EECS 1710
LAB 5 :: Audio & FileIO

Prerequisite (labs 0-4) – please ensure that you have setup your EECS account, and can log into the lab machines prior to starting this lab, and you are familiar with submit/websubmit.

NOTE: This lab is worth 3% of your final grade.

STEP 1: Importing and unzipping the starter code

Download and extract the following lab4 starter code:
http://www.eecs.yorku.ca/course_archive/2022-23/F/1710/labs/lab5/lab5.zip

There are three files to submit for this lab. Assuming the above zip file is downloaded and extracted into the `1710/labs/` folder in your home directory, you should be able to navigate to the labs folder and submit everything with the following commands:

```
cd
cd 1710/labs/
submit 1710 lab5 lab5/*
```

This will submit all files within the lab5 folder. Alternatively, you may use web submit to submit/delete individual files (see link at the end of this document). To list files submitted, type:

```
submit -l 1710 lab5
```

STEP 2: Exercises

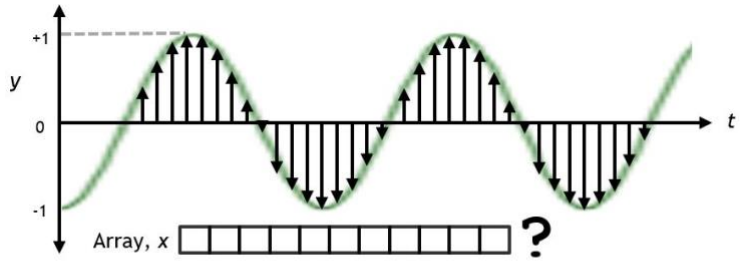
Please note, **the organization of your starter code** is similar to **`lab4`**, in that all the files for your lab are contained within a single sketch folder. The main file for the lab is **`lab5.pde`** (this contains your setup, draw and any event methods like `mousePressed()` that processing recognizes as standard), and all the other methods (stored in additional files within your sketch folder) are invoked from here.

The reference libraries for the Sound library can be found here.

Sound Library: https://processing.org/reference/libraries/sound/index.html
AudioSample: https://processing.org/reference/libraries/sound/AudioSample.html

Question 1: Synthesizing and plotting tones.

Recall that a sound's physical property of "frequency" corresponds to the human sensation of "pitch". In **`Question1.pde`**, and referencing the course lecture materials for Weeks 6-8 (arrays, objects and audio), construct a method that will output a float array for a single tone (sinusoid) of the given frequency.



$$y(t) = \sin(2\pi f t), \qquad t = \frac{i}{\mathrm{SR}}$$

`float[] generateTone(float duration, float freq, int sampleRate)`

- A typical value for **`sampleRate`** = 44100 (samples per second)
- The argument **`duration`** is a `float` that indicates the number of seconds the sound wave should run for when played (it can be used together with sampleRate to determine the total number of samples in the returned array
- The argument **`freq`** is the frequency of the sinusoid to be generated (according to the formulas discussed in Week 8 lecture slides (shown in the above figure)

- The **`setup()`** method should call the **`generateTone()`** method, and should store the resulting array in a variable called `myTone`.
- Include a call in setup to play `myTone` by instantiating a new **`AudioSample`** object (using `myTone`) and running the play method once (this is just to test your tone) - see lecture slides for more information.

  Note: you may have to install and import the Sound library within Processing, to make these audio objects available for your program – see lecture slides.

Modify your **setup()** method so that it generates several (3-4) waveforms (of differing frequency (440Hz, 580Hz and 820Hz), but similar duration (e.g. 1-2 sec) & sampling rate (44100 samples/sec), and stores them in separate variables.

In the **Question1.pde** file, add another method: **displayTone(** …), that plots the first **num** samples of a given **tone**, in a given **colour**, and according to a given **mode** of plot:

```
void displayTone(float[] tone, int yPos, int num, int amplitude,
                                                  int colour, String mode)
```

For this method, the **yPos** relates to where the tone's zero amplitude values will be positioned vertically in the app window). The tone should display only its first **num** values, and should extend for the entire width of the app window, and should have its amplitude scaled by the parameter **amplitude**.

The look of the plot will differ depending on the mode: a parameter that uses a String to identify which version of the plot will be created:

- **mode = "CIRCLE"** -> will plot each sample as a filled circle of radius 2
- **mode = "LINE"** -> will plot each sample as a positive/negative line extending from the **yPos** vertically to the sample point on the sinusoid.

The fill/stroke of the circles/lines will be determined by the parameter: **colour**
An example of plotting several waveforms is shown below:

```
float[] tone440 = generateTone(1.5,440,44100);
float[] tone580 = generateTone(1.5,580,44100);
float[] tone820 = generateTone(1.5,720,44100);

displayTone(tone440,height/6,1000,height/3,color(255,0,0),"CIRCLE");
displayTone(tone580,height/2,1000,height/3,color(0,255,0),"CIRCLE");
displayTone(tone820,5*height/6,1000,height/3,color(0,0,255),"LINE");
```

Add a **keyPressed()** method to your **lab4.pde** file, that plays each of the above tones through to finishing, depending on the key pressed. If pressing '1' it should play **tone440**, if pressing '2' it should play **tone580**, and if pressing '3' it should play **tone820**. If a tone is currently playing when a key is pressed, it should not play anything (see methods for `AudioSample` & lecture notes for how to query for this).

*** You will need a set of headphones to listen to your output if doing your lab on the lab machine.*

Question 2: Synthesizing a Chirp (up and down)

A chirp is a sinusoid that changes frequency with time (by either increasing or decreasing its frequency value). For this, you need to create a new method (which is similar to the method in **`Question1.pde`**, however there is a **`factor`** that is required to modify the frequency as the sample number gets higher – essentially your frequency value is getting incremented/decremented by multiplying it by this factor within the loop you use to generate your sinusoidal values. If the factor is slightly >1 (e.g. 1.0001), the frequency should grow, if the factor is slightly <1 (e.g. 9.9999) it should shrink. You may need to play around with the value to get a good rate of growth/decay.

```
float[] generateChirp(float duration, float freq, int  freqFactor)
```

- Assume a sample rate of `44100` samples/sec is used inside this method
- The freq can be modified by multipling by `freqFactor` (and saving this as the new value of freq for the next iteration of the loop generating the sinusoid. A value of just under 1 will result in the frequency slowly decreasing over time (a chirp down), while a value of slightly greater than 1 will result in the frequency slowly increasing with time (a chirp up).
- The **`setup`** method should call the **`generateChirp`** method, and should store the resulting array in a variable called **`myChirp`**.
- The **`setup`** method should then play **`myChirp`** by using it to instantiate a new AudioSample object, and playing that object (you may comment out any calls made to methods used in question 1 here)

Now, create a modified version of the chirp method above, called:

```
float[] chirpsBetween(float duration, float f1, float f2,
                                                    int repeats)
```

- Again assume sample rate is `44100` samples/sec in this method
- The frequency of the sinusoid should vary linearly (increment or decrement by a fixed step) between **`f1`** and **`f2`** over the **`duration`** of a single chirp
- The single chirp should be repeated **`repeats`** number of times

Test out your chirps (attempt to chirp up and down between two frequencies.. remember to not let the frequency go past these bounding values).

Question 3: Create your own "sound effect" by combining tones with frequency, noise and amplitude variations

In **`Question3.pde`**, you are to try to create a new method that combines waveforms (such as tones generated above, or other waveforms explored in lectures, with other effects such as noise, or amplitude/frequency articulations), to generate your own custom "sound effect".

```
float[] genSoundEffect( … )
```

You have complete freedom over how you explore this task (so you can define your own input arguments/parameters – although there are some basic ones you are likely to need – such as duration, sample rate etc.).

You can try to add waveforms directly (as we explore in Week 8 lectures), where the waveforms are equal in size (number of samples), ... or you can try to create longer waveforms that concatenate (join) different waveforms together end to end.

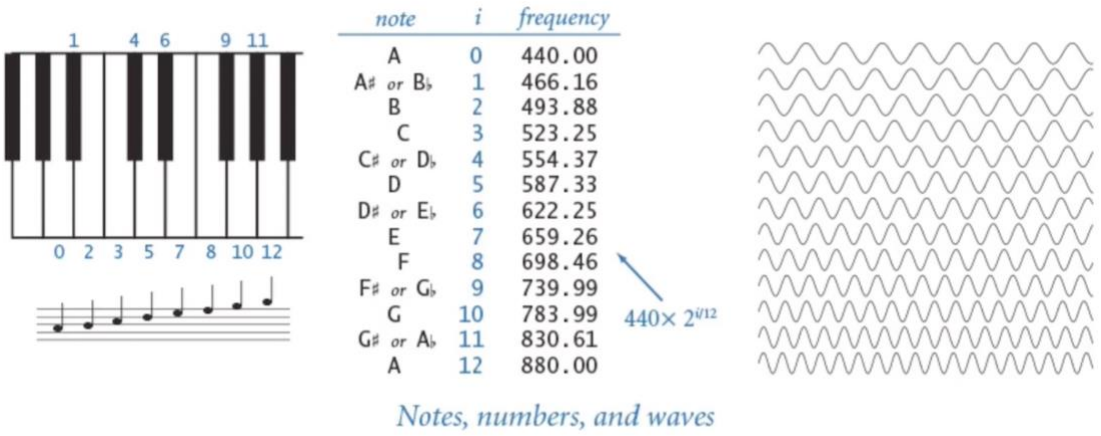
For example, a sinusoid for 1 second joined with a noisy chirp for 1 second, joined with a tremolo/articulated tone. Or you can try to combine different tones (of different frequencies, etc.). Essentially try to manipulate waveforms we have looked at in different ways, and listen to them. It is a good idea to make use of the **`displayTone()`** method from Question1 to see what things look like (you can adjust the number of samples shown to fit more of the waveform into the app window).

When you have generated your desired sound effect, it should be returned from your method as a **`float[]`** waveform. In the setup method, generate your sound effect and attempt to connect it to a mouse click or key press… so that you can readily play it back.

OR… if you are adventurous, connect it to a hit-test (like the one considered in Week 4, Lecture 7). When a hit/collision is detected (you can just create a simple object that moves with the mouse or keys that collides with another object in the scene), it should trigger your sound effect to play (remember to connect it to a new instance of AudioSample).

Question 4: Driving the tone generator from a file input

Create a new method in **`Question4.pde`** that will accept the **`duration`** and **`freq`** value from by an input set of tones and tone durations from an input file. In this example however, the notes are not expressed as **`freq`** values in the input file, rather as indexes **`(i)`** to keys (0-12) according to the following diagram:



Notes, numbers, and waves

A sample file is included in the sketch folder: "`sample1.txt`".

The format for this file is that each line represents a new tone, where the first digit is an integer representing `(i)` in the diagram above (i.e. a key on the piano); and the second number is a float representing the duration of that tone (in seconds). The idea is to read the lines of text into a `String[]` array or `StringList`, and process them to extract the numbers for each note. Use a loop to read each line from the `String[]` array, generate the tone, and play it, before repeating this until there are no notes remaining.

Run the method and listen to your output. Create a new input file and include it in your lab4 sketch folder for submission: "myMelody.txt" – see if you can come up with a decent melody of your own (anything is ok).

`sample1.txt` →

```
7 0.25
6 0.25
7 0.25
2 0.25
5 0.25
3 0.25
0 0.5
```

STEP 3: SUBMISSION (Deadline 5:00pm Tue Nov 15th, 2022)

NOTE: ***REMEMBER,*** *you can choose to use the web-submit function (see Lab 0 for walkthrough). You will need to find and upload your lab5 *.pde files independently if doing it this way.*

Web-submit can also be used to check your submission from the terminal, and can be found at the link: https://webapp.eecs.yorku.ca/submit/