# EECS 110: Project 1 (60 Points)

Audio Track

---

This assignment asks you to apply what you've learned to create a variety of interactive music features. This is a newly designed project, so unlike the graphics track, I do not have any prior examples of student projects. Please calibrate your effort according to your experience level with programming. You'll get out of it what you put into it. See the rubric below to get a sense of how you will be evaluated on this assignment. Download the starter files and navigate to the option2_audio folder.

## Dependencies

If you haven't already, you will have to install two pieces of software before you begin this assignment:

1. Sonic Pi
   Please download the version that corresponds with your operating system (Windows or Mac).
2. psonic
   Please open your command prompt (if you're a Windows user, don't forget to use the Anaconda command prompt instead of the "normal" command prompt), and type the following:

```
$ pip install python-sonic
```

## A Note on Threading

One complication to the audio track (and why it has taken me so long to design this assignment!) is that up until this point, we only have the knowledge to run riffs sequentially: i.e. one riff has to end before the next one starts. However, in the real world, multiple tracks (or instruments, or voices) play at the same time. Therefore, to accomplish playing multiple riffs at the same time, we have to use something called threading — where the operating system lets several programs appear like they're running at the same time. This is beyond the scope of this class, so I have designed a little module called threader.py that manages threading for you. You don't need to understand how it works — just how to use it. A sample can be found in **bach.py** (at the bottom, after the riff functions):

```
# Play
controller = threader.SonicPiController(tempo=0.5, riff=controller_riff)
low_loop_thread = threader.SonicPiLoop(controller, low_part)
high_loop_thread = threader.SonicPiLoop(controller, high_part)
low_loop_thread.start()
high_loop_thread.start()
```

**Step 1**: Create a ***SonicPiController*** object, which accepts an optional tempo argument and an optional riff argument. The controller's job is to synchronize and coordinate your threads (so that they play at the same time).

**Step 2:** Once you make the controller, you can subsequently create as many looping or sample threads as you want by doing the following:

a. Create a riff function that accepts a keyword argument called tempo. In this example, "low_part" and "high_part" are each riffs that accept a tempo argument.
b. Create a ***SonicPiLoop*** object (if you want the riff to repeat) or a ***SonicPiSample*** object (if you don't). Each object requires (1) a *SonicPiController* argument (which you created in step 1), and (2) a riff function argument (which you created in (a)).
c. Once you have created your threads, you can start and stop them by calling the stop() or start() methods.

## Part 1: Make a Midi Reader (35 points)

In your starter code, you have two text files: bach_1.txt and bach_2.txt. You are going to write a program that can read and play music that conforms to the following file format (an example of which is shown below, in the bach_1.txt file:

```
# Bach Minuet in G
# First 8 measures (low part)
55,59
-|1
57|0.5
59|1.5
60|1.5
59|1.5
57|1.5
55|1.5
62,59,55,62|0.5
```

```
50,60,59,57|0.25
```

The rules of this file are as follows:

1. Anything that has a hashtag in front of it should be ignored
2. Each line represents a different rule set (blank lines are ignored)
3. The pipe character's job (|) is to separate the notes or pauses from their duration:
    a. What comes before the pipe is ***either a note or a pause***:
        i. A single dash (-) character represents a pause (or rest)
        ii. Integers represent midi notes. If there is more than one midi note, the notes will be separated (or delimited) by commas.
    b. What comes after the pipe is the ***duration***: how long the note or pause should last:
        i. if there is only one note or pause before the pipe, then the pause should be applied to each note sequentially
    c. If there is no pipe after a note or notes, they should be played at the same time (a chord).
4. An example of how bach_1.txt and bach_2.txt should be played can be found in bach.py. Note that each 'part' gets its own thread so they can be played at the same time (as described in "A Note on Threading") example.

## YOUR TASKS

1. Make a new song (happy_birthday.txt) that conforms to the specifications outlined above.
2. Create a file called midi_reader.py
3. Write a program that prompts the user for a file_name (like bach_1.txt or bach_2.txt) and then plays the song as specified in the text file.
4. Show that your reader works for bach_1.txt, bach_2.txt, and happy_birthday.txt
5. We will be testing your reader against a new midi file that we create :)

## Part 2: Make a GUI Music Mixer (25 points)

I have set up a GUI Mixer for you in main.py (run it) with all of the threads set up at the top of the file (see below). You will be replacing the looping and sample functions with your own sound designs.

```
controller = threader.SonicPiController(tempo=1, riff=riffs.controller_riff)

# 2. loops play continuously until they are stopped
loop_1 = threader.SonicPiLoop(controller, riffs.riff_8)
loop_2 = threader.SonicPiLoop(controller, riffs.riff_1)
loop_3 = threader.SonicPiLoop(controller, riffs.riff_3)
loop_4 = threader.SonicPiLoop(controller, riffs.riff_4)
loop_5 = threader.SonicPiLoop(controller, riffs.riff_8)

# 3. samples only play once
sample_1 = threader.SonicPiSample(controller, riffs.riff_5)
sample_2 = threader.SonicPiSample(controller, riffs.riff_6)
sample_3 = threader.SonicPiSample(controller, riffs.play_harley_sound)
sample_4 = threader.SonicPiSample(controller, riffs.riff_7)
sample_5 = threader.SonicPiSample(controller, riffs.riff_1)
```

### YOUR TASKS

Use the 'starter' interface provided (main.py) and replace the 5 different looped riffs and samples with your own. Checklist:

#### 1. Create five different riffs to be looped (10 points)

- Create 5 different riffs
- Ensure that each has one and only one keyword parameter called tempo, and that your riffs honor the tempo parameter. This is necessary for threading to work, and for the controller to control the tempo.
- Update your **loop_1**, **loop_2**, **loop_3**, **loop_4**, and **loop_5** objects (towards the top of your file) so that they use your newly designed riffs

#### 2. Create five different samples (10 points)

- Create 5 different samples. Whereas the riffs will be looped, the samples will not be
- Like with the loops, ensure that each has one and only one keyword parameter called tempo, and that the sample honors your tempo parameter (if it makes sense).
- Update your **sample_1**, **sample_2**, **sample_3**, **sample_4**, and **sample_5** objects (towards the top of your file) so that they use your newly designed riffs

### 3. Wire up the interface (5 points)

- Connect your loop and sample riff functions to the threads and to the checkboxes (for loops) and buttons (for samples)

### 4. Extra Credit Enhancements

1. **2 points.** Ensure that at least one of your functions makes use of the psonic.play arguments (e.g. sustain, amplitude, release, attack, etc.)
2. **2 points.** Create samples that come from mp3s that you find on the Internet (or create). Make use of the psonic.sample parameters (e.g. rate, attack, sustain, release, beat_stretch, start, finish, amp, pan).
3. **Up to 5 points.** When the riff is playing, display some visual display on the canvas (or give the user some feedback re: how long the riff is, what it does, etc.).
4. **2 points.** Make the + and - buttons at the top of the interface control the speed. You can do this by increasing and decreasing the controller's "tempo" variable.
5. **Up to 5 points.** A way for your samples or loops to read from a MIDI text file (in the same format as bach_1.txt and bach_2.txt, as described above).

## What to Turn in

1. Your code as a zip file. This should include all of the files that make your code work.
2. A link to a short (think ~15 seconds, but longer OK) video of your music mixer in action. To do this:
    a. Take a screencast (QuickTime works, and is free)
    b. Upload it to Google Drive
    c. Share your video so that anyone with link can view

## POLICY ON CODE SAMPLES AND COLLABORATION

**YOU MAY....**

- Make use of any sample code that I have provided you at any point during the course
- Consult with and incorporate ideas from Internet sources, so long as you are typing the code for yourself (not copying) and understand how every line of it works
- Help each other debug your code and discuss ideas together

**YOU MAY *NOT*...**

- Copy anything (except for sample code I have provided you)
- Share code — or look at your neighbor's screen and transcribe their code. We have plagiarism detection software to flag code similarities (e.g. MOSS) — even when whitespace, variable names, and ordering have been changed.

- Use code copied directly from third-party sources; or programming techniques that you don't understand.
- Talk to CS majors (or other more experienced people) and ask them to write code for you.

**RUBRIC (Out of 60 Possible Points)**

| Feature | Possible Points | Scoring Guidelines |
|---|---|---|
| New midi file | 5 | Made a new song (happy_birthday.txt) that conforms to the specifications outlined above. |
| Midi parser | 25 | - File is named midi_reader.py (2 points)<br>- Reader prompts user for a midi file and reads through it (3 points)<br>- Reader handles rests correctly (5 points)<br>- Reader handles chords correctly (5 points)<br>- Reader handles sequential notes (and tempos) correctly (5 points)<br>- Reader plays entire song w/o errors (5 points) |
| Creating sound loops | 10 | - User created 5 samples to be used for looping<br>- Loops were coherent and coordinated with one another |
| Creating sound samples | 10 | - User created 5 sample to be used for one-time sound effects |
| Creating the interface | 5 | - Students connected her riffs to the player. Player turns samples and loops on and off successfully |
| Code Quality | 5 | - All functions, variables, and file_names do not have pneumonic names / are not snake case (up to -1 point)<br>- Code is disorganized, with a lot of unused or redundant code — excluding the starter code that came with the assignment. (up to -1 point) |
| Extra credit | Up to 10 points | For projects that exceed expectations in any of the following ways:<br>- Went above and beyond with their riff designs<br>- Went above and beyond with their user interface design<br>- Completed any of the extra credit enhancements |