Project 2: The Sound of Music

Due date: April 10 at 3:30pm.

You will not demo this homework to a lab leader; instead, you will upload it to Dropbox when you are *complete*, **even if it is after the due date**. There are no written discussion questions for this homework.

Overview

In this project you will develop an application that plays musical notes, chords, and scales according to a user's request.

I will provide a Python module called `musicbox` that contains code for playing musical notes through your computer's MIDI synthesizer. Your program will gather input from the user, validate it, and then "drive" the `musicbox` module with commands to play appropriate notes and scales according to the user's request. The `musicbox` module will output information about what notes it has been asked to play in addition to generating those sounds through your speakers. You will work with functions, lists, and strings in this assignment, in addition to previous topics like loops and arithmetic.

Introduction

In music theory, a **note** is a sound with a particular *frequency*, that is, the sound wave generated by the note oscillates a certain number of times per second. For example, the note "middle A" has a frequency of 440 Hz, and cycles 440 times per second. Notes with higher frequencies sound "higher" in pitch and oscillate more frequently than notes with lower frequencies/pitch.

In Western music, an **octave** (a range of sound frequencies, where the ending frequency is exactly 2 times the starting frequency) is broken into 12 equal parts called **temperments**. 7 capital letters and a pair of accent marks ("sharp" and "flat") are used to name each of the 12 parts, traditionally starting with the letter C as such:

C C♯ D D♯ E F F♯ G G♯ A A♯ B (C)

The "distance" between two adjacent notes in this arrangement is called a "half step", and the distance between every other note is a "whole step." The ♯ ("sharp") symbol denotes a half step above the preceding letter. Alternatively, sharps can be replaced by equivalent ♭ ("flats") which are a half-step below a preceding letter. For example, the notation above with flats instead of sharps looks like

C D♭ D E♭ E F G♭ G A♭ A B♭ B (C)

and you should understand that saying D♭ is the same as saying C♯.

If this seems complicated, it is, and only after a few courses in music theory would one really appreciate how this works. In a computer, however, music is represented in a different way using numbers instead of letters.

Computer representations of music:

An alternative way of describing musical notes is through **integer notation**. In this notation, the note "C" is given an integer value of 60, "C♯" a value of 61, "D" a value of 62, all the way until note "B" having a value of 71. A value of 72 is given to the note that is one octave above "C" – this is also a "C" note, but its frequency is twice that of the note with integer value 60. The sequence of notes seen above is then represented solely with integers as such:

60 61 62 63 64 65 66 67 68 69 70 71 (72)

MIDI synthesis:

The question then becomes how to generate sounds from a computer. One method is using the **MIDI standard** – for **M**ultiple **I**nstrument **D**igital **I**nterface. MIDI is a method for playing synthesized sounds from a computer sound board. Because the sounds are synthesized (and not sampled from recordings of real, live instruments), MIDI music often sounds "fake", "electronic", etc., but despite this the standard is used in every electronic keyboard, drumset, or other digital instrument; in music video games like *Guitar Hero* or *Rock Band*; and in many popular genres of music that feature synthesized sounds. (Indeed, most pop music uses synthesized instruments rather than paying for live musicians.)

MIDI was developed using the **integer notation** for specifying notes. When you write a program using MIDI, you instruct the computer to "play note 63", whereas you would instruct a live musician to "play note D sharp." Playing music through MIDI then becomes as much about arithmetic, loops, and general programming as it does about musical knowledge.

Scales:

A **scale** is a sequence of musical notes in ascending or descending order. **Scales** start with a particular note (e.g. C, A, or B-flat) and choose subsequent notes in the scale by following a pattern specific to the "type" of scale used. For example, a "C major scale" is comprised of the notes C-D-E-F-G-A-B-C, and a "C minor" scale uses the notes C-D-E flat-F-G-A flat-B flat-C.

A **major scale** is a sequence of notes, where the steps between notes in the scale follow this pattern:

whole, whole, half, whole, whole, whole, half

For example, the C major scale is

C D E F G A B C
whole whole 1/2 whole whole whole 1/2

and the A major scale is

A B C♯ D E F♯ G♯ A

In integer notation (what we will be using in the computer), the steps in a major scale (called **intervals**) are instead written

2, 2, 1, 2, 2, 2, 1

indicating that the second note in the scale has an integer value 2 greater than the first, so on down the line. The C major scale in integer notation would then be

60 62 64 65 67 69 71 72

A **minor scale** is a sequence of notes where the steps between notes in the scale follow this pattern:

whole, half, whole, whole, half, whole, whole

The C minor scale is

C D E♭ F G A♭ B♭ C
whole 1/2 whole whole 1/2 whole whole

Recall that E♭ is the same note as D♯. In this scale, E♭ is chosen to avoid repeating the letter D in the scale, i.e., C D E♭ vs. C D D♯.

The E minor scale is

E F♯ G A B C D E

Again, F♯ is chosen as opposed to G♭ to avoid repeating the letter G.

In integer notation, the intervals in a minor scale are written

2, 1, 2, 2, 1, 2, 2

The C minor scale in integer notation is

60 62 63 65 67 68 70 72

Note that there is no confusion about using D♯ vs. E♭: they are both the integer value 63, so there is no ambiguity.

You should now understand that playing a scale through MIDI is about selecting the integer notation value of the starting note, and then following a pattern for the particular scale type to generate the following notes in the scale.

:

Operations

Your program will act as a driver for my `musicbox` module. You will give a menu to the user, accept and validate their input, then call certain methods of the `musicbox` module in order to play notes, scales, and chords as requested. You will support these specific menu choices:

1. Play notes
   (a) Given a sequence of notes, play each one through the music box.
2. Play scale
   (a) Given the name of a note and a scale type (major or minor), play each note in the scale through the music box.
3. Quit.

Data and Functions

Your program must declare these constant variables in a global scope at the top of your .py file:

- A list called `NOTE_LETTERS`. This list should contain a series of note letters corresponding to the seven musical notes **without sharps or flats**, starting with "C" and ending with "B".
- A list called `NOTE_NUMBERS`. This list should contain the **integer notation values** corresponding to the seven musical notes in `NOTE_LETTERS`, **in the same order as** `NOTE_LETTERS`. For example, the first element of `NOTE_LETTERS` should be `"C"`; the first element of `NOTE_NUMBERS` should be the integer notation value of C, which is `60`.
- A list called `MAJOR_SCALE_INTERVALS`. This list should contain integers corresponding to the **intervals** of a **major scale**. Likewise, a list `MINOR_SCALE_INTERVALS` should contain the interval integers of a minor scale.

You will then create these functions, for use by the main program:

1. `def note_to_int(note)`
   (a) Function behavior:
      i. Given a string representing a note (examples: `"A"`, `"B#"`, `"Fb"`), this function **returns** the integer value corresponding to that note, in the range 60 to 72 inclusive.
      ii. You must do this **without** a long list of `if` statements, by looping through your `NOTE_NAMES` variable, looking for an element that is equal to the first letter of the `note` string.
      iii. If you find such an element in `NOTE_NAMES`, take the integer value that you find at the **same index** in `NOTE_NUMBERS`, and adjust that value up or down if the `note` has a sharp (`#`) or flat (`b`).
      iv. Return the adjusted integer value corresponding to the parameter `note`.
      v. -1 should be returned if the note is not recognized, either by being an invalid letter (`H`) or an invalid symbol (`C%`).
   (b) Restrictions:
      i. You can use at most **5 total** `if`, `elif`, or `else` statements in this function.
      ii. Do not print or use `input()` from this function.
   (c) Examples:
      i. `note_to_int("C#")` returns 61; `note_to_int("Ab")` returns 68.

2. `def note_to_scale(note, type)`
   (a) Function behavior:
      i. Given an **integer** representing a note, and a **string** representing a scale type (`"major"` or `"minor"`, as a string), this function **returns** a list of notes making up the given scale.

    ii. Use the given integer and the intervals in `MAJOR_SCALE_INTERVALS` or `MINOR_SCALE_INTERVALS` to append 8 integer values into a list
    iii. Return the list of 8 notes.
  (b) Restrictions:
    i. You cannot call `note_to_int` from this function.
    ii. Do not print or use `input()` from this function.

3. `def print_menu()`
  (a) Function behavior:
    i. Prints the program's main menu. See the Example Output.
  (b) Restrictions:
    i. Does not use `input()`.

4. `def get_menu_choice()`:
  (a) Function behavior:
    i. Read the user's selected menu option using `input()`, and validate it.
    ii. The validated selected option is **returned**.
  (b) Restrictions:
    i. No other functions are called from `get_menu_choice`.

5. `def get_notes()`
  (a) Function behavior:
    i. Ask the user to input a sequence of notes.
    ii. Split the input to break the string into individual notes.
    iii. For each individual note, convert that note into an integer using `note_to_int`.
    iv. Add each note integer that was recognized by `note_to_int` into a list. Do not add any note that was not recognized. (Hint: what does `note_to_int` return for unrecognized notes?)
    v. Return the list.
  (b) Restrictions:
    i. You must return a list of integers, not a string or a list of strings.
  (c) Examples
    i. If the user inputs "`C D H Eb G`", you would return the list `[60, 62, 63, 67]`.

6. `def play_notes(notes)`
  (a) Function behavior:
    i. Given a list of **integers** corresponding to notes, call the `play_note` function on your music box for each note.
    ii. Use `500` for the second parameter to `play_note`, which will play each sound for 500 milliseconds.
  (b) Restrictions:
    i. Do not print from this function. (The music box will print, but you should not.)

7. `def menu_play_notes()`: implements menu option 1, "Play notes"
  (a) Function behavior:
    i. Call `get_notes` to get the user's selected notes to play.
    ii. If the resulting list is empty, print a message to the user: "I don't know any of those notes".
    iii. If the resulting list is not empty, call `play_notes` with the list.

8. `def get_scale()`

(a) Function behavior:
   i. Ask the user to input the name of a scale.
   ii. Validate the scale name.
      A. A valid scale name has a valid note followed by *either* “`major`” or “`minor`”.
      B. Repeatedly ask the user for a valid scale name until they comply.
   iii. Return the validated scale name as a string.

(b) Restrictions:
   i. You must use `note_to_int` to validate the note part of the scale.

9. `def play_scale(scale)`
   (a) Function behavior:
      i. Given a list of **integers** corresponding to notes in a scale, call the `play_note` function on your music box, **once for each note in the list**.
      ii. Use `500` for the second parameter to `play_note`, which will play each sound for 500 milliseconds.
   (b) Restrictions:
      i. Do not print from this function. (The music box will print, but you should not.)

10. `def menu_play_scale()`: implement menu option 2, “Play scale”.
   (a) Function behavior:
      i. Call `get_scale` to get the user’s selected scale to play, as a string.
      ii. Split the string into two parts: the note to play, and the type of scale.
      iii. Use `note_to_int` to convert the note to play into an integer.
      iv. Pass the note integer and scale type to `note_to_scale`, receiving the returned list of integers
      v. Pass the list of integers to `play_scale`.
      vi. Enjoy the music.

Using the MusicBox Module

To use the MusicBox module, you must make sure to do the following:

1. Start PyCharm, then click Create New Project. (**File** menu.)
2. Set the Location to wherever you would like to save your project.
3. Make sure the Interpreter indicates that **Python 3** is selected – you should see a file path that has something like `Python3__` in it, where the `__` is a number like 5 or 6 or 7.
4. Click Create.
5. When the program fully loads (watch the lower right corner to make sure it does not say “XX processes running” and does not have a progress bar), go to **File->Settings**. On the left, expand **Project: nameofyourproject.** Click **Project Interpreter**. You will see a **green plus sign** – click it.
6. On this screen, type “pygame” into the search bar. Select it from the list that appears, then click the Install Package button. (This will take a minute.) When complete, close the two Settings windows to go back to the main PyCharms interface.
7. Go to BeachBoard and download the file `musicbox.py` from the Projects category. You need to place this file in the folder you chose when creating the New Project in PyCharm – you will recognize this folder because it will have a folder called `.idea`. If you do this correctly, you will see the file `musicbox.py` show up in the Project in PyCharm.
8. Right click on your Project in PyCharm and select **New->Python File**. Name the file `soundofmusic.py`. This is where you will write your application.

9. In your `soundofmusic.py` file, type these lines:

   ```
   import musicbox
   def main():
       m = musicbox.MusicBox()
       m.play_note(60, 1000)
       m.close()

   main()
   ```

   which will play a single note for 1 second. To run the program, go to the **Run->Run...** menu. In the window that appears, click on `soundofmusic`. If you get no errors and hear the note, then you can begin working on the project. Otherwise, seek help for the errors reported.

Once your project is set up correctly, you will use the following methods to play music:

1. First, at the top of your file, create a music box that you can use globally:
   `my_music = musicbox.MusicBox()`
   You can put a number from 0 to 127 in the parentheses to change the instrument that is played.

2. At the bottom of the program, after `main()` is called, write
   `my_music.close()`
   or else you will get runtime errors.

3. To play a note: `my_music.play_note(`**MidiNumber**`,` **Milliseconds**`)`

   (a) Plays a note with the given MIDI number for the given number of milliseconds.

   (b) The rest of your program will pause until the notes have played.

   (c) Example:
   ```
   noteToPlay = 61 # this is C sharp
   my_music.play_note(noteToPlay, 500) # plays C sharp for 500 milliseconds
   ```

Example output

NOTE: any line that starts with <MusicBox: is printed by my MusicBox. **You do not need to print those lines yourself.**

```
Main menu:
1. Play notes
2. Play scale
3. Quit
Please enter a selection:
1
Please enter a sequence of notes separated by spaces:
C D# Gb H
<MusicBox: play_note(60, 500)>
<MusicBox: play_note(63, 500)>
<MusicBox: play_note(66, 500)>

Main menu:
1. Play notes
2. Play scale
3. Quit
Please enter a selection:
1
Please enter a sequence of notes separated by spaces:
A! Bc Xx
I don't know any of those notes.

Main menu:
1. Play notes
2. Play scale
3. Quit
Please enter a selection:
2
Please enter a scale name (Ex. C major):
J# major
Please enter a scale name (Ex. C major):
Gb minor
<MusicBox: play_note(66, 500)>
<MusicBox: play_note(68, 500)>
<MusicBox: play_note(69, 500)>
<MusicBox: play_note(71, 500)>
<MusicBox: play_note(73, 500)>
<MusicBox: play_note(74, 500)>
<MusicBox: play_note(76, 500)>
<MusicBox: play_note(78, 500)>

Main menu:
1. Play notes
2. Play scale
3. Quit
Please enter a selection:
3
```