About Lab 9

In Lab 9 you will create a Soundwave class in file soundwave.py, test it out by running a few programs that import and use your Soundwave class, and then write 2 programs on your own that use it:

scale.py	This asks the user for a tonic note (as a number of steps above or below middle C) and a mode (major, minor, blues), and then plays the indicated scale
mozart.py	This creates a composition from a randomization algorithm that Mozart himself wrote.

The Soundwave constructor looks like this:

```
def __init__(self, halftones=0, duration=0.0, amp=1.0, samplerate=44100):
```

The body of the constructor does nothing except build up a list self.samples by evaluating a sine function for every value of t in the range from 0 to int(duration*samplerate)

Even if you loathed Trigonometry in high school you can do this; it is just a matter of calling math.sin( ) with the argument given to you in the lab directions.

The Soundwave class has 3 additional methods:

The play method is just

```
def play(self):
    audio.play(self.samples)
```

There are two methods for combining Soundwaves: concat and plus. Concat is used when you want to play one sound, then follow it with another. Plus is used when you want to play two sounds simultaneously.

The concat method is

```
def concat(self, s2)
```

where s2 is another Soundwave objects. This just adds s2's samples onto the end of self's samples. So of S1 and S2 are soundwave objects

```
S1.concat(S2)
```

modifies S1 to be longer.

The plus method is

```
def plus(self, s2)
```

This adds together the individual samples of self and s2, and returns a new Soundwave object with the resulting sample list. The only thing tricky about this is handling the fact that self.samples and s2.samples might have different lengths. You want to sum until the shorter list runs out, then tack on the extra elements of the longer list.

For example, if one Soundwave had samples

[2, 3, 5, 7]

and the other had

[1, 2, 3, 4, 5, 6]

you would want the result to have samples

[3, 5, 8, 11, 5, 6]

Before you start coding this think of the algorithm you will use. There are several ways to think about it, just find one and be sure it works on paper before you code it.

For the scale.py program you should have a loop that asks the user for a mode, with possible responses "major", "minor", "blues", and "quit". If the response is "quit" just exit the loop and the program. For the other three responses ask for the starting note of the scale, as an offset from middle C. You should then create a Soundwave object that creates and plays this scale.

If variable *offset* holds the starting note, we can make a Soundwave that represents a half-second tone at that pitch with

```
S = soundwave.Soundwave(offset, 0.5)
```

For the rest of the notes, walk through the appropriate Intervals list, computing the offsets and concatenating the corresponding Soundwave objects onto S.

For the mozart.py program you have an algorithm for constructing a minuet and trio based on measures written by Mozart. The measures are in .wav files with names like M32, T41, and so forth. The algorithm is easy, but it is based on having the correct numbers in two large two-dimensional tables. We give you those tables in two files, so the first step of your program is to read the files into the tables.

Both tables should be divided into rows with 16 entries in each row. The text files we give you have the numbers in one long sequence, separated by spaces.

You need the individual numbers in the file. You can do this by opening the file into variable F, reading F into a string s, and splitting s into the individual fields,

or by using our usual

```
for line in F:
    nums = line.split( )
```

Once you know how to get the elements of the file, do the following:

```
Table = [ ]
row = [ ]
numInRow = 0
for every number in the file:
        append the number to row
        add one to numInRow
        if numInRow is 16:
                append row onto Table
                row = [ ]
                numInRow = 0
```

Note that you don't need to evaluate the numbers you get out of the table -- you can leave them as strings. We will use them as part of a file name. If you get the number 42 out of the Minuet file, the next measure of your composition is

```
soundwave.Soundwave( "../Mfiles/M42.wav" )
```

Once you have built the tables, the actual algorithm is easy. For the minuet, you need 16 measures, so 16 times you generate a random number between 0 and 10. Suppose the number you get for measure 3 is 7. The minuet table entry for row 7, column 3 is 50. Then you will concatenate onto your minuet the file "../Mfiles/M50.wav" You do the same thing for the trio, but your trio table has only 6 rows so you choose a random number between 0 and 5. Finally, you concatenate your minuet again onto the composition (so save the minuet file numbers when you generate them).