CS 105 – Lab #12 – Making music in Python

To begin, please create a new folder on your USB drive or account, and call it lab12. Save all of today's work there. The class handout showing you the musical scale may be helpful for this lab!

Setting up

The computer speakers can be loud. So, you may want to adjust the volume. Here is one way to do it. Click the Start button, and type "adjust system volume." A volume box will appear, and you can adjust to a lower level.

Next, we need to install the Python music system. Go to the class Web site, and click on the link "Jython Music." Save the file to your lab12 folder or the desktop. This is a ZIP file, representing a large folder that has been compressed. To uncompress it, right click on the file icon and select the option to "extract all." When this is finished, you should see a new folder appear called jythonMusic. Double-click on this folder icon to go into the folder. Find the file called JEM2.jar, and double click on this icon to invoke the Python editor.

You will need to keep the JEM program open throughout the lab today because we will use it to run all of our music programs.

Underneath the word "File" in the menu bar, you should see two buttons that will be useful to us today. The triangle-shaped button is used when we want to run a program. The square button next to it may be helpful if you want to exit a program while it's running. In a way, these buttons have the effect of play and stop.

Now we're ready for some music!

A dry run

Let's make sure that we are able to play anything and that your speakers are adjusted to the proper volume. Download one of the pre-existing Python music programs from the class Web site. For example, one-note.py. Save it to your lab12 folder.

Inside JEM, go to the File menu and choose Open. Open the one-note.py program you just downloaded. You should see the source code appear in the JEM editor window. Now, click the run button. Did you hear anything?

Now that we are done with one-note.py, we can close the file. Go to the File menu, and select Close. It is best to close a file when you are finished with it, so that you are only working on one program at a time.

Structure of a Python music program

Today’s programs will be short and they will have a simple structure, as follows:

```
# example_template.py
from music import *

# Our music will be put into a "phrase."
p = Phrase()
# Here you can set tempo and/or instrument

# To insert a single note, use p.addNote like this
p.addNote(C4, QN)

# Or, to insert a list of notes, create a list of pitches
# and a list of durations, and then use p.addNoteList
pitch = [C4, E4, G4]
duration = [EN, EN, HN]
p.addNoteList(pitch, duration)

# To insert a chord, use p.addChord.  But note that the
# first argument is a list of pitches.
chord = [BF3, D4, F4]
p.addChord(chord, HN]

# At the end of the program, we tell Python to play the music.
# Please not that this play command should only appear once.
Play.midi(p)
```

As you can see, there are basically 4 essential things you must put in your program.

- Tell Python to include everything from the music library.
- Create a new Phrase.
- Add musical notes and other attributes such as tempo to the phrase. I usually like to set the tempo because the default tempo is often too slow.
- Play the phrase.

Your first program: scales

Our first program will work with the C major scale. Inside JEM, create a new Python program and call it scale.py. Type the following code into the editor:

```
# scale.py – Practice with musical scales.

from music import *

pitch =    [C4, D4, E4, F4, G4, A4, B4, C5]
duration = [QN, QN, QN, QN, QN, QN, QN, QN]

p = Phrase()
p.setTempo(120)
p.addNoteList(pitch, duration)

# We'll add more music later...

Play.midi(p)
```

Save and run your program. You should hear the C major scale!

Next, let's add some more music. We will play the C major scale in reverse. There is a straightforward way to do this without typing all the notes again. Notice that we have a list called `pitch`. In Python, it's easy to reverse the order of elements in a list. If L is a list, then you reverse the list with the statement: `L.reverse()`. With this in mind, enter the code necessary to reverse the pitch list, and then run the program. It should play the C major scale going down immediately after it goes up.

1. What code did you need to add to your program to accomplish the descending scale?

Before we continue, please enter a statement in your program that will reverse the pitch list again so that it is going in ascending order.

Next, let's raise the scale to C#. We do so by *adding 1 to each note* in the list of pitches. Copy the following code into your program:

```
pitch2 = []
for note in pitch:
    pitch2.append(note + 1)

p.addNoteList(pitch2, duration)
```

Save and run your program. Your program should now play 3 scales: the C major scale going up, the C major scale going down, and then the C# major scale going up. You should notice a slight uptick in pitch, because C# is the next note higher than C.

Finally, let's play some chords. Specifically, we will play all 12 possible major chords, from C major, C# major, D major, …, all the way up to B major. We will insert each new chord into the phrase using a loop. Copy the following code into your program after the three scales you just did:

```
chord = [C4, E4, G4]
for i in range(0, 12):
    newChord = [chord[0] + i, chord[1] + i, chord[2] + i]
    p.addChord(newChord, HN)
```

Save and run your program. Count the chords that you hear to verify that there are 12 of them.

If we can play major chords, why not minor chords too? A minor chord is almost exactly the same as a major chord. The only difference is that the second note is one half step lower. So, rather than starting with [C4, E4, G4], we would start with [C4, EF4, G4]. In other words, the E becomes E-flat. Insert the code to play the 12 minor chords immediately after the major chords. The way you write the minor chord loop is the same as with the major chord loop.

Have the instructor or lab aide listen to your program, and then you may close the source file.

Hee-Haw!

Let's write a little program that plays the same pair of notes several times in order to imitate the sound of a mule. Hee-haw-hee-haw-hee-haw-hee-haw!

Create a new source file, and call it mule.py. In it, create a phrase, and set its tempo to 120 beats per minute. You should select an instrument other than a piano.

Now, on to the notes. Write a for-loop with 4 iterations. Inside the for-loop, add a high note (the "hee") followed by a low note (the "haw").

2. What instrument did you use? What two statements did you put inside the body of your for-loop?

Save and close mule.py.

Jaws

Another music effect we should experiment with is the infamous motif from the movie *Jaws*.

The purpose of this experiment is to slowly accelerate the sound. It won't work by modifying the tempo. Instead we will do something a little unusual: we will reduce the length of the notes by 10% on each iteration of a loop.

Inside JEM, create a new source file called jaws.py. In it, create a new phrase, and set its tempo to 120 and its instrument to a BASSOON. Then, copy the following code into your program:

```
# establish a starting duration
beats = 3.0

for i in range(0, 24):
    p.addNote(_____, beats)
    p.addNote(_____, beats)
    beats *= 0.9

Play.midi(p)
```

Note that I did not specify the pitches in the addNote() function calls. You should determine what these notes should be by experimentation. They should be one half step apart. After you have played your program and scared your neighbor, save and close your program.

3. Which two note values did you select? Why those in particular?

A simple song

We are now ready to tackle a song. Maybe you will recognize it…. Create a new source file called row.py. I am going to give you the correct pitches, but not the correct durations. You will need to figure out how to set the durations appropriately in the program. Copy the following code into your program.

```
from music import *

p = Phrase()
```

```
p.setTempo(100)      # 120 is too fast

pitch = [C4, C4, C4, D4, E4, E4, D4, E4, F4, G4,
         C5, C5, C5, G4, G4, G4, E4, E4, E4, C4, C4, C4,
         G4, F4, E4, D4, C4]

# these durations are not right, but will be a good start
duration = []
for i in range(0, 27):
    duration.append(QN)

p.addNoteList(pitch, duration)

Play.midi(p)
```

Save and run the program. You need to fix the durations of this song. It turns out that the notes are not all quarter notes. So, instead of using a loop, it will probably be necessary to list the 27 notes individually, although many of the notes will have the same length.

Important hint: many of the notes will have a length of one-third of a QN, and others will have a length of two-thirds of a QN.

When you are ready to play, save your program. Run your program, and get your neighbor to start his/her program after yours has played the first 5 notes!

Transcribing sheet music

It would be a very useful skill if you could take sheet music of an actual song, and transcribe the notes into a Python program. Give it a try! Find some sheet music online, or try the example given out by the instructor. Note that if you have a song like Moon River, you may notice that the same set of notes is repeated, so you only need to enter these notes once, and then enclose this code inside a loop that has 2 iterations.

Play your musical piece for the instructor or lab aide.