Make Music!

The goal of this assignment is to practice using ints, floats, expressions, user input, and functions from an imported module. And hopefully have a little fun while doing it.

Deliverables: a zip file named with your carleton username (e.g. my submission would be named sgoings.zip) containing **2 files**, your **music.py** file and your **music.txt** file.

- When creating the zip file (a compressed/zipped folder), make sure you name the folder with your username, **then** compress it, don't name the folder something else, compress it, and simply change the name of your compressed file to your username, otherwise when I and the grader uncompress your file the folder created will have that old name and we will have to figure out which folder is yours.
- To compress a folder on a mac, simply right click on the folder and choose "compress 'name_of_folder'", the compressed folder will then appear in the same place as the original with the same name as the original folder plus the .zip extension.
- To compress a folder on a PC, right click on the folder and choose "send to zip file" or something similar depending on what version of Windows you are running.

Note: this is an individual assignment, but you are encouraged to get as much help still as needed, from myself, the prefect, the lab techs, and your classmates. You should not, however, copy anyone's code, rather you should discuss/share ideas and help each other to debug errors if needed. Be sure to re-read the section on the syllabus about plagiarism and especially be clear in your comments of any sources you receive help from!

Basics

One notation that can be used to specify musical tones is x.y where x represents the octave and y the step in the octave. There are 12 steps in an octave on a standard instrument (see this link for an online virtual keyboard to experiment on). So typically, if 0.0 represented the note "A", 0.9 would represent the note "F#", which is 9 steps above "A" on a keyboard, and 1.3 would represent the note "C" on the next higher octave. We, however, are going to use a system borrowed from planet unrealistica, where octaves are ½ the standard size, so the values of y in x.y can only be 0-5. We also have a limited amount of notes (due to issues with Python sound packages and the lab macs), so you will only have 13 notes to work with, 2 full unrealistica octaves (or 1 full earth octave). This is likely confusing, so look at the mapping below to see the exact notes that correspond with each specified value.

0.0	0.1	0.2	0.3	0.4	0.5	1.0	1.1	1.2	1.3	1.4	1.5	2.0
A	A#	B	C	C#	D	D#	E	F	F#	G	G#	A2
0	1	2	3	4	5	6	7	8	9	10	11	12

Tones module

I have stored these notes in a list in the tones.py module that you can use to actually play these tones on the lab computers (it may also work on other computers, but absolutely no guarantees). The bottom row of the above table shows the index at which each note is stored. You will need to use this number to play the given note.

To use the tones module you will need to download the “tones.py” file to your computer, and save it in the same folder as your own music.py file. You will then add the line

```
import tones
```

at the top of your program in order to be able to use the functions in the tones module in your own code. To use one of these function in the tones module you will use the following syntax

```
tones.function_name(parameters)
```

The tones module includes the following functions for you to use:

`setup()` – takes no parameters, sets up the music file to allow tones to be written

`closefile()` – takes no parameters, closes the music file

`addNote( pitch, length )` – writes a note to the music file. Pitch will be an integer 0-12 that represents a note as shown in the previous table. It must be an integer because this function uses it to index into a list. You will get an error if you try to use a float. Length is the amount of time the pitch will play in hundredths of seconds (i.e. length=100 means the note will play for 1 second).

`addPause( length )` – writes a pause to the music file of the specified length. Length is once again in hundredths of seconds.

Your task:

Your task will be to write a program that asks the user for the note they want to play in the “octave.step” notation previously discussed (shown in the top row of the above table), and then uses functions from the “tones” module to write the notes to a music file which you can then play. I have broken this task down into the following steps for you to complete.

Step One:

Download tones.py and save in the same folder as the music.py file you create. The first line of music.py should be:

```
import tones
```

Next call the `setup()` function from the tones module. The code will look like this:

```
tones.setup()
```

Next put the code to call the `closefile()` function from the tones module at the END of your program. You want to add this now because if you forget to close it, it can cause problems later. The code looks like:

```
tones.closefile()
```

Now play with the `addNote` and `addPause` functions, try adding the following code to see an example of a simple tune:

```
tones.addNote(0, 50)
tones.addPause(10)
tones.addNote(0, 50)
tones.addPause(10)
tones.addNote(7, 50)
tones.addPause(10)
tones.addNote(7, 50)
tones.addPause(10)
tones.addNote(9, 50)
tones.addPause(10)
tones.addNote(9, 50)
tones.addPause(10)
tones.addNote(7, 75)
tones.addPause(10)
tones.addNote(5, 50)
tones.addPause(10)
tones.addNote(5, 50)
tones.addPause(10)
tones.addNote(4, 50)
tones.addPause(10)
tones.addNote(4, 50)
tones.addPause(10)
tones.addNote(2, 50)
tones.addPause(10)
tones.addNote(2, 50)
tones.addPause(10)
tones.addNote(0, 75)
tones.addPause(10)
```

Run your program and **look in the same directory as your program file is stored for a file named "music.aifc".** Open this file (simply double-click and it will open in itunes) and it should play a familiar tune.

Step Two:

Now you will write the code to ask the user for the notes they want to play. First **remove** the code that plays the example tune, you should not include that in your final project! The code below creates a loop that will allow the user to continue entering notes and pauses until they enter "q" as a note to indicate that they are done. The commented lines inside the while loop describe the code you need to add to make the program work.

```
note = input("enter note: ")
while note != 'q':
    # convert user input to the corresponding note's index (remember
    input returns a string not a number)
    # prompt the user for the length of the note and store it
    # prompt the user for the length of the pause after the note and
    store it
    # tones.addNote( your converted pitch, length of note)
    # tones.addPause( length of pause) - note could be 0!
    note = input("enter note: ")
```

Remember the user is inputting each note in the format “x.y” where x is 0-2 and y is 0-5, and your program must convert it to the appropriate integer pitch 0-12. You can assume the user will not enter any notes other than the 13 allowed (shown in the table in the beginning). You **may NOT** use “if” and “else” statements to do this conversion. You can do it using only the int and float operations we learned in class. Remember the idea of modulo arithmetic, if the user enters 0.3, you want the note (0+3), if they enter 1.1, you want the note (6+1), if they enter 2.0 you want the note (12+0). Notice the relations: 0,0 ; 1,6 ; 2,12. Don’t forget to **convert the note the user enters to a float** before you try to do math operations on it, and to **convert your final pitch to an int** if need be. You will get an error if you pass anything other than an int to `addNote()`.

Test your code by entering various notes, lengths, and pauses.

Step Three:

Finally write a song in text file called “music.txt”. You will do this by writing one number per line, like this:

```
Note (in form x.y)
Note length
Pause length
Note
Note length
Pause length
….
q
```

Be sure the ‘q’ is on the line that “Note” would usually be on, the program assumes there is a pause after every note, but you can use a pause of 0 to make it so you don’t hear it. So for instance the sample code I gave you in part one that plays a simple song would look like this in a file:

```
0
50
10
0
50
10
1.1
50
10
1.1
50
10
1.3
50
10
1.3
```

```
50
10
1.1
75
0
.5
50
10
.5
50
10
.4
50
10
.4
50
10
.2
50
10
.2
50
10
0
75
0
q
```

To have your program play your file, put your music.txt file in the same folder as your music.py program, and on the command line type this:

```
python3 music.py < music.txt
```

The “<” tells your program to get input from a file instead of the keyboard, so it will read each line from your file as if you had entered them one at a time at the prompts. Note it will still output the messages in your prompts, that is OK!

Now play the resulting “music.aifc” file, it’s your song! Your song can be as simple or complex as you like (with just the 13 notes that are available), but write something in the file and make sure it works.

Step Four - Bonus (2 extra pts):

What happens if the user tries to enter a note that is outside of the allowed range? Such as 2.5? Add error checking (you may use “if/else/while” statements and now) so that if the user enters an invalid note you print a statement saying the note is invalid and ask for a new note. This should happen anytime the user enters an invalid note, even if they enter invalid notes many times in a row. Only when they enter a valid note should your program move on and ask for the note length.