Name: Evan Leglar	**Lab Time T 1200**

Names of people you worked with:
• Declan Siewart • Andrea oswalt

Websites you used:
• https://stackoverflow.com/questions/2582138/finding-and-replacing-elements-in-a-list • https://stackoverflow.com/questions/23377665/python-scipy-fft-wav-files • https://www.youtube.com/watch?v=17cOaqrwXlo •

Approximately how many hours did it take you to complete this assignment (to nearest whole number)?	2+2+4+8+10 = 32

The Rules: Everything you do for this lab should be your own work. Don't look up the answers on the web, or copy them from any other source. You can look up general information about Python on the web, but no copying code you find there. Read the code, close the browser, then write your own code.

By writing or typing your name below you affirm that all of the work contained herein is your own, and was not copied or copied and altered.

Evan Leglar

Note: Failure to sign this page will result in a 50 percent penalty. Failure to list people you worked with may result in no grade for this lab. Failure to fill out hours approximation will result in a 10-percent penalty.

Turn in your files to Gradescope

BEFORE YOU BEGIN

1. Review your knowledge of Fourier transforms. Here's a good place to start if you don't remember. Long story short, all signals can be approximated by sine waves (and if they are discrete signals, you can get a perfect reconstruction).

2. Remind yourself of what the term Nyquist frequency means in the context of sampled signals. A link is here. Long story short, the higher your sampling rate, the higher ranges of frequencies you can record in your signal.

3. Download the sinewave1000hz.wav and starwars.wav files on Canvas.

Learning Objectives:
The goal of this homework is to have you write an application that is actually does something useful. In this case, we're having you work with digital signal processing, but we could just as easily have you working with applications like dynamic system simulation, etc. We just chose music files because, well, who doesn't like music?

Thoughts (come back and read these after you've read the problems):

1. There are two parts to this assignment. The first part contains the logic for the digital signal processing. The second involves the GUI. The idea is that you can write the digital signal code and then import it inside of your GUI. Even though we tell you to write the digital signal stuff first, you should feel free to write methods in that may help you out for the GUI.

2. Remember that if you're copy-pasting code, there's usually a way you can refactor the code so that it's reusable. For instance, if you find that while making your GUI you're copy and pasting 5-line chunks of code multiple times, there's probably a way to condense those 5 lines into a single reusable command, which will make your life easier if you find out you did something wrong and have to go back and edit each of the chunks again. You may also find it useful to reuse some of the code in Lab 8.

3. You should sanity check your bandpass() function by running it on the provided sine wave file, which is 1000 Hz, and saving it back out. If 1000 Hz is in your frequency range, the signal should sound identical. If it's outside, you should hear silence.

Grading Checkpoints (20 points + 3 extra credit)

Discrete signal (10 pts): (Autograder will give immediate feedback)

1. Class properly initializes and sets the initial attributes correctly. **(2 points)***
2. `DigitalSignal.from_wav()` works. **(1 point)***
3. `set_time()` and `set_frequency_bounds()` work. **(1 point)***
4. `bandpass()` returns a correctly transformed signal (3 points) with integer values (1 point) and the correct time subset (1 point). **[5 points total]***
5. `save_wav()` works. **(1 point)***

GUI (10 points, but can earn up to 13): (Manual feedback)

6. We can load in a valid file. **(1 point)**
7. We can't crash your window during the course of normal operation. **(2 points)**
8. Some combination of the following features **(Can earn up to 10 points)**
    1. Four time sliders, two for time and two for frequency. The min and max should be set to the min and max of the respective parameter of the current audio file. Each slider should have a label showing the current value of the slider. **(2 points for freq, 2 points for time)**
    2. Some sort of useful message is displayed if there is something wrong with the file input to the load. (Printing to the console does not count as a useful message**.) (1 point)**

       Note: Even if you don't do this part, your code shouldn't crash when we hit the Load button.
    3. A save file field which saves the audio file with the values from the sliders. **(1 point)**
    4. A plot showing the audio signal of the loaded file **(2 points)**
    5. The plot updates when the sliders are moved. **(2 points)**
    6. The sliders cannot slide past each other, i.e. if the low bound exceeds the upper bound, the upper bound gets "pushed" along, and vice versa. **(2 points)**

Hand in this document as a PDF, along with your digital_signal.py and audio_gui.py file to Gradescope.

Problem 1 Signal Processing Class

Create a new file called **digital_signal.py**. All your code from Problem 1 and 2 should go in here.

The first part of this assignment will not involve the GUI at all; however, it will provide you with some useful interfaces to which you can connect the GUI actions.

1. **Write a class called `DigitalSignal`** whose __init__ method takes in two arguments: a 1-D integer Numpy array, and a positive integer representing the sampling frequency.
   - `max_freq`: The maximum frequency present in the signal (i.e. the Nyquist frequency)
   - `start`: Initialized to 0
   - `end`: Initialized to the final time value of your data, assuming the first sample is t=0
   - `bound_low`: Initialized to 0
   - `bound_high`: Initialized to the Nyquist frequency

2. **Write an *class* method called `from_wav()`** which takes in a single argument, a file name string. It should return a `DigitalSignal` with the data and sampling rate from the WAV file. You may assume that the WAV file is mono (so there will only be a single channel, returned by scipy's wavfile read function as a 1-D array of some length). However, for your own sake, if you want to work with stereo, you should extract simply extract the first channel to work with (we will not grade you on this).

   Note that with a class method, we should be able to call your code like this:

   ```
   my_signal = DigitalSignal.from_wav('music.wav')
   ```

Comments for grader/additional information (if any)

Problem 2 Signal Processing

1. **Write an instance method called `set_time()`**, which takes in two optional time arguments `start` and `end`. It should perform the following logic: If `start` is specified, it sets `self.start` to that value, but if not, it sets it to 0. Similarly, if `end` is specified, it should set `self.end` to that value, but if not, it should set `end` to the final time value.

   E.g.
   ```
   self.set_time(1.5, 2.4) # Goes from 1.5 to 2.4
   self.set_time(end=7.8)  # Goes from 0 to 7.8
   self.set_time()  # Effectively resets time
   ```

2. **Write a similar instance method called `set_frequency_bounds()`**, which takes in two optional arguments `low` and `high` and set `self.bound_low` and `self.bound_high` accordingly. If `low` is not specified, it should set the low bound to 0. If `high` is not specified, it should set the high bound to the Nyquist frequency.

   E.g.
   ```
   self.set_frequency_bounds(10, 3000)    # 10Hz to 3000Hz
   self.set_frequency_bounds(high=14000)  # 140000 Hz
   self.set_frequency_bounds()            # 0Hz to Nyquist freq
   ```

3. **Write an instance method called `bandpass()`** which takes in **no** arguments. It should first take the entire audio signal, compute its Fourier transform, zero out all frequency components with magnitude (strictly) less than `self.bound_low` and (strictly) greater than `self.bound_high`, and then compute the inverse Fourier transform. It should then return an array subset of the signal corresponding to the time in the interval [`self.start, self.end`]. Note that your return values should be real-valued integers.

   Note: For consistency, please use the fft-related functions from Scipy's fft module, not the Numpy versions.

   Hint: You may be interested in Scipy's fftfreq function, which returns the frequencies corresponding to each element of a FFT array.

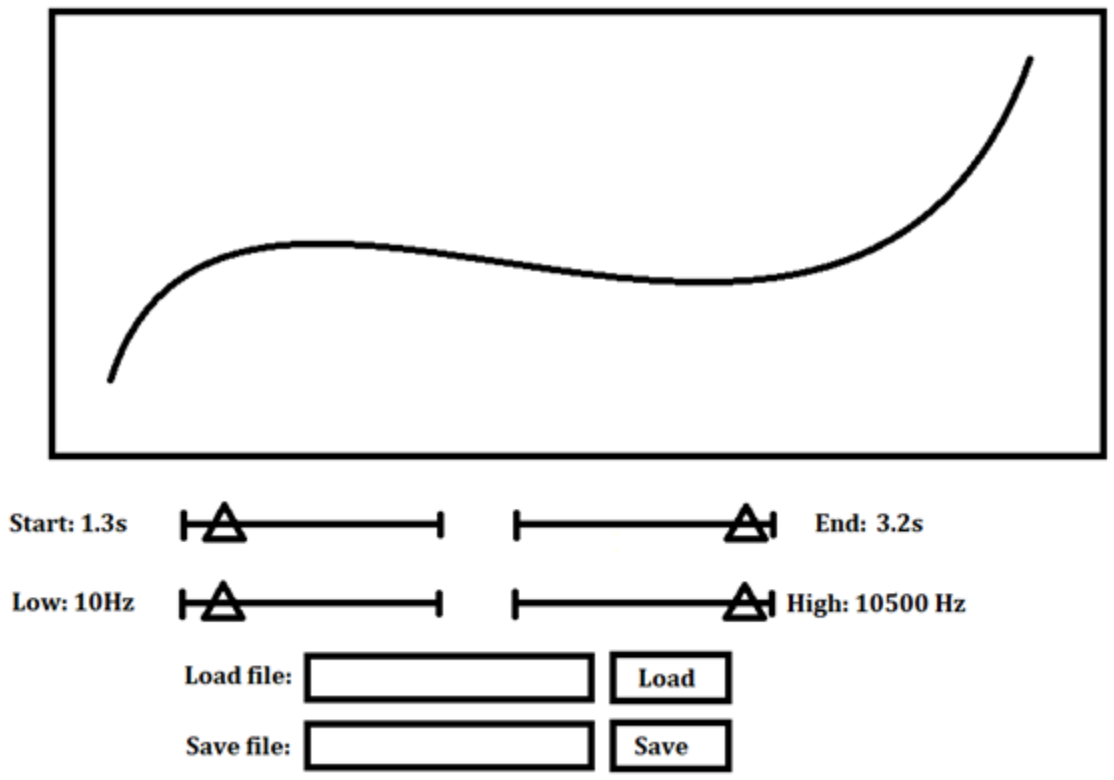
4. **Write an instance method called `save_wav()`** which takes in one argument, a filename, and saves the audio signal for the corresponding time and frequency bounds to the filename.

Comments for grader/additional information (if any)

Problem 3: GUI

Create a new file called **audio_gui.py**.

In this section, your goal is to implement a GUI which provides easy access to the features of the class. It will look something like the prototype below:



It does not need to look exactly like the above. It must have a Load button which reads user input from a form field and loads the file in. It also should not crash during the normal course of operation.

From there, **you may choose the following features to implement**. You only need to do 7 points worth of them to get full credit, but you can earn up to 10 points (i.e. 3 extra credit points). (Note that there are 12 points available total; if you do all 12, you will only get 10, but you can pat yourself on the back for a job well done.)

1. Four time sliders, two for time and two for frequency. The min and max should be set to the min and max of the respective parameter of the current audio file, so the start can go from 0 to max_time, the low from 0 to the Nyquist frequency, etc. These values should update each time we load a new file in.

   Each slider should have a label showing the current value of the slider, and the function of each slider should be obvious.

   (2 points for freq, 2 points for time)

2. Some sort of useful message is displayed if there is something wrong with the file name being loaded.

   Printing to the console does not count as a useful message. Also, there are two kinds of errors that can happen when loading a file; you should have a separate message for each. **(1 point)**

   Note that even if you don’t do this part, your code shouldn’t crash as a result of us clicking the Load button under any condition.

3. A save file field and button which saves the audio file to the desired name with the values from the sliders. **(1 point)**

4. A plot showing the audio signal of the loaded file **(2 points)**

5. The plot updates when the sliders are moved. **(2 points)**

   Note: For some inputs, the graph may update slowly. This is OK; we’re not grading you on having a speedy application (though it also shouldn’t take over 10 seconds on the supplied WAV files).

6. The sliders cannot slide past each other, i.e. if the low bound exceeds the upper bound, the upper bound gets “pushed” along, and vice versa. **(2 points)**

Comments for grader/additional information (if any)
GUI takes incredible amount of time to complete. This project should have multiple lab sections to complete or more guidance steps on how to create variables that update properly across multiple classs instances.