NYU Tandon School of Engineering

CS-UY 1114 Fall 2022

Homework 09

Due: 11:59pm, Thursday, December 1st, 2022

Submission instructions

1. You should submit your homework on **Gradescope**.
2. For this assignment you should turn in 1 `.py` files named according to the following pattern: `hw9_q1.py`.
3. Your Python file should contain a header comment block as follows:

```
"""
Author: [Your name here]
Assignment / Part: HW9 - Q1
Date due: 2022-12-01, 11:59pm
I pledge that I have completed this assignment without
collaborating with anyone else, in conformance with the
NYU School of Engineering Policies and Procedures on
Academic Misconduct.
"""
```

No late submissions will be accepted.

REMINDER: *Do not use any Python structures that we have not learned in class.*

The use of `eval()` and `break` are no longer permitted in this class.

For this specific assignment, you may use everything we have learned up to, **and including**, file i/o. Please reach out to us if you're at all unsure about any instruction or whether a Python structure is or is not allowed.

Do **not** use, for example, object-oriented programming.

Note: All functions in this homework assignment must be implemented in the same file, `hw9_q1.py`.

Problems

1. **Happiness Is A File's Code** `hw9_q1.py`
    1. **Background**
    2. `get_chord_dictionary()`
    3. `get_possible_chords()`
    4. `get_chord_progression()`
    5. `write_progression_file()`
2. **The 2nd Students' Choice CA Awards**

Happiness Is A File's Code

Background

Music streaming companies such as Spotify and Apple Music are able to generate curated recommendations to their users by analysing the music that those users listen to daily. Incredibly, one of the metrics that they use to measure how "recommendable" a song is to a user is by comparing a song's **chord progression** to those of the user's favourite songs. To give a simple example, if the user listens to a lot of songs with the following chord progression:

```
C major -> E dominant 7th -> F major -> F minor
```

Then other songs that follow a similar chord progression are more likely to get recommended to this user.

Let's use this premise as a use case for our next problem where we are given a file (which you can download **here**) containing a song's chord progression broken down into individual notes:

```
A,C#,E,F#,A
G#,D#,F#,C#,F,C,A#
C#,E,G#
E,G#,B
A,C#,E,F#,A
```

Figure 1: That is, for example, the first chord is comprised of the notes A (which appears twice), C#, E, and F#.

We would like to be able to read these lines and create a new file that contains the names of the chords that they could potentially form:

```
A6,A13
G#13
C#m
Emaj,E6,E9,E13
A6,A13
```

Figure 2: That is, for example, the notes A, C#, E, and F# can be potentially part of the chords A^6 and A^{13}.

This file, in other words, is going to be your final output.

Even if you don't know a lick of music theory, programmers who work in these companies are given supplementary files containing information about chords and the notes that make them up. Go ahead download **this** file–we'll be using it to help us out throughout this problem.

`get_chord_dictionary()`

(sig: `str -> dict`)

Let's say your manager provides you with a file that contains the notes that each chord contains:

```
Root,Chord Name,Notes
Ab,maj,G#-C-Eb
```

```
Ab,m,G#–B–Eb
Ab,6,G#–C–Eb–F
Ab,9,G#–C–Eb–F#–Bb
Ab,dim,G#–B–D
Ab,13,G#–C–Eb–F#–Bb–C#–F
A,maj,A–C#–E
...
G#,dim,G#–B–D
G#,13,G#–C–Eb–F#–Bb–C#–F
```

Figure 3: A CSV file containing the names and notes of several chords for any given root note (key). That is, the A-flat (`Ab`) major (`maj`) chord contains the notes `G#`, `C`, and `Eb`.

Our first step is to turn the contents into something we can work with in Python code. We would like to extract the data from this file and turn it into a dictionary of the following format:

```
{
    'A': {
        '13':  ('A', 'C#', 'E', 'G', 'B', 'D', 'F#'),
        '6':   ('A', 'C#', 'E', 'F#'),
        '9':   ('A', 'C#', 'E', 'G', 'B'),
        'dim': ('A', 'C', 'Eb'),
        'm':   ('A', 'C', 'E'),
        'maj': ('A', 'C#', 'E')
    },
    'A#': {
        '13':  ('Bb', 'D', 'F', 'G#', 'C', 'Eb', 'G'),
        '6':   ('Bb', 'D', 'F', 'G'),
        '9':   ('Bb', 'D', 'F', 'G#', 'C'),
        'dim': ('Bb', 'C#', 'E'),
        'm':   ('Bb', 'C#', 'F'),
        'maj': ('Bb', 'D', 'F')
    },
    'Ab': {
        '13':  ('G#', 'C', 'Eb', 'F#', 'Bb', 'C#', 'F'),
        '6':   ('G#', 'C', 'Eb', 'F'),
        '9':   ('G#', 'C', 'Eb', 'F#', 'Bb'),
        'dim': ('G#', 'B', 'D'),
        'm':   ('G#', 'B', 'Eb'),
        'maj': ('G#', 'C', 'Eb')
    },
    ...,
    'Gb': {
        '13':  ('F#', 'Bb', 'C#', 'E', 'G#', 'B', 'Eb'),
        '6':   ('F#', 'Bb', 'C#', 'Eb'),
        '9':   ('F#', 'Bb', 'C#', 'E', 'G#'),
        'dim': ('F#', 'A', 'C'),
        'm':   ('F#', 'A', 'C#'),
        'maj': ('F#', 'Bb', 'C#')
    }
}
```

Figure 4: Our chord dataset from figure 3 turned into a chord dictionary.

Write a function, `get_chord_dictionary()`, that accepts the chord dataset's filepath (a string) as its only parameter and returns a dictionary similar to the one in figure 4. If implemented correctly, you should observe the following behaviour:

```
from pprint import pprint  # for nice formatting of dictionary output

chord_dictionary = get_chord_dictionary("chords.csv")  # assuming this file is
in your working directory

# "Give me all of the CHORDS we have available in the key of Eb (E-flat)"
pprint(chord_dictionary["Eb"])

# "Give me all of the NOTES that make up the F diminished chord (Fdim)"
print(chord_dictionary["F"]["dim"])

# "Give me the 0th note from the notes that make up a C sharp thirteenth chord
(C#13)"
print(chord_dictionary["C#"]["13"][0])
```

Output:

```
{'13': ('Eb', 'G', 'Bb', 'C#', 'F', 'G#', 'C'),
 '6': ('Eb', 'G', 'Bb', 'C'),
 '9': ('Eb', 'G', 'Bb', 'C#', 'F'),
 'dim': ('Eb', 'F#', 'A'),
 'm': ('Eb', 'F#', 'Bb'),
 'maj': ('Eb', 'G', 'Bb')}
('F', 'G#', 'B')
C#
```

Notes:

- You may not assume that the chords dataset CSV file will exist–your function must check for its existence before attempting to read from it. If, for whatever reason, the file is not found, your function should return an empty dictionary instead.
- You may assume that the integrity of the data inside of the dataset file will always be solid i.e. each data point will always be either a note name or a chord name.
- While the contents of the file can be different depending on the file that your boss gives you, you can assume that it will always contain a header line.

`get_possible_chords()`

(sig: `list, dict -> tuple`)

Okay, so now that we have our chord dictionary, we would like to have a function that determines whether a list of notes can form any of the chords included in it. For example, if we assume that the first note represents the root

of the chord, the following list of notes:

```
E, B, G#
```

Can form the chords Emaj, E6, E9, and E13, since all four of these chords contain those three notes:

```
Emaj -> E, G#, B
E6   -> E, G#, B, C#
E9   -> E, G#, B, D, F#
E13  -> E, G#, B, D, F#, A, C#
```

Write a function `get_possible_chords()` that does just this: accept a list of notes (string objects) and a chord dictionary like the one returned by `get_chord_dictionary()` and returns a **tuple** of chords from the chord dictionary that this list of notes could satisfy. For example:

```
chord_dictionary = get_chord_dictionary(chords_filepath)
print(get_possible_chords(['E', 'B', 'G#'], chord_dictionary))
```

Output:

```
('Emaj', 'E6', 'E9', 'E13')
```

You can assume that the first note in your list of notes is the root note (the key) and that all notes in the list will exist in the chord dictionary.

`get_chord_progression()`

(sig: `str, str -> list`)

Let's put our two functions from above together. Write a function `get_chord_progression()` that accepts two strings:

- The first string represents the address for a CSV file that contains the chord progression of a song broken down into notes (see figure 1).
- The second string represents the address for the CSV file we'll use to create our chord dictionary (see figure 3).

Using these two files and our functions `get_chord_dictionary()` and `get_possible_chords()`, `get_chord_progression()` must return a **list of tuples**, one tuple per chord in the chord progression file, where each tuple contains the chords from the chord dictionary that those notes could satisfy.

For example, if we use the files from figures 1 and 3:

```
chord_progression = get_chord_progression("chord-progression.csv",
"chords.csv")
print(chord_progression)
```

Output:

```
[('A6', 'A13'), ('G#13'), ('C#m'), ('Emaj', 'E6', 'E9', 'E13'), ('A6', 'A13')]
```

Notes:

- You may not assume that the chord progression file exists. If it doesn't, `get_chord_progression()` must return an empty list.
- You *must* use `get_chord_dictionary()` and `get_possible_chords()` in this function in order to receive full credit.

`write_progression_file()`

(sig: `list, str -> None`)

Finally, we'd like to write a function `write_progression_file()` that accepts a chord progression list like the one returned by `get_chord_progression()` and **writes a file** like the one in figure 2. That is, it writes one CSV line per tuple. Your function must also accept a string denoting the name of the file that `write_progression_file()` will create:

```
chord_progression = get_chord_progression("chord-progression.csv",
"chords.csv")
write_progression_file(chord_progression)
```

Your function must work for any list of any number of different chord tuples.

The 2nd Students' Choice CA Awards

This semester, we are continuing our greatest tradition here in the CS1114 team: the Students' Choice CA Awards!

The rules are simple:

- Select your 2-5 favorite CAs from the check-boxes. While you can select more than that, limiting your choices will make your vote more meaningful.
- If you want explain your choices and maybe send a thank you note to your chosen CAs, feel free to do so!
- Voting will close Wednesday, Dec 7th, 2022 at 1pm.

Click **here** to access the voting form. The top three winners will be announced during our last lab!

Addendum

Those of you with some music theory background will notice that notes that can have two names (i.e. B-flat can also be called A-sharp, etc.) are standardised to a single name in the problem.

That is every instance of a(n)...

- A-sharp was converted to B-flat.
- D-flat was converted to a C-sharp.
- D-sharp was converted to an E-flat.
- G-flat was converted to an F-sharp.
- A-flat was converted to a G-sharp.

We basically standardised these notes to only have one name for the sake of simplicity. It's not important to know what this means; we just initially uploaded a README document that had examples before these measures were put into place by accident. Everything should be okay now.