each time. Each test that produces the correct/expected output will earn 1 or more points. This assignment contains 3 problems, and is worth a total of 60 points. **Note that not every problem may be worth the same number of points!**

- ⚠ The starter file has a comment block at the top with various important information. Be sure to add your information (name, ID number, and NetID) to the first three comment lines, so that we can easily identify your work. **If you leave this information out, you may not receive credit for your work!**
- ⚠ **DO NOT** zip or otherwise compress your file before you submit it. **If you do so, you will receive a 0 for this assignment!**
- ⚠ Each of your functions **MUST** use the names and parameter lists indicated in the starter code file. Submissions that have the wrong function names (or whose functions contain the wrong number of parameters) can't be graded by our automated grading system, and may receive a grading penalty (or may not be graded at all).
- ⚠ Each of your functions must explicitly ***return*** its final answer; the grading program ignores anything printed by your code. **DO NOT** use `input()` anywhere before the `if __name__ == "__main__":` statement in your files; every function will receive any data it needs from its parameters. Programs that crash will likely receive a failing grade, so test your code **thoroughly** before submitting it.
- ⚠ Blackboard will provide information on how to submit this assignment. You **MUST** submit your completed work as directed by the indicated due date and time. We will not accept (or grade) any work that is submitted after the due date and time, or that is submitted before you have signed the course's Academic Honesty agreement.
- ⚠ **ALL** of the solution code that you submit **MUST** be your own work! You may not receive assistance from or share any materials with anyone else, except for the instructor and the (current) course TAs.

Assignment: Creating Custom Musical Playlists (Part 2)

Suppose that you have a side business working as a DJ for parties and other events. Every event runs for a different length of time and has a different clientele, so you need a way to quickly generate a customized playlist of songs from your music library. In this assignment, you will extend your work from Homework 2 to generate randomized playlists based on different sets of criteria.

Sample output is provided at the end of this document. More test data files for each function will be posted shortly. Note that your specific results may (and probably will) vary due to the output of the random number generator.

Part I: Building a Basic Playlist (20 points)

Function Summary

Parameters: A dictionary in the format produced by `createLibrary()`, a list of strings, and a non-negative integer.

Return Type: A list of 2-element tuples. Each tuple contains two strings (a song title and its performer/artist).

Complete the `buildPlayList()` function, which takes three arguments, in the following order:

1. a dictionary of songs grouped by musical genre (like the one produced by `createLibrary()`)
2. a list of strings representing one or more musical genres
3. an integer *n* representing the total number of songs to select for the resulting playlist

This function returns a list of exactly *n* 2-element tuples (each tuple consists of a song title, followed by the artist name), randomly drawn from the genres specified above ***(and ONLY those genres)***. For example, if the second argument was the three-element list ["hip-hop", "classical", "metal"], the new list should only contain songs from those three genres (i.e., the result should not contain any songs that belong to a different genre like "country" or "reggae").

- If one or more of the requested genres do not exist in the library, your function should ignore that genre (for example, if the library had no "classical" songs in the example above, then the function should only consider songs from the "hip-hop" and "metal" genres).
- If the library does not contain **ANY** of the requested genres, then the function should return an empty list.
- If *n* is greater than the total number of songs that are available in all of the requested (and available) genres combined, the function should return a list with all of the songs in those genres combined, in any order. For example, if the user requests a 20-song playlist drawn from four genres that have 2, 5, 4, and 6 songs respectively, the function should simply return a list with all 17 available songs.

Start by constructing a list of potential songs from each of the specified genres. Then use a function like `random.sample()` (from the `random` module — see the slides for an example of how to use this function) to construct your playlist.

Part II: Building a Playlist by Song Length (20 points)

Function Summary

Parameters: A dictionary in the format produced by `createLibrary()`, a list of strings, and a non-negative integer.

Return Type: A list of 2-element tuples. Each tuple contains two strings (a song title and its performer/artist).

Complete the `buildPlayListByLength()` function, which takes three arguments, in the following order:

1. a dictionary of songs grouped by musical genre (like the one produced by `createLibrary()`)
2. a list of strings representing one or more musical genres
3. an integer *n* representing the ***minimum required length*** of the playlist in seconds, including a 2-second gap between each song in the resulting playlist

This function returns a list of exactly *n* 2-element tuples (each tuple consists of a song title, followed by the artist name), randomly drawn from the genres specified above ***(and ONLY those genres)***. The total length of the songs in the playlist (measured in seconds), including a 2-second gap between each pair of songs (don't add the gap after the final song in the playlist), must be greater than or equal to the third function argument, but must not exceed it by more than one song. For example, `buildPlayListByLength(song_library, ["hip-hop", "classical", "metal"], 368)` would create a playlist of songs from those three genres that was at least 368 seconds long (including the 2-second gaps).

- If one or more of the requested genres do not exist in the library, your function should ignore that genre (for example, if the library had no "classical" songs in the example above, then the function should only consider songs from the "hip-hop" and "metal" genres).
- If the library does not contain **ANY** of the requested genres, then the function should return an empty list to signify failure.

Start by constructing a list of every song from all of the specified genres. Then use a function like `random.sample()` to create a new version of the original list, only in random order.

Use a loop to add up the lengths of all of the songs in your randomized list, adding an extra 2 seconds for each song except the last one. If this value is less than the required playing time, return an empty list to signify failure.

Finally, create a list of songs — represented as two-element tuples in (song title, artist) order — selected from your list of randomly-ordered songs. Starting with an empty list, use a `while` loop to add songs, in order, to your playlist, stopping as soon as the total length meets or exceeds the required length. This will require two additional variables: the current index in your list of all songs (which starts at 0), and the current total playing time of the playlist that you are creating (this will be your loop variable).

Each time the loop executes:

1. If your playlist is empty, add the length of the next song in the source list to the total playing time, and then append the song information (title and artist) to your playlist.
2. If your playlist is **not** empty, follow the directions in the previous step but also add 2 to the total playing time.

When the loop ends, return your playlist.

Part III: Building a Playlist by Rating (20 points)

Function Summary

Parameters: A dictionary in the format produced by `createLibrary()`, a list of strings, and a non-negative integer in the range 0–5, inclusive.

Return Type: A list of 2-element tuples. Each tuple contains two strings (a song title and its performer/artist).

Complete the `buildPlayListByRating()` function, which takes three arguments, in the following order:

1. a dictionary of songs grouped by musical genre (like the one produced by `createLibrary()`)
2. a list of strings representing one or more musical genres
3. an integer *n* in the range 0–5 inclusive, representing the minimum song rating that any song in the playlist can have

This function returns a list of exactly *n* 2-element tuples (each tuple consists of a song title, followed by the artist name), randomly drawn from the genres specified above ***(and ONLY those genres)***. For example, if the second argument was the three-element list ["hip-hop", "classical", "metal"], the new list should only contain songs from those three genres (i.e., the result should not contain any songs that belong to a different genre like "country" or "reggae"). Each song's rating **MUST** be greater than or equal to the third parameter. For example, `buildPlayListByRating(song_library, ["hip-hop", "classical", "metal"], 3)` would create a playlist of songs from those three genres where each song had a rating of 3, 4, or 5.

- If one or more of the requested genres do not exist in the library, your function should ignore that genre (for example, if the library had no "classical" songs in the example above, then the function should only consider songs from the "hip-hop" and "metal" genres).
- If the library does not contain **ANY** of the requested genres, then the function should return an empty list to signify failure.

Start by constructing a list of every song from all of the specified genres. Then use a loop to create a new list: for each song in the original list, if its rating is greater than or equal to the required minimum, add that song to the new list. If the length of this new list is 0, return an empty list to signify failure. Otherwise, use `random.sample()` to create and return a randomly-shuffled copy of the new list.

Sample Output

```
createLibrary("library1.txt") returns the dictionary:

{'childrens': [['Be Our Guest', 'Angela Lansbury', 224, 0], ['Lullabye', 'Billy Joel',
 213, 0]], 'dance': [['Happy Now', 'Kygo', 211, 0], ['Grapevine', 'Tiesto', 150, 0],
 ['Headspace', 'Dee Montero', 271, 0]], 'blues': [['Dream of Nothing', 'Bob Margolin',
 208, 3], ['Rock and Stick', 'Boz Scaggs', 290, 0], ['At Last', 'Etta James', 181, 0],
 ["You're Driving Me Crazy", 'Van Morrison', 286, 1]], 'kpop': [['Not That Type',
 'gugudan', 191, 3], ['IDOL', 'BTS', 222, 2], ['Believe Me', 'Seo In Young', 191, 0],
 ['Baam', 'MOMOLAND', 208, 1], ['Hide Out', 'Sultan of the Disco', 257, 0]]}


createLibrary("library2.txt") returns the dictionary:

{'jazz': [['Frankenstein', 'Christian Sands', 523, 0], ['Lebroba', 'Andrew Cyrille',
 344, 0]], 'classical': [['Perfect', 'The Piano Guys', 310, 0], ['To Where You Are',
 'Katherine Jenkins', 226, 0]], 'country': [['Chrome', 'Trace Adkins', 203, 0],
 ['Nothing I Can Do About It Now', 'Willie Nelson', 198, 0],
 ["Should've Been a Cowboy", 'Toby Keith', 302, 0], ['Better Than You', 'Terri Clark',
 242, 0]]}


createLibrary("library3.txt") returns the dictionary:

{'metal': [['Call The Man', 'Pentagram', 229, 0], ['Brimstone', 'Whitechapel', 205, 0],
 ['Lightning Strike', 'Judas Priest', 209, 0], ['Enter Sandman', 'Metallica', 331, 0]],
 'reggae': [['Sidung', 'Kranium', 200, 0], ['Naked Truth', 'Sean Paul', 215, 0],
 ['Red Red Wine', 'UB40', 320, 0]], 'hiphop': [['Spaceship Odyssey 2000',
 'Malik Yusef', 276, 0], ["It's Tricky", 'Run-DMC', 183, 0], ['Cuttin Rhythms',
 'Tone Loc', 311, 0]]}


createLibrary("library4.txt") returns the dictionary:

{'electronic': [['Fast Life', 'Jackson and His Computer Band', 308, 0],
 ['Stairway to Heaven', 'Jana', 245, 0], ['Silikon', 'Modeselektor', 226, 0]],
 'rock': [["School's Out", 'Alice Cooper', 210, 0], ['My Reply', 'The Ataris', 254,
 0], ['Burning Bridges', 'Bon Jovi', 164, 0], ["Escher's Staircase", 'Lana Lane',
 366, 0]], 'alternative': [['Psycho Killer', 'Talking Heads', 260, 0],
 ['So Far So Good', 'Thornley', 202, 0], ['Nothing At All', 'Wired All Wrong', 198,
  0], ['Chocolate', 'The 1975', 227, 0]]}


buildPlayList() based on "library2.txt" using the genres ["electronic","rock"] and
 a length of 12 returns:
[]
```

```
buildPlayList() based on "library4.txt" using the genres ["electronic","rock"] and a
 length of 4 returns:
[('Fast Life', 'Jackson and His Computer Band'), ('Silikon', 'Modeselektor'),
 ('Stairway to Heaven', 'Jana'), ("Escher's Staircase", 'Lana Lane')]

buildPlayListByLength() based on "library4.txt" using the genres ["electronic","rock"]
 and a time of 800 returns:
[('Fast Life', 'Jackson and His Computer Band'), ('Stairway to Heaven', 'Jana'),
 ('Silikon', 'Modeselektor'), ("School's Out", 'Alice Cooper')]

buildPlayListByLength() based on "library2.txt" using the genres ["jazz"] and a time
of 2000 returns:
[]

buildPlayListByLength() based on "library2.txt" using the genres ["jazz","classical"]
and a time of 1000 returns:
[('Frankenstein', 'Christian Sands'), ('Lebroba', 'Andrew Cyrille'), ('Perfect',
 'The Piano Guys')]

buildPlayListByRating() based on "library1.txt" using the genres ["blues","kpop"]
and a rating of 4 returns:
[]

buildPlayListByRating() based on "library1.txt" using the genres ["blues","kpop"]
and a rating of 2 returns:
[('Dream of Nothing', 'Bob Margolin'), ('Not That Type', 'gugudan'), ('IDOL', 'BTS')]
```