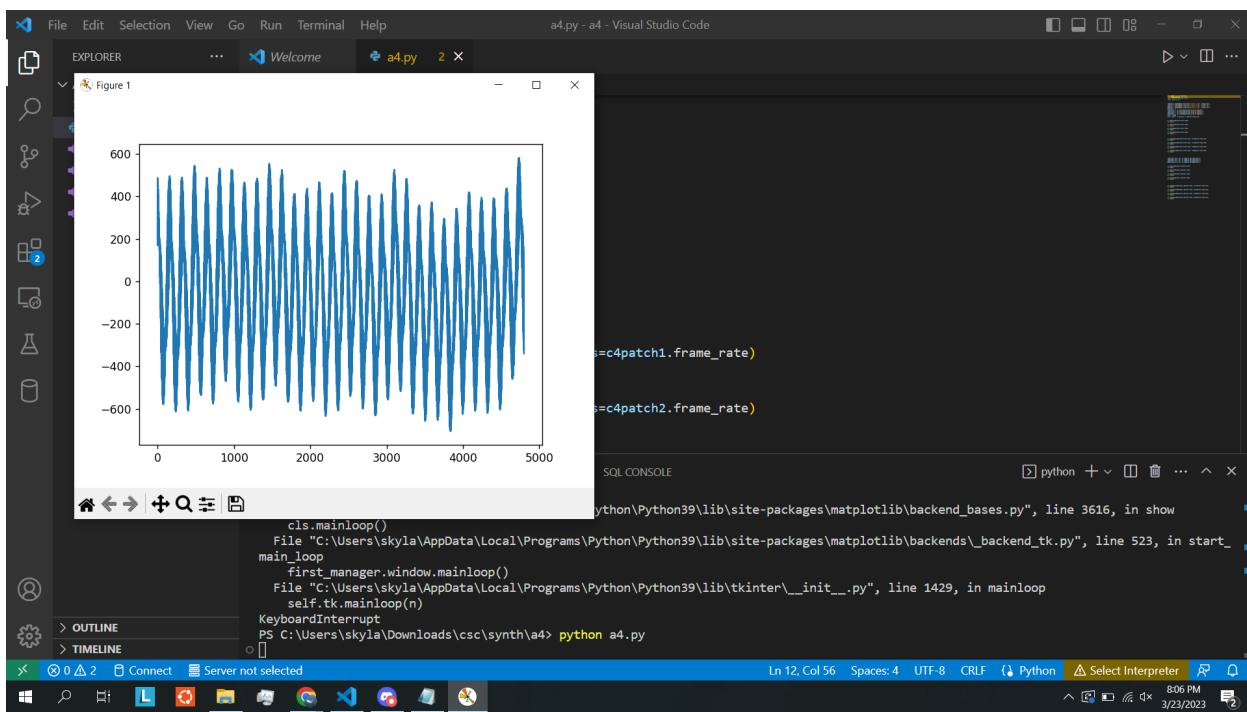
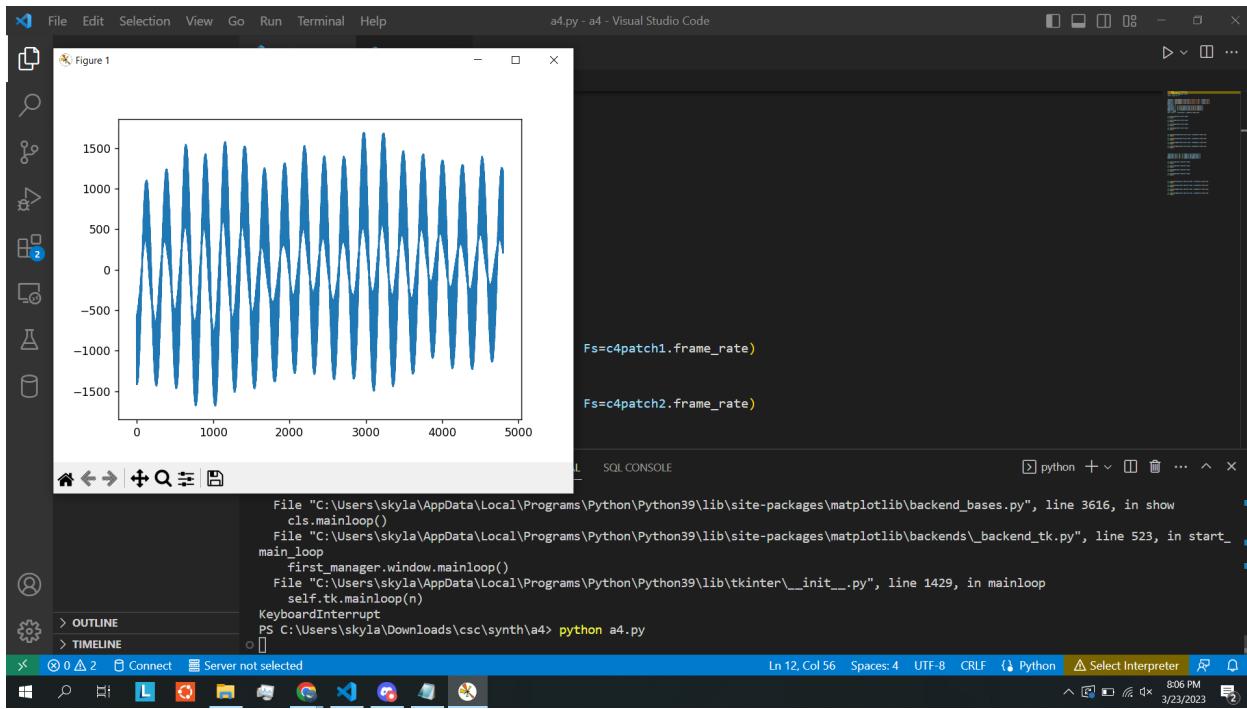
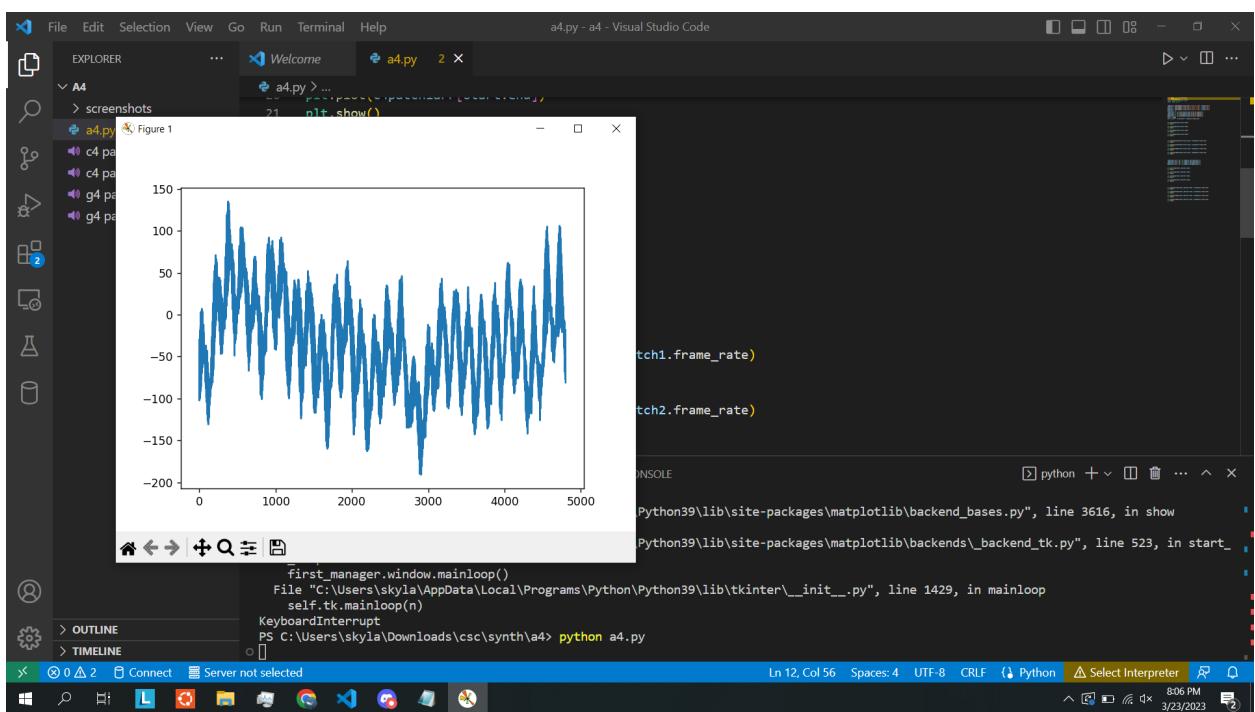
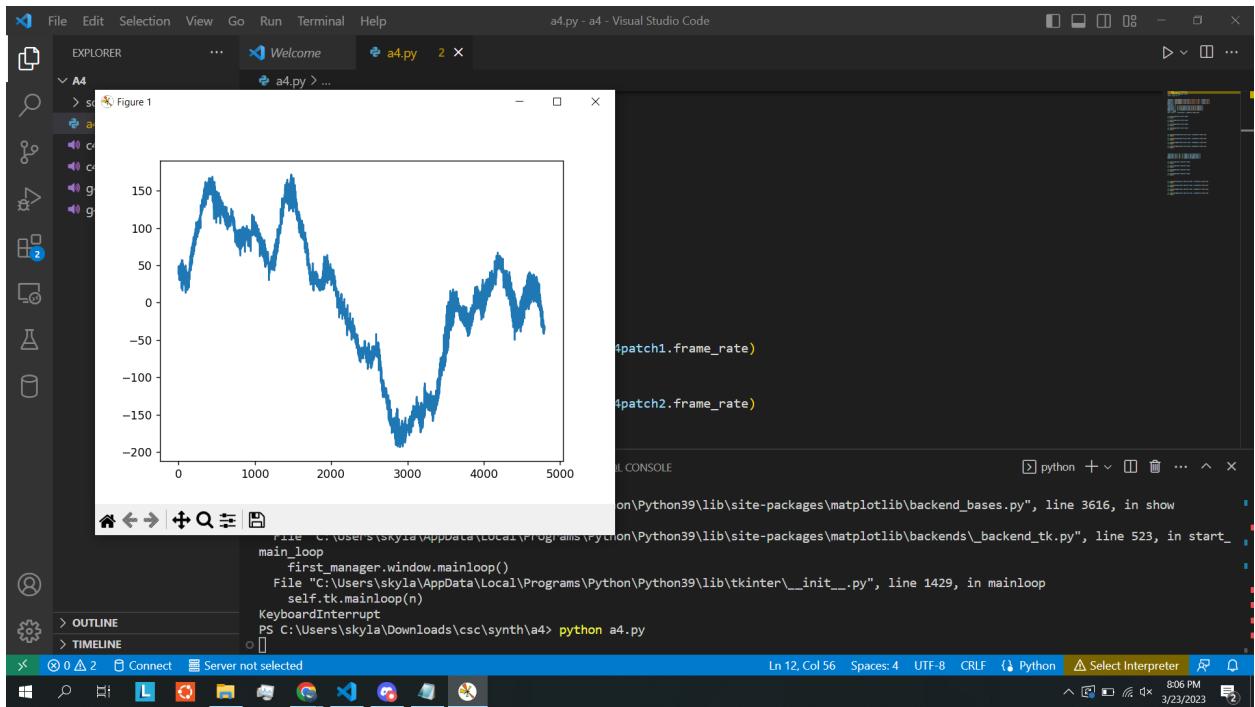


For question 1 Watch a video on YouTube about the DX 21 and make an explicit connection to any of the concepts we have covered in the course. I watched a video called "Yamaha DX21 Basics - Intro & Special Features" from the resources page on the github. The Yamaha DX21 synthesizer was a significant advancement in music technology during the 1980s, as it was the first affordable synthesizer to utilize Yamaha's FM synthesis technology and MIDI communication. One interesting fact about the Yamaha DX21 synthesizer is that it was the first synthesizer to use the newly-developed Yamaha FB01 sound generator chip, which allowed for a more affordable implementation of the FM synthesis technology used in the DX7. The DX21's affordability and versatility helped to democratize electronic music production, making it accessible to a wider range of musicians and producers. The DX21's impact on music technology is still felt today, as it paved the way for future advancements in synthesis and MIDI communication.





File Edit Selection View Go Run Terminal Help

a4.py - a4 - Visual Studio Code

EXPLORER ... Welcome a4.py 2 ×

A4

screenshots a4.py 2

c4 patch Figure 1

g4 patch

a4.py > ...

21 plt.show()

22

1.frame_rate)

2.frame_rate)

OLE

python + ...

File "C:\Users\skyla\AppData\Local\Programs\Python\Python39\lib\tkinter_init__.py", line 3616, in show

File "C:\Users\skyla\AppData\Local\Programs\Python\Python39\lib\site-packages\matplotlib\backend_bases.py", line 3616, in show

File "C:\Users\skyla\AppData\Local\Programs\Python\Python39\lib\site-packages\matplotlib\backends_backend_tk.py", line 523, in start_

self.tk.mainloop(n)

KeyboardInterrupt

PS C:\Users\skyla\Downloads\csc\synth\%a4% python a4.py

OUTLINE

TIMELINE

0 2 Connect Server not selected

Ln 12, Col 56 Spaces: 4 UTF-8 CRLF Python Select Interpreter

806 PM 3/23/2023

File Edit Selection View Go Run Terminal Help

a4.py - a4 - Visual Studio Code

EXPLORER ... Welcome a4.py 2 ×

A4

screenshots a4.py 2

c4 patch 1.mp3

g4 patch 1.mp3

a4.py > ...

21 plt.show()

22

23 plt.plot(c4patch2arr[start:end])

Frame_rate)

Frame_rate)

OLE

python + ...

File "C:\Users\skyla\AppData\Local\Programs\Python\Python39\lib\tkinter_init__.py", line 3616, in show

File "C:\Users\skyla\AppData\Local\Programs\Python\Python39\lib\site-packages\matplotlib\backend_bases.py", line 3616, in show

File "C:\Users\skyla\AppData\Local\Programs\Python\Python39\lib\site-packages\matplotlib\backends_backend_tk.py", line 523, in start_

self.tk.mainloop(n)

KeyboardInterrupt

PS C:\Users\skyla\Downloads\csc\synth\%a4% python a4.py

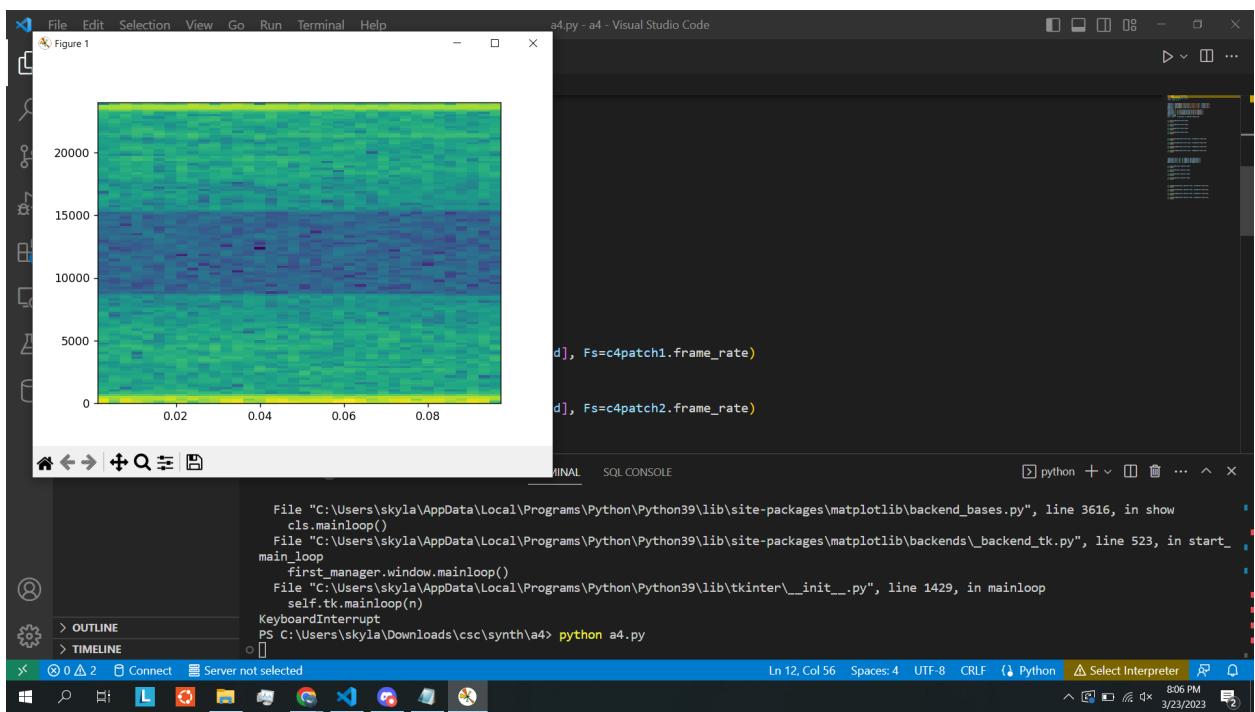
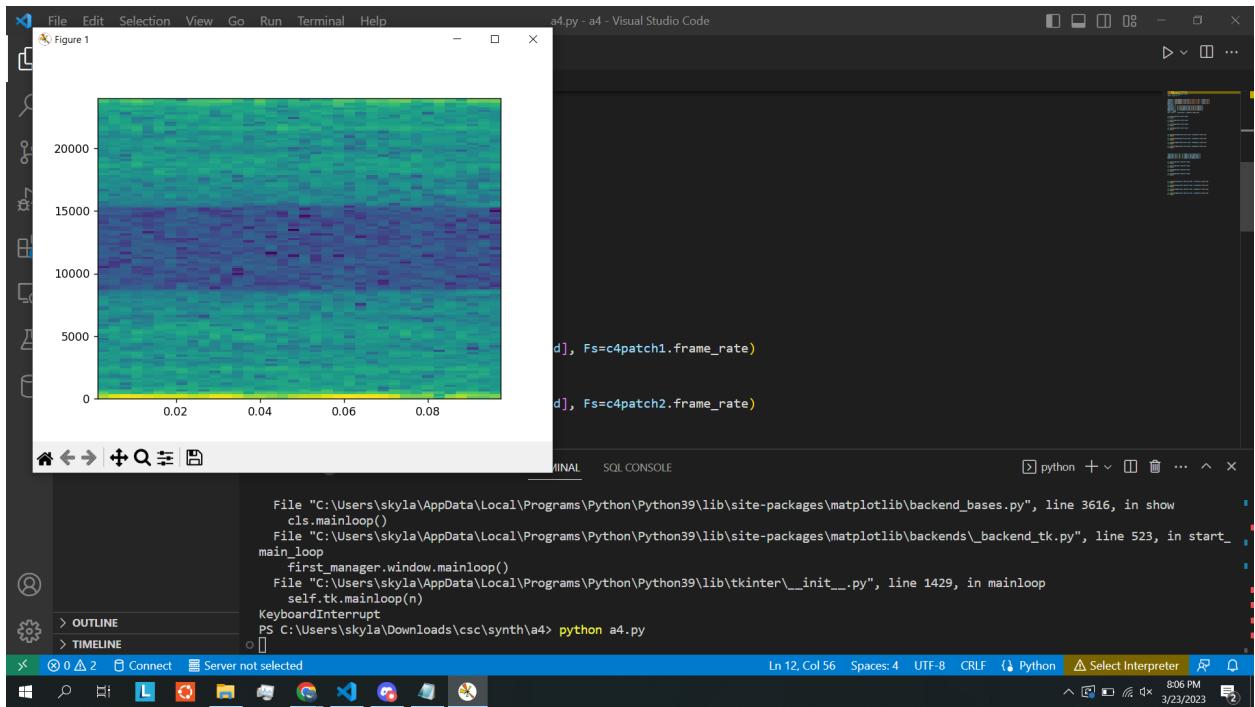
OUTLINE

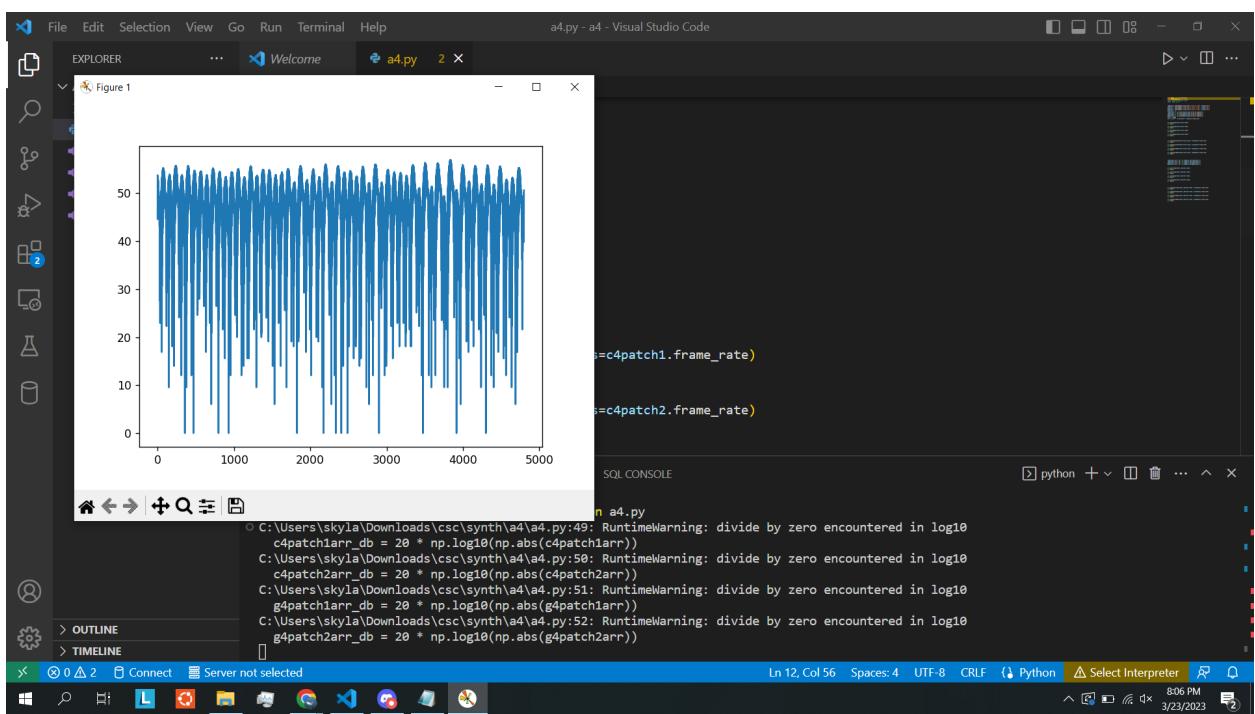
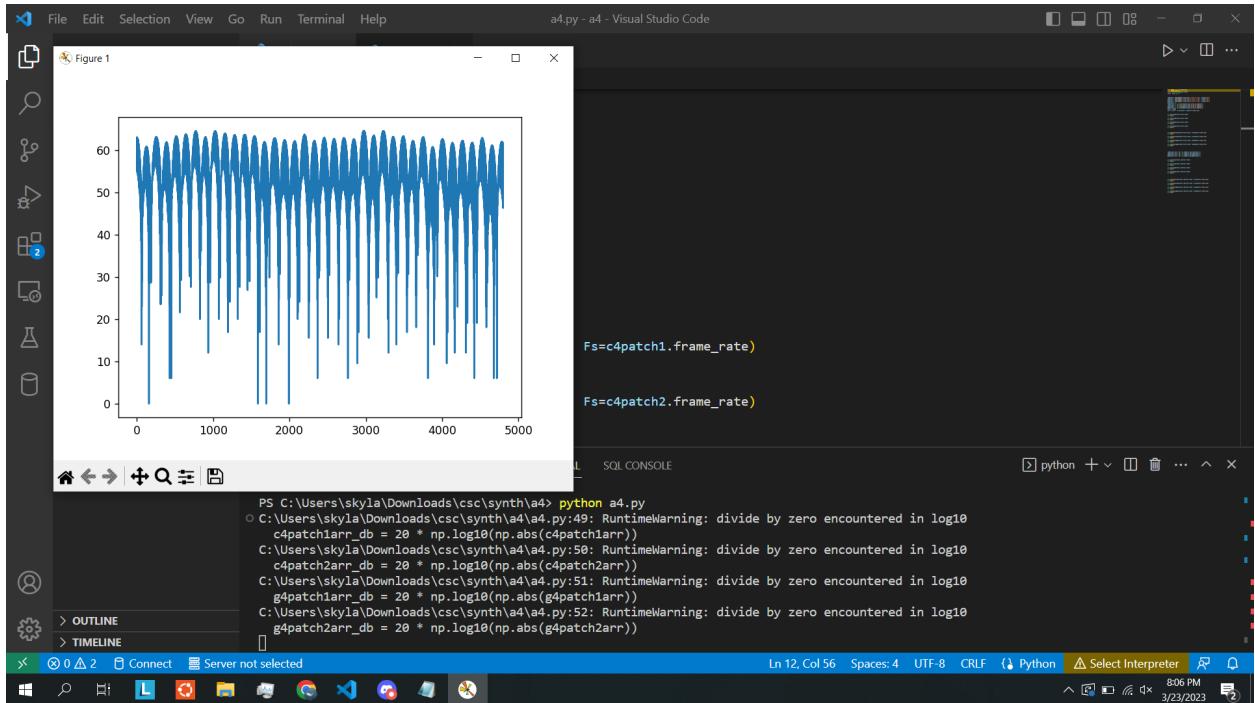
TIMELINE

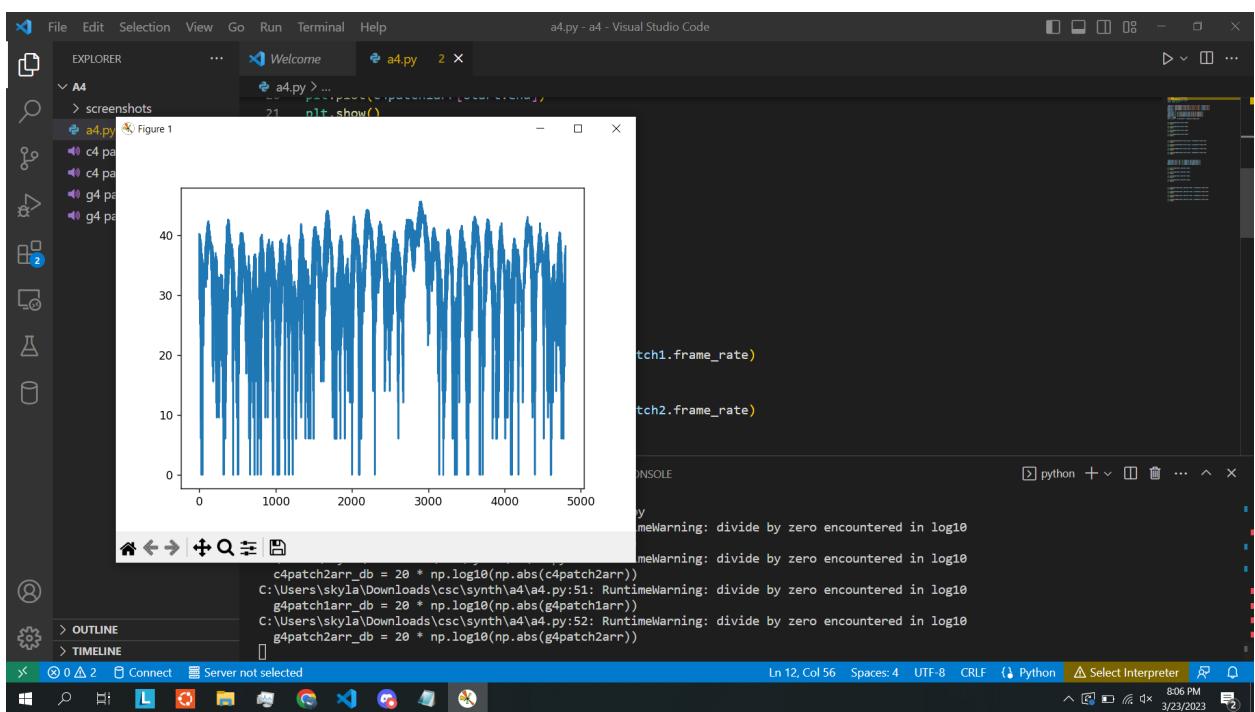
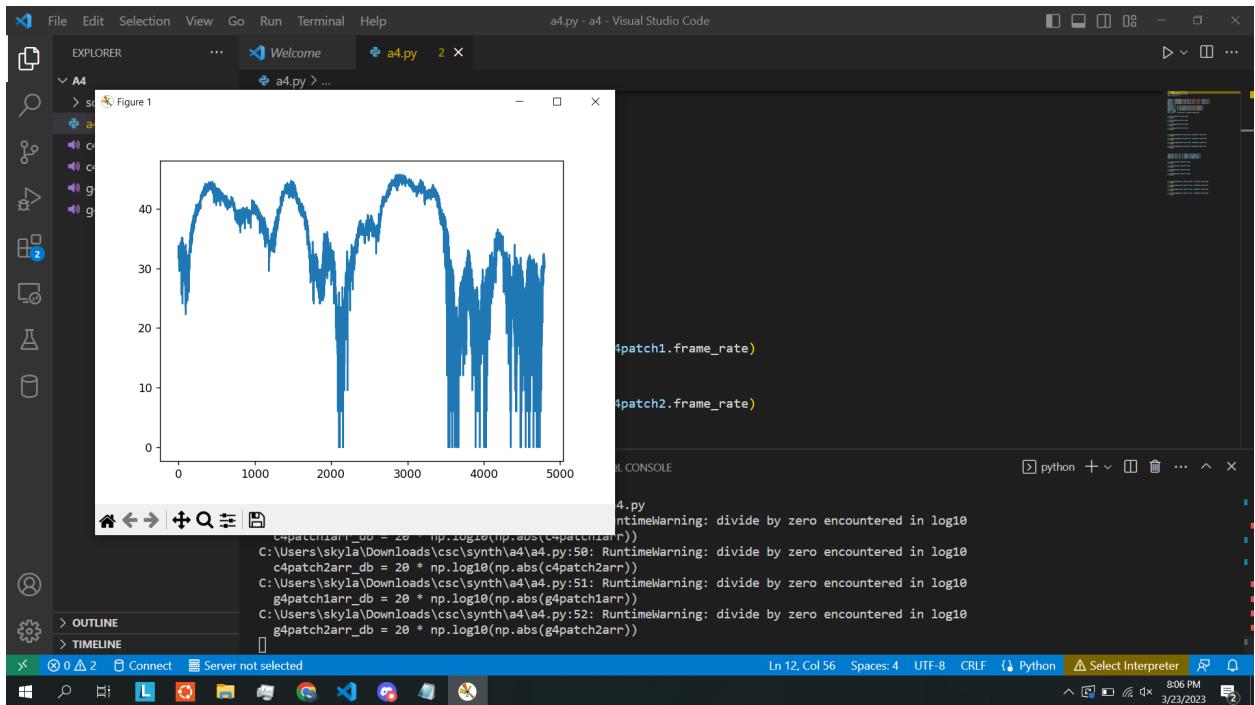
0 2 Connect Server not selected

Ln 12, Col 56 Spaces: 4 UTF-8 CRLF Python Select Interpreter

806 PM 3/23/2023







File Edit Selection View Go Run Terminal Help

a4.py - a4 - Visual Studio Code

EXPLORER ... Welcome a4.py 2 ×

a4.py > ...

21 plt.show()

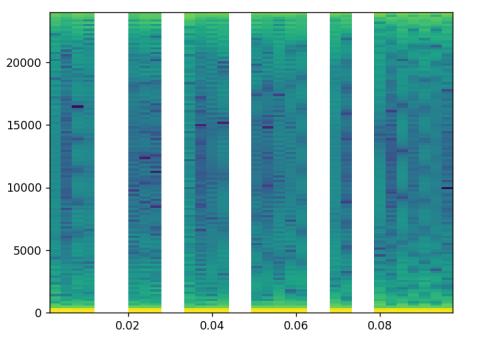
22

c4 patch Figure 1

c4 patch

g4 patch

g4 patch



1.frame_rate)

2.frame_rate)

OLE

b\site-packages\matplotlib\mlab.py:396: RuntimeWarning: invalid value encountered

b\site-packages\matplotlib\mlab.py:418: RuntimeWarning: invalid value encountered

result[slc] *= scaling_factor

C:\Users\skylla\AppData\Local\Programs\Python\Python39\lib\site-packages\matplotlib\mlab.py:424: RuntimeWarning: invalid value encountered

d in divide

result /= Fs

0 2 Connect Server not selected

Ln 12, Col 56 Spaces: 4 UTF-8 CRLF Python Select Interpreter

806 PM 3/23/2023

File Edit Selection View Go Run Terminal Help

a4.py - a4 - Visual Studio Code

EXPLORER ... Welcome a4.py 2 ×

a4.py > ...

21 plt.show()

22

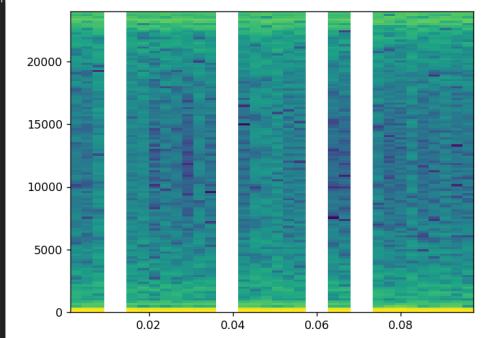
23 plt.plot(c4patch2arr[start:end])

c4 patch 1.mp3

c4 patch 2.mp3

g4 patch 1.mp3

g4 patch 2.mp3



Frame_rate)

Frame_rate)

site-packages\matplotlib\mlab.py:418: RuntimeWarning: invalid value encountered

site-packages\matplotlib\mlab.py:424: RuntimeWarning: invalid value encountered

C:\Users\skylla\AppData\Local\Programs\Python\Python39\lib\site-packages\matplotlib\mlab.py:384: RuntimeWarning: invalid value encountered

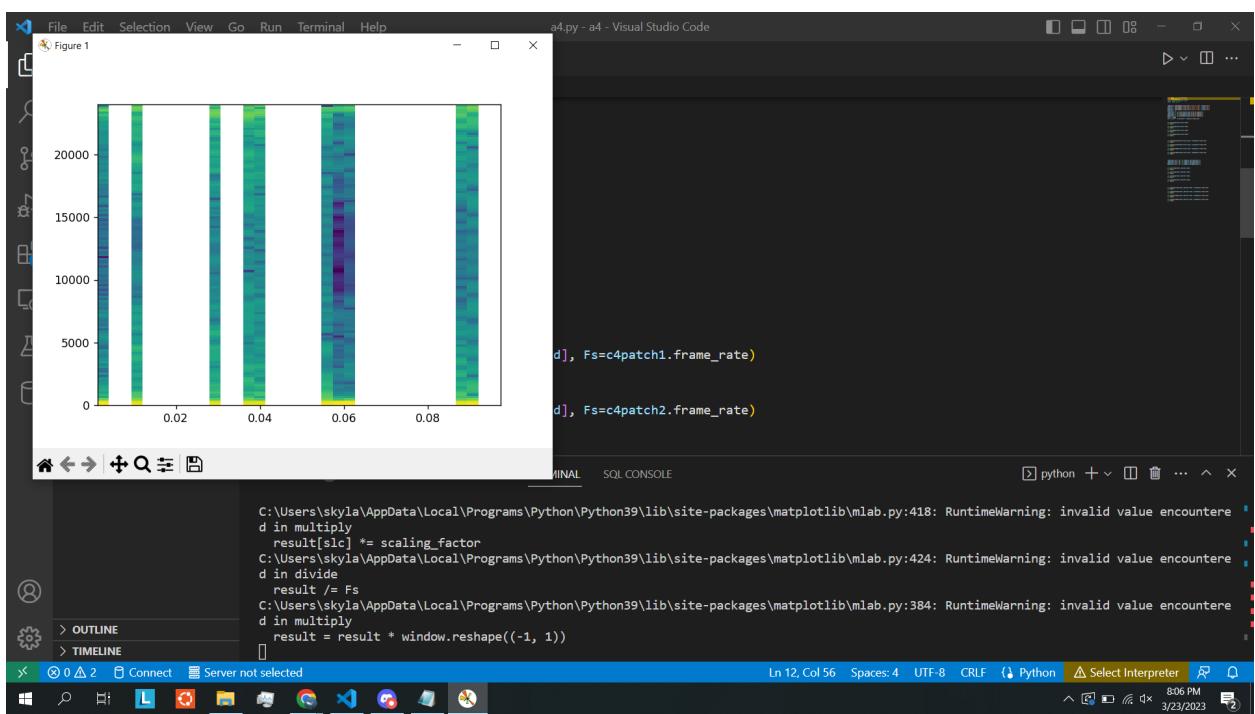
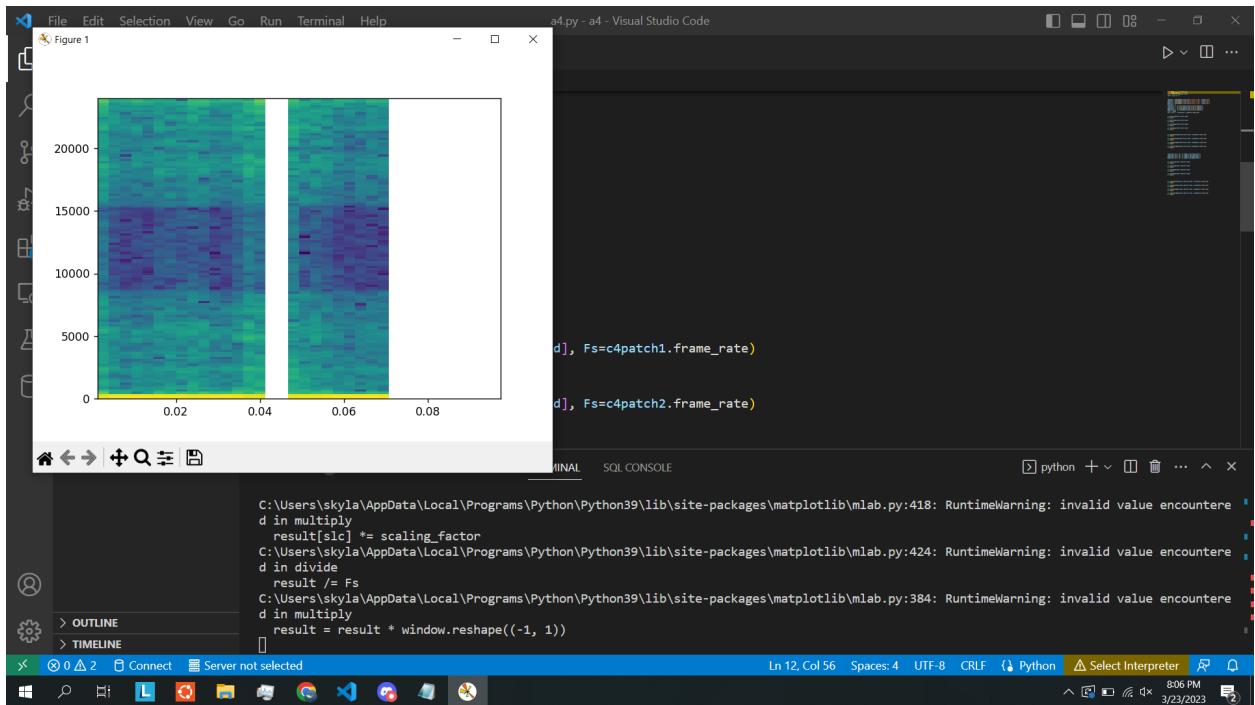
d in multiply

result = result * window.reshape((-1, 1))

0 2 Connect Server not selected

Ln 12, Col 56 Spaces: 4 UTF-8 CRLF Python Select Interpreter

806 PM 3/23/2023



The screenshot shows the Visual Studio Code interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Title Bar:** a4.py - a4 - Visual Studio Code
- Explorer:** Shows a folder named 'A4' containing 'screenshots', 'a4.py', and several MP3 files: 'c4 patch 1.mp3', 'c4 patch 2.mp3', 'g4 patch 1.mp3', and 'g4 patch 2.mp3'. There are two tabs open in the editor.
- Editor:** The active tab contains the following Python code:

```
1  from pydub import AudioSegment
2  from pydub.playback import play
3  import matplotlib.pyplot as plt
4  import numpy as np
5
6
7  c4patch1 = AudioSegment.from_file("c4 patch 1.mp3", format="mp3")
8  c4patch2 = AudioSegment.from_file("c4 patch 2.mp3", format="mp3")
9  g4patch1 = AudioSegment.from_file("g4 patch 1.mp3", format="mp3")
10 g4patch2 = AudioSegment.from_file("g4 patch 2.mp3", format="mp3")
11 duration = .1
12 c4patch1arr = np.array(c4patch1.get_array_of_samples())
13 c4patch2arr = np.array(c4patch2.get_array_of_samples())
14 g4patch1arr = np.array(g4patch1.get_array_of_samples())
15 g4patch2arr = np.array(g4patch2.get_array_of_samples())
16 start = 40000
17 end = start + int(duration * c4patch1.frame_rate)
18
19
```
- Terminal:** Shows command-line output with several RuntimeWarning messages related to invalid values in arrays.
- Bottom Status Bar:** Ln 12, Col 56, Spaces: 4, UTF-8, CRLF, Python, Select Interpreter, 806 PM, 3/23/2023.

The screenshot shows the Visual Studio Code interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Editor:** The main editor window displays the file `a4.py` with the following code:

```

23     plt.plot(c4patch2arr[start:end])
24     plt.show()
25
26     plt.plot(g4patch1arr[start:end])
27     plt.show()
28
29     plt.plot(g4patch2arr[start:end])
30     plt.show()
31
32
33
34     plt.specgram(c4patch1arr[start:end], Fs=c4patch1.frame_rate)
35     plt.show()
36
37     plt.specgram(c4patch2arr[start:end], Fs=c4patch2.frame_rate)
38     plt.show()
39
40     plt.specgram(g4patch1arr[start:end], Fs=g4patch1.frame_rate)
41     plt.show()

```

- Terminal:** The terminal tab shows command-line output related to matplotlib warnings.
- Bottom Status Bar:** Shows the current file is `a4.py`, the line number is 12, column 56, and the date is 3/23/2023.

The screenshot shows the Visual Studio Code interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Editor:** The main editor window displays the file `a4.py` with the following code:

```

48
49     c4patch1arr_db = 20 * np.log10(np.abs(c4patch1arr))
50     c4patch2arr_db = 20 * np.log10(np.abs(c4patch2arr))
51     g4patch1arr_db = 20 * np.log10(np.abs(g4patch1arr))
52     g4patch2arr_db = 20 * np.log10(np.abs(g4patch2arr))
53
54
55     plt.plot(c4patch1arr_db[start:end])
56     plt.show()
57
58     plt.plot(c4patch2arr_db[start:end])
59     plt.show()
60
61     plt.plot(g4patch1arr_db[start:end])
62     plt.show()
63
64     plt.plot(g4patch2arr_db[start:end])
65     plt.show()
66

```

- Terminal:** The terminal tab shows command-line output related to matplotlib warnings.
- Bottom Status Bar:** Shows the current file is `a4.py`, the line number is 12, column 56, and the date is 3/23/2023.

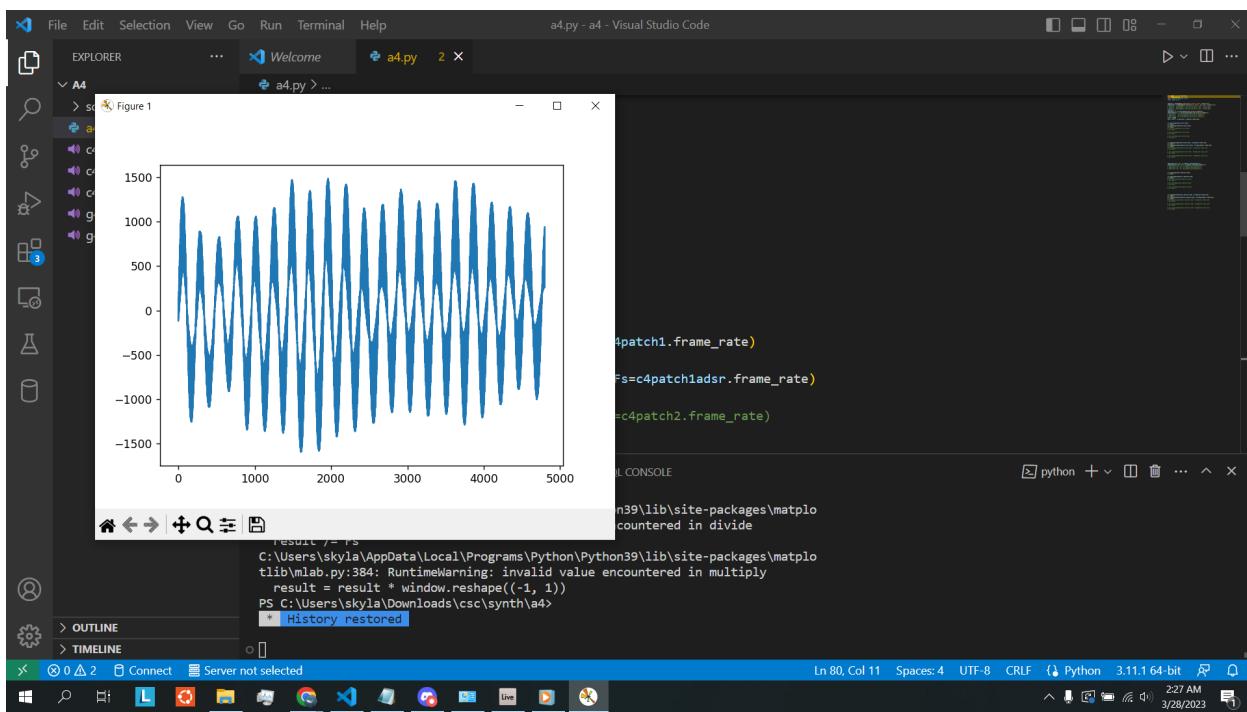
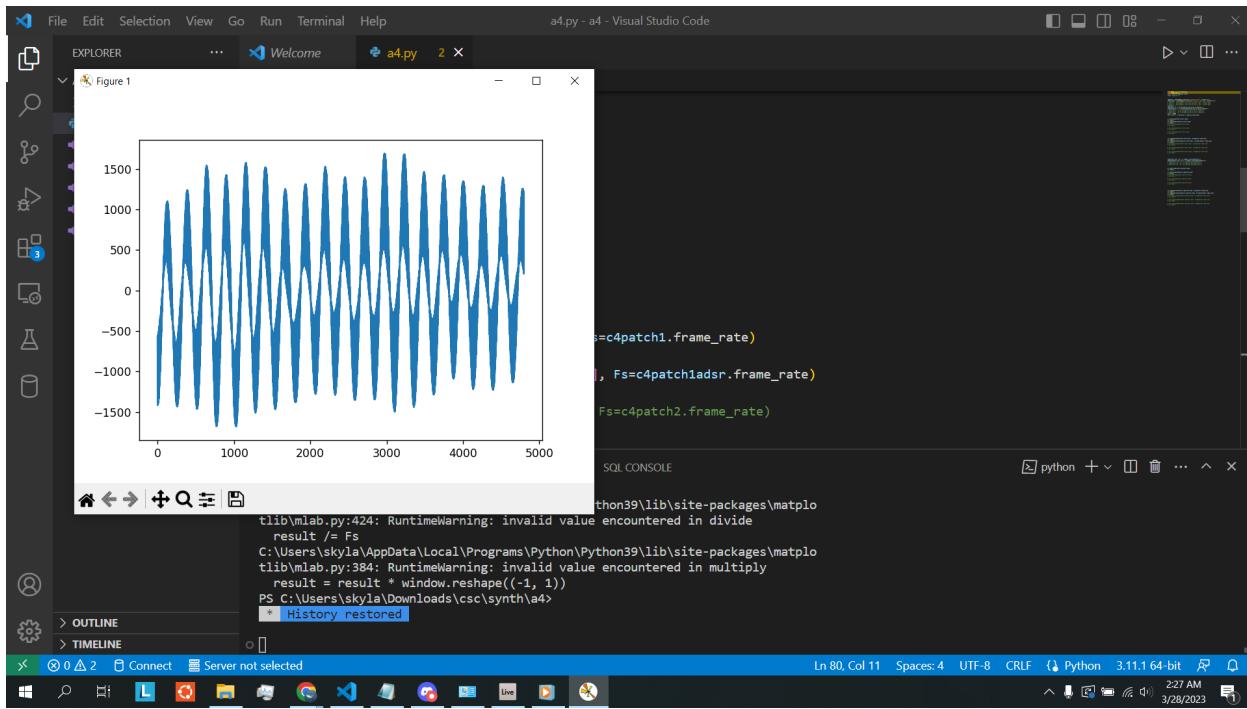
For question 2 I had to record four notes. two of C4 and the other two of G4. The reason I have two of each note is because they are from two different patches of the Yamaha DX21. I used the Yamaha DX21 that is in ECS602. I have to graph the time domain plots and magnitude spectrograms of these “four” notes in linear and decibel amplitudes. I used a very short duration that corresponds to a few pitch cycles of the notes. The graph screenshots show a cycle of C4 note 1, C4 note 2, G4 note 1, and then G4 note 2. Using this cycle you can see, for each note, the time domain plots for linear amplitude, then magnitude spectrograms for linear amplitude,

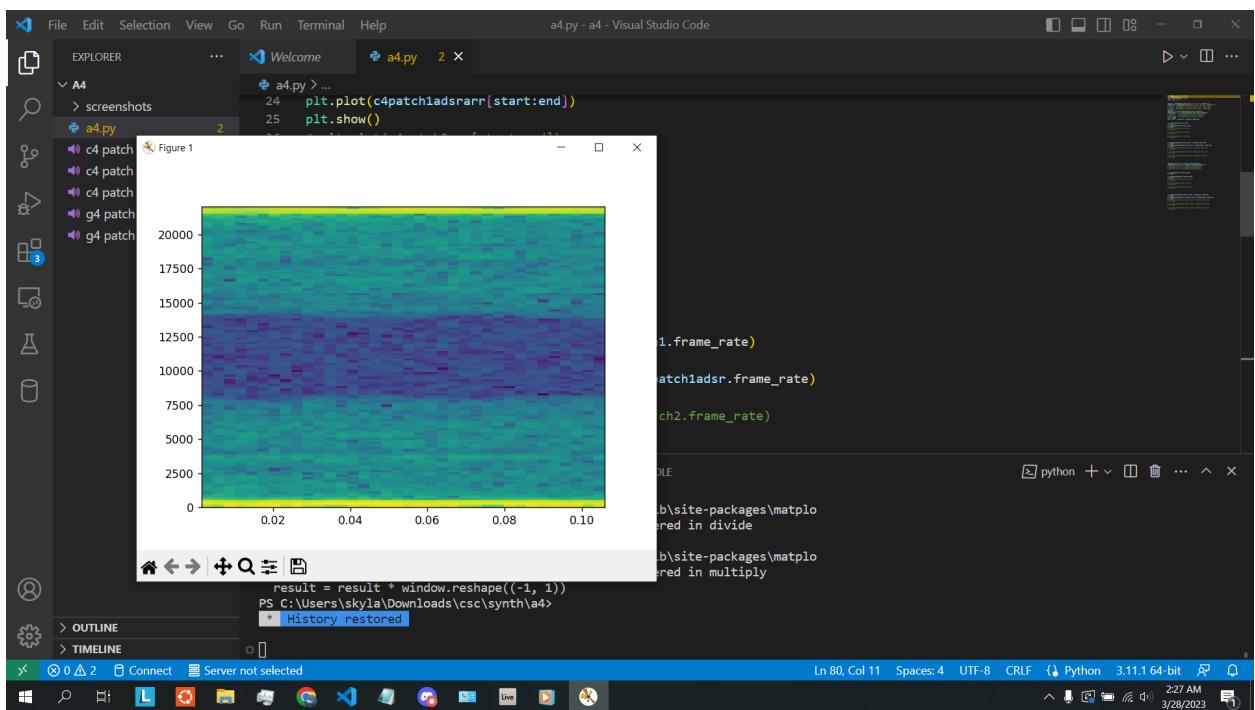
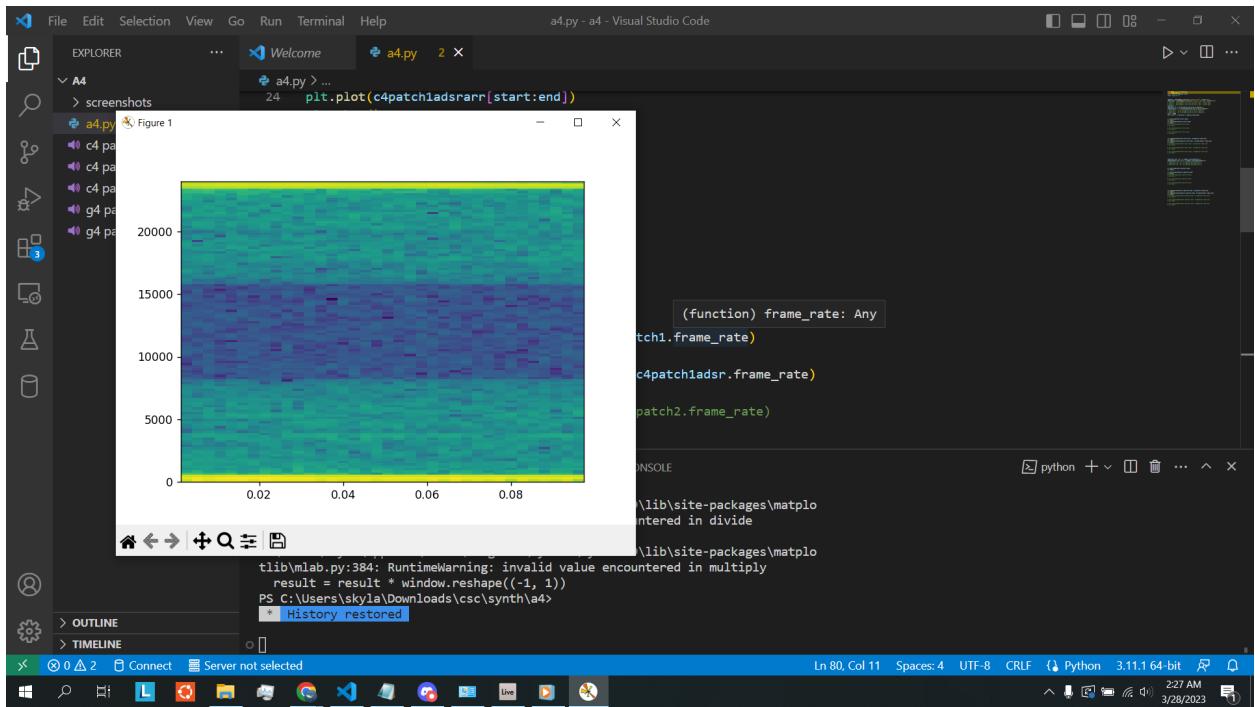
then the time domain plots in decibel amplitude, and finally the magnitude spectrograms in decibel amplitude. In terms of the code shown on the other screenshots, the process was to load the mp3 files using pydub and define the short duration of each note in seconds. The short duration I chose to correspond with a few pitch cycles of the notes was .1 seconds. Then after I convert the mp3 files to numpy arrays I can get a starting and ending spot for the sample of the duration I set earlier. After that I can plot the time domain signals using linear amplitude. I can also plot the magnitude spectrograms for linear amplitude as well using ".specgram". Then after converting the amplitude to dB I can plot the time domain signals and magnitude spectrograms using dB amplitude and the same process of utilizing ".plot" and ".specgram".



For question 3 I have to get the two FM algorithms that are used for each of my sound patches from the previous question where I used the DX21 synthesizer. To learn more about the FM algorithms I used the manual for the DX21 synthesizer and some discussion sites that can be found online for research. Of the DX21 synthesizer's available algorithms for FM synthesis, I found that I was using Algorithms 5 and 1. Algorithms for the DX21 synthesizer are typically referred to by their number designation in most documentation and discussions. They do not have specific names like some other synthesizers may give to their algorithms. Conversely I have found that some users may refer to what they do specifically, such as calling them the percussive algorithm or the pad algorithm. Each algorithm uses a different set of operator connections. Algorithm 1 consists of two operators: Operator 1 acts as the modulator, and

Operator 2 acts as the carrier. In this algorithm, the output of Operator 1 is fed into the frequency modulation input of Operator 2. Algorithm 5, on the other hand, is a more complex algorithm that uses four operators. Operators 1 and 2 act as modulators, while Operators 3 and 4 act as carriers. In this algorithm, the output of Operator 1 is fed into the frequency modulation input of Operator 2, which in turn is fed into the frequency modulation input of Operator 3. The output of Operator 2 is also fed into the frequency modulation input of Operator 4. The manual for the DX21 synthesizer and the discussion sites I browsed were very helpful for getting the information I needed and then clarifying it so it was more digestible.





File Edit Selection View Go Run Terminal Help

EXPLORER A4

a4.py > ... a4.py 2 x

```
24 plt.plot(c4patch1adsrarr[start:end])
25 plt.show()
26 # plt.plot(c4patch2arr[start:end])
```

Figure 1

Frame_rate)

c4patch1adsr.frame_rate)

c4patch2arr.frame_rate)

in multiply

x=4545, y=41.2 warning: divide by zero encountered in log10

C:\Users\skyla\Downloads\csc\synth\aa4\aa4.py:54: RuntimeWarning: divide by zero encountered in log10

c4patch1adsrarr_db = 20 * np.log10(np.abs(c4patch1adsrarr))

Line 80, Col 11 Spaces: 4 UTR-8 CRLF Python 3.11.1 64-bit 2:27 AM 3/28/2023

File Edit Selection View Go Run Terminal Help

Figure 1

```
d])
)
)
)

d], Fs=c (parameter) Fs: float
t:end], Fs=c4patch1adsr.frame_rate)
end], Fs=c4patch2.frame_rate)
```

tlib\mlab.py:384: RuntimeWarning: invalid value encountered in multiply
result = result * window.reshape((-1, 1))
PS C:\Users\skyla\Downloads\csc\synth\aa4\aa4.py:53: RuntimeWarning: divide by zero encountered in log10
c4patch1arr_db = 20 * np.log10(np.abs(c4patch1arr))
C:\Users\skyla\Downloads\csc\synth\aa4\aa4.py:54: RuntimeWarning: divide by zero encountered in log10
c4patch1adsrarr_db = 20 * np.log10(np.abs(c4patch1adsrarr))

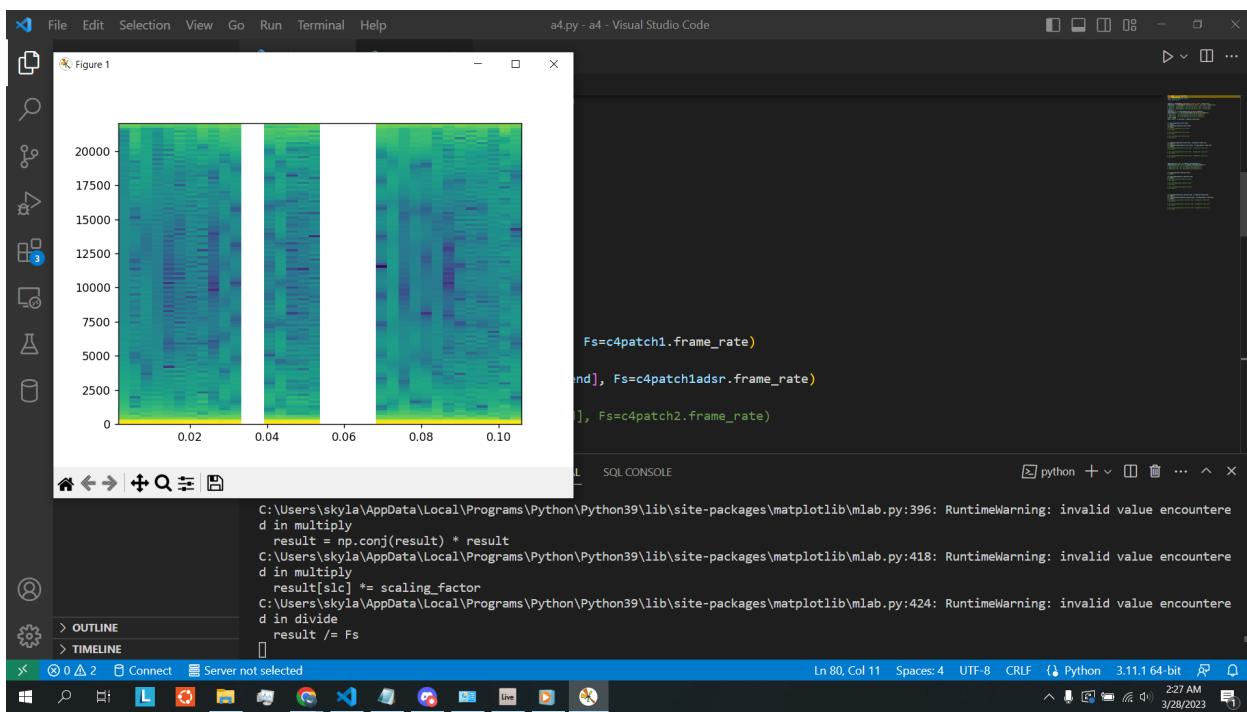
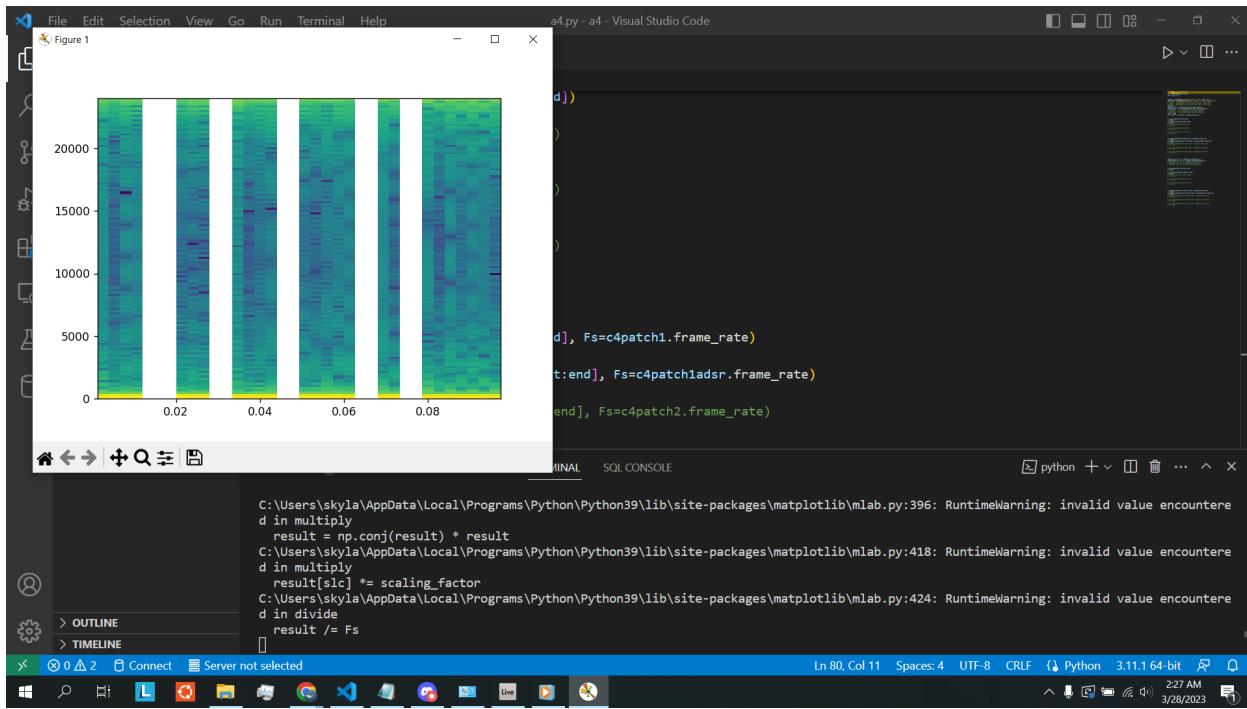
* History restored

SQL CONSOLE

x=4545, y=41.2 warning: divide by zero encountered in log10

C:\Users\skyla\Downloads\csc\synth\aa4\aa4.py:54: RuntimeWarning: divide by zero encountered in log10
c4patch1adsr.frame_rate)

Line 80, Col 11 Spaces: 4 UTR-8 CRLF Python 3.11.1 64-bit 2:27 AM 3/28/2023



The screenshot shows a Visual Studio Code interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Editor:** The main area displays the code for `a4.py`.

```
1  from pydub import AudioSegment
2  from pydub.playback import play
3  import matplotlib.pyplot as plt
4  import numpy as np
5
6
7  c4patch1 = AudioSegment.from_file("c4 patch 1.mp3", format="mp3")
8  c4patch1adsr = AudioSegment.from_file("c4 patch 1 adsr.mp3", format="mp3")
9  # c4patch2 = AudioSegment.from_file("c4 patch 2.mp3", format="mp3")
10 # g4patch1 = AudioSegment.from_file("g4 patch 1.mp3", format="mp3")
11 # g4patch2 = AudioSegment.from_file("g4 patch 2.mp3", format="mp3")
12 duration = .1
13 c4patch1arr = np.array(c4patch1.get_array_of_samples())
14 c4patch1adsrarr = np.array(c4patch1adsr.get_array_of_samples())
15 # c4patch2arr = np.array(c4patch2.get_array_of_samples())
16 # g4patch1arr = np.array(g4patch1.get_array_of_samples())
17 # g4patch2arr = np.array(g4patch2.get_array_of_samples())
18 start = 4000
19 end = start + int(duration * c4patch1.frame_rate)
```
- Terminal:** Shows runtime warnings from `mlab.py`:

```
C:\Users\skyla\AppData\Local\Programs\Python\Python39\lib\site-packages\matplotlib\mlab.py:396: RuntimeWarning: invalid value encountered in multiply
    result = np.conj(result) * result
C:\Users\skyla\AppData\Local\Programs\Python\Python39\lib\site-packages\matplotlib\mlab.py:418: RuntimeWarning: invalid value encountered in multiply
    result[slc] *= scaling_factor
C:\Users\skyla\AppData\Local\Programs\Python\Python39\lib\site-packages\matplotlib\mlab.py:424: RuntimeWarning: invalid value encountered in divide
    result /= Fs
```
- Bottom Status Bar:** In 80, Col 11, Spaces: 4, UTF-8, CRLF, Python 3.11.1 64-bit, 2:27 AM, 3/28/2023.

The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Title Bar:** a4.py - a4 - Visual Studio Code
- Explorer:** A4, screenshots, a4.py (2), c4 patch 1 adsr.mp3, c4 patch 1.mp3, c4 patch 2.mp3, g4 patch 1.mp3, g4 patch 2.mp3.
- Editor:** Code editor showing a4.py with the following content:

```
21 plt.plot(c4patch1arr[start:end])
22 plt.show()
23 plt.plot(c4patch1adsrarr[start:end])
24 plt.show()
25 # plt.plot(c4patch2arr[start:end])
26 # plt.show()
27 # plt.plot(g4patch1arr[start:end])
28 # plt.show()
29 # plt.plot(g4patch2arr[start:end])
30 # plt.show()
31
32 plt.specgram(c4patch1arr[start:end], Fs=c4patch1.frame_rate)
33 plt.show()
34
35
36
37 plt.specgram(c4patch1arr[start:end], Fs=c4patch1adsr.frame_rate)
38 plt.show()
39 plt.specgram(c4patch1adsrarr[start:end], Fs=c4patch1adsr.frame_rate)
```

- Terminal:** powershell +, SQL CONSOLE.
- Bottom Status Bar:** In 80, Col 11, Spaces: 4, UTF-8, CRLF, Python 3.11.1 64-bit, 2:27 AM, 3/28/2023.

The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Title Bar:** a4.py - a4 - Visual Studio Code
- Explorer:** A4, screenshots, a4.py (2), c4 patch 1 adsr.mp3, c4 patch 1.mp3, c4 patch 2.mp3, g4 patch 1.mp3, g4 patch 2.mp3.
- Editor:** Code editor showing a4.py with the following content:

```
51
52
53 c4patch1arr_db = 20 * np.log10(np.abs(c4patch1arr))
54 c4patch1adsrarr_db = 20 * np.log10(np.abs(c4patch1adsrarr))
55 # c4patch2arr_db = 20 * np.log10(np.abs(c4patch2arr))
56 # g4patch1arr_db = 20 * np.log10(np.abs(g4patch1arr))
57 # g4patch2arr_db = 20 * np.log10(np.abs(g4patch2arr))
58
59
60 plt.plot(c4patch1arr_db[start:end])
61 plt.show()
62
63 plt.plot(c4patch1adsrarr_db[start:end])
64 plt.show()
65 # plt.plot(c4patch2arr_db[start:end])
66 # plt.show()
67
68 # plt.plot(g4patch1arr_db[start:end])
69 # plt.show()
```

- Terminal:** powershell +, SQL CONSOLE.
- Bottom Status Bar:** In 80, Col 11, Spaces: 4, UTF-8, CRLF, Python 3.11.1 64-bit, 2:27 AM, 3/28/2023.

For question 4 I used one of the patches that I had used previously. I kept the algorithm the same but changed the adsr to have a much lower sustain. Using the code from question 2 I can show the time-domain waveform and magnitude spectrum before the changes and after the changes. I used two C4 note sounds from the first patch but with different adsr. I used a very short duration that corresponds to a few pitch cycles of the notes. The graph screenshots show a back and forth cycle of the former C4 note 1 and the C4 note 2 which has a changed adsr. With this cycle you can see, for each note, the time domain plots for linear amplitude, then

magnitude spectrograms for linear amplitude, then the time domain plots in decibel amplitude, and finally the magnitude spectrograms in decibel amplitude. In terms of the code shown on the other screenshots, the process was to load the mp3 files using pydub and define the short duration of each note in seconds. The short duration I chose to correspond with a few pitch cycles of the notes was .1 seconds. Then after I convert the mp3 files to numpy arrays I can get a starting and ending spot for the sample of the duration I set earlier. After that I can plot the time domain signals using linear amplitude. I can also plot the magnitude spectrograms for linear amplitude as well using “.specgram”. Then after converting the amplitude to dB I can plot the time domain signals and magnitude spectrograms using dB amplitude and the same process of utilizing “.plot” and “.specgram”.

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows a folder named "A4" containing files: screenshots, a4.py, c4 patch 1 adsr.mp3, c4 patch 1.mp3, c4 patch 2.mp3, g4 patch 1.mp3, g4 patch 2.mp3, and q5.py.
- Code Editor:** Displays the content of q5.py. The code uses PyAudio to generate a sine wave at a specified frequency based on a MIDI note input. It includes imports for pyaudio, numpy, and sys, and handles command-line arguments for note selection.
- Terminal:** Shows the command PS C:\Users\skyla\Downloads\csc\synth\A4> python q5.py 60 being run, which generates a sine wave at note A4 (440 Hz).
- Status Bar:** Shows the file is 24 columns wide, 4 spaces per tab, in UTF-8 encoding, using Python 3.11.1 64-bit, and was last updated on 3/28/2023 at 4:50 PM.

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the same folder structure as the first screenshot.
- Code Editor:** Displays the continuation of q5.py. The code calculates the frequency from a note number (note - 69) divided by 12.0, generates a sine wave using np.sin, and writes it to the stream. It then stops the stream, closes it, and terminates the PyAudio instance.
- Terminal:** Shows the command PS C:\Users\skyla\Downloads\csc\synth\A4> python q5.py 60 being run again, generating a sine wave at note A4.
- Status Bar:** Shows the file is 24 columns wide, 4 spaces per tab, in UTF-8 encoding, using Python 3.11.1 64-bit, and was last updated on 3/28/2023 at 4:50 PM.

For question 5 I had to create a real-time synth in python. The synth has a single sinusoidal oscillator and takes midi note numbers from the command line as input. Based on the code shown in the code, the process of creating this real-time synth goes as follows. Set up PyAudio stream parameters, the PyAudio stream, and the synth parameters. Initialize the starting frequency and the gain of oscillator. Once all this is complete we can check for command line input and calculate frequency based on MIDI note number. Using this input we generate audio

output for x number of seconds and once we are done generating that sound we close PyAudio stream and the program concludes.

Visual Studio Code interface showing the file `q5.py` (a4 - Visual Studio Code). The code implements fmssynthesis using NumPy and PyAudio:

```
q5.py > fmssynthesis
1 import pyaudio
2 import numpy as np
3 #import time
4 import argparse
5
6
7 rate1 = 44100
8 buffer1 = 1024
9 duration = 4.0
10 attack = 0.1
11 decay = 0.2
12 #waveform=0
13 #t=lineform(0,0,0)
14 #env=zeros(t)
15 sustain = 0.5
16 release = 0.2
17 car = 440.0
18 modfreq = 3.0
19 modind = 10.0
20 # def adsr_envelope(sample_rate, attack_time, decay_time, sustain_level, sustain_time, release_time):
21 #     attack_samples = int(attack_time * sample_rate)
22 #     decay_samples = np.linspace(0, 1, attack_samples)
23 #     sustain_samples = np.linspace(1, sustain_level, decay_samples)
24 #     release_samples = int(release_time * sample_rate)
25 #     attack = ???????
26 #     decay = ??????
27 #     sustain = np.ones(sustain_samples) * sustain_level
28 #     release = np.linspace(sustain_level, 0, release_samples)
29 #     envelope = combine
30 #     return decay
```

Visual Studio Code interface showing the file `q5.py` (a4 - Visual Studio Code). The code defines `adsrenv` and `fmssynthesis`:

```
q5.py > fmssynthesis
31
32 def adsrenv(duration):
33     totalss = int(duration * rate1)
34     attackss = int(attack * rate1)
35     decayss = int(decay * rate1)
36     releasess = int(release * rate1)
37
38     env = np.zeros(totalss)
39     env[:attackss] = np.linspace(0, 1, attackss)
40     env[attackss:attackss+decayss] = np.linspace(1, sustain, decayss)
41     env[-releasess:] = np.linspace(sustain, 0, releasess)
42     #print(env)
43     #print(attackss)
44     #print(decayss)
45     #print(releasess)
46     #print(sustain)
47     return env
48
49
50 def fmssynthesis(note, duration):
51     carfreq = car * 2 ** ((note - 69) / 12)
52     moddd = carfreq * modfreq
53     modamp = carfreq * modind / moddd
54
55     t = np.linspace(0, duration, int(duration * rate1), endpoint=False)
56     modulator_waveform = np.sin(2 * np.pi * moddd * t) * modamp
57     carrier_waveform = np.sin(2 * np.pi * carfreq * t + modulator_waveform)
58
59     return carrier_waveform
60
```

The screenshot shows a Visual Studio Code interface with a dark theme. The Explorer sidebar on the left lists files: 'a4' (containing 'a4.py', 'q5.py', and several MP3 files), 'q5.py' (selected), 'OUTLINE', and 'TIMELINE'. The main editor area displays 'q5.py' with the following code:

```
61 parser = argparse.ArgumentParser(description='Real-time synth with single sinusoidal oscillator and ADSR env')
62 parser.add_argument('note', type=int, help='MIDI note number (0-127)')
63 args = parser.parse_args()
64
65 # start_time = time.time()
66
67 # while (time.time() - start_time) < 4:
68 #     print(time.time() - start_time)
69
70 note = args.note
71 duration = duration
72 waveform = fmsynthesis(note, duration)
73 env = adsrenv(duration)
74 waveform *= env
75
76 # t = np.linspace(0, duration, int(sample_rate * duration), False)
77 # freq = 440
78 # waveform = np.sin(2 * np.pi * freq * t)
79 audio = pyaudio.PyAudio()
80 stream = audio.open(format=pyaudio.paFloat32, channels=1, rate=rate1, frames_per_buffer=buffer1, output=True)
81
82 for i in range(0, len(waveform), buffer1):
83     stream.write(waveform[i:i+buffer1].astype(np.float32).tobytes())
84
85
86 stream.stop_stream()
87 stream.close()
88 audio.terminate()
89
```

The status bar at the bottom shows: In 50, Col 16, Spaces: 4, UTF-8, CRLF, Python 3.11.1 64-bit, 5:09 PM, 3/29/2023.

For question 6 I had to extend the synthesizer from the previous question to support an ADSR envelope and two-operator FM synthesis. In order to do this we first have to calculate the number of samples for each stage of the envelope using the predetermined time duration and the sample rate. Next, we generate arrays for each stage of the envelope using the `numpy.linspace()` function. In the case of the attack and release stages, we generate arrays that go from 0 to 1 and from 1 to 0. In the case of the decay stage, we generate an array that goes from 1 to the sustain level. Finally, in the case of the sustain stage, we generate an array of constant values equal to the sustain level. We then concatenate the four arrays into a single envelope array using the `numpy.concatenate()` function. Finally, we return the envelope array. When applied to an audio waveform, the ADSR envelope generated by this function will shape the amplitude of the waveform over time, producing a characteristic "pluck" or "punch" sound. For fm synthesis we first have to calculate the number of samples for the waveform using the provided duration and sample rate. We then create arrays for the carrier and modulator waveforms using the `numpy.linspace()` function. In this case, we're generating a wave for both the carrier and modulator waveforms. Next, we calculate the modulation index, which determines the amount of frequency modulation applied to the carrier waveform. We then apply the frequency modulation to the carrier waveform by multiplying it with the modulation index. Finally, we return the modulated carrier waveform.

```
1 import pyaudio
2 import numpy as np
3 #import time
4 import argparse
5
6
7 rate1 = 44100
8 buffer1 = 1024
9 duration = 4.0
10 attack = 0.1
11 decay = 0.2
12 waveform=0
13 #env=zeros(t)
14 #env=ones(t)
15 sustain = 0.5
16 release = 0.2
17 car = 440.0
18 modfreq = 3.0
19 modind = 10.0
20
21 # def adsr_envelope(sample_rate, attack_time, decay_time, sustainlevel, sustain_time, release_time):
22 #     attack_samples = int(attack_time * sample_rate)
23 #     decay_samples = np.linspace(0, 1, attack_samples)
24 #     sustain_samples = np.linspace(1, sustainlevel, decay_samples)
25 #     release_samples = int(release_time * sample_rate)
26 #
27 # def callback():
28 #     #print("hello")
29 t = np.linspace(0, duration, int(duration * rate1), False)
```

```
29 t = np.linspace(0, duration, int(duration * rate1), False) > wf
30 op1ratioo = 1.0
31 op1uned = 0.0
32 op1level = 99
33 op2ratioo = 2.003
34 op2uned = 0.0
35 op2level = 0
36 op3ratioo = 4.0
37 op3uned = -7.0
38 op3level = 99
39 op4ratioo = 0.0
40 op4uned = 0.0
41 op4level = 0
42
43
44 #     output1 = op1note + op2note + op3note + op4note
45 #     #print(output1)
46 #     output1 = np.asarray(output1, dtype=np.float32).tobytes()
47 #     #print(output1)
48 #
49 #     # Scale the waveform to the range [-1, 1]# op1note = op1level * np.sin((2 ** t) + op1uned - np.sin(2 *t))
50 #     op2note = op2level * np.sin((2 ** t) + op2uned - np.sin(2 *t))
51 #     op3note = op3level * np.sin((2 ** t) + op3uned - np.sin(2 *t))
52 #     op4note = op4level * np.sin((2 ** t) + op4uned - np.sin(2 *t))
53 #     #print("hello")
54 op1note = op1level * np.sin(2 * np.pi * op1ratioo * t + op1uned * np.sin(2 * np.pi * op1ratioo * t))
55 op2note = op2level * np.sin(2 * np.pi * op2ratioo * t + op2uned * np.sin(2 * np.pi * op2ratioo * t))
56 p3note = op3level * np.sin(2 * np.pi * op3ratioo * t + op3uned * np.sin(2 * np.pi * op3ratioo * t))
57 op4note = op4level * np.sin(2 * np.pi * op4ratioo * t + op4uned * np.sin(2 * np.pi * op4ratioo * t))
58 outputnote = (op1note + op2note + op3note + op4note)
59
```

```

def adsrenv(duration):
    totallss = int(duration * rate1)
    attackss = int(attack * rate1)
    decayss = int(decay * rate1)
    releasess = int(release * rate1)

    env = np.zeros(totallss)
    env[attackss:] = np.linspace(0, 1, attackss)
    env[attackss:attackss+decayss] = np.linspace(1, sustain, decayss)
    env[-releasess:] = np.linspace(sustain, 0, releasess)

    #print(env)
    #print(attackss)
    #print(decayss)
    #print(releasess)
    #print(sustain)
    return env

def fmsynthesis(note, duration):
    carfreq = car * 2 ** ((note - 69) / 12)
    modddd = carfreq * modfreq
    modamp = carfreq * modind / modddd

    t = np.linspace(0, duration, int(duration * rate1), endpoint=False)
    modulator_waveform = np.sin(2 * np.pi * modddd * t) * modamp
    carrier_waveform = np.sin(2 * np.pi * carfreq * t + modulator_waveform)

    return carrier_waveform

```

```

parser = argparse.ArgumentParser(description='Real-time synthesis')
parser.add_argument('note', type=int, help='MIDI note number (0-127)')
args = parser.parse_args()

# start_time = time.time()

# while (time.time() - start_time) < 4:
#     print(time.time() - start_time)

note = args.note
duration = duration
waveform = fmsynthesis(note, duration)
env = adsrenv(duration) * outputnote
waveform *= env

# t = np.linspace(0, duration, int(sample_rate * duration), False)
# freq = 440
# waveform = np.sin(2 * np.pi * freq * t)
audio = pyaudio.PyAudio()
stream = audio.open(format=pyaudio.paFloat32, channels=1, rate=rate1, frames_per_buffer=buffer1, output=True)
# print("open")
# stream = audio.open(format=8,
#                      channels=1,
#                      rate=rate1,
#                      output=True,
#                      stream_callback=callback1)

for i in range(0, len(waveform), buffer1):
    #print("write")
    stream.write((waveform[i:i+buffer1].astype(np.float32)).tobytes())

```

For question 7 we have to implement one of the algorithms and try to approximate one patch sound. I had to extend the implementation to 4 operators and also implement the frequency ratios and detune parameters but ignore the LFO and Pitch envelopes. I started by defining the parameters for each operator, making sure to include the frequency ratio, detune, and amplitude modulation parameters. The waves that are generated are modulated and the amplitude modulation is multiplied with the wave using a modulating waveform. Next, I combined the waves for each operator by summing the waves together. The amplitude of each operator is

multiplied by a scaling factor to avoid clipping and ensure that the output is within the range of -1.0 to 1.0. I then normalized the output to make sure the range is correct. After normalizing the output, the rest of the code runs making the calculated audio that has the adsr envelope and the patch run for the designated number of seconds. The patch that is being replicated in this program is a classic DX21 bass sound. While searching online this patch is characterized by a rich, warm tone with a fast attack and moderate decay. The sound is created using four operators, with the first two operators set to a 1:1 frequency ratio, and the third and fourth operators set to a 3:2 frequency ratio. The detune parameter is set to create a slight detuning effect, which adds depth to the sound. The sound is then shaped using a combination of envelope generators and amplitude modulation. In the original DX21 patch, the envelope generators are used to shape the pitch and amplitude of the sound. However, in this implementation, only the amplitude envelope generator is used to shape the amplitude of the sound.

File Edit Selection View Go Run Terminal Help

q5.py - a4 - Visual Studio Code

EXPLORER A4

- > screenshots
- a4.py 2
- c4 patch 1 adsr.mp3
- c4 patch 1.mp3
- c4 patch 2.mp3
- g4 patch 1.mp3
- g4 patch 2.mp3
- q5.py
- test.py
- test.wav

q5.py > ...

```
36 op3ratioo = 4.0
37 op3uned = -7.0
38 op3level = 99
39 op4ratioo = 0.0
40 op4uned = 0.0
41 op4level = 0
42 #
43
44 lfofreq = 5.0
45 lfodepth = 0.1
46
47
48 pitchattack= 0.1
49 pitchdecay = 0.2
50 pitchsustain = 0.7
51 pitchrelease = 0.3
52 pitchamount = 0.5
53 #     output1 = op1note + op2note + op3note + op4note
54 #     print(output1)
55 #     output1 = np.asarray(output1, dtype=np.float32).tobytes()
56 #     print(output1)
57 #
58 # Scale the waveform to the range [-1, 1]# op1note = op1level * np.sin((2 ** t) + op1uned - np.sin(2 *t))
59 # op2note = op2level * np.sin((2 ** t) + op2uned - np.sin(2 *t))
60 # op3note = op3level * np.sin((2 ** t) + op3uned - np.sin(2 *t))
61 # op4note = op4level * np.sin((2 ** t) + op4uned - np.sin(2 *t))
62 #print("hello")
63 op1note = op1level * np.sin(2 * np.pi * op1ratioo * t + op1uned * np.sin(2 * np.pi * op1ratioo * t))
64 op2note = op2level * np.sin(2 * np.pi * op2ratioo * t + op2uned * np.sin(2 * np.pi * op2ratioo * t))
65 p3note = op3level * np.sin(2 * np.pi * op3ratioo * t + op3uned * np.sin(2 * np.pi * op3ratioo * t))
```

In 128, Col 23 Spaces: 4 UTF-8 CRLF Python 3.11.1 64-bit 2:53 PM 3/31/2023

File Edit Selection View Go Run Terminal Help

q5.py - a4 - Visual Studio Code

EXPLORER A4

- > screenshots
- a4.py 2
- c4 patch 1 adsr.mp3
- c4 patch 1.mp3
- c4 patch 2.mp3
- g4 patch 1.mp3
- g4 patch 2.mp3
- q5.py
- test.py
- test.wav

q5.py > ...

```
72 #     envelope = combine
73 #     return decay
74 def q8envelopes(envelope):
75     lfo waveform = np.sin(2 * np.pi * np.arange(len(envelope)) * lfofreq / rate1)
76     lfo waveform = (lfo waveform + 1) * lfodepth
77     #print(lfo waveform)
78     newenvelope = envelope * lfo waveform
79     pitch = np.linspace(1.0, pitchsustain, len(envelope))
80     #print(lfo waveform)
81     pitch[:int(pitchdecay * rate1)] = np.linspace(1.0, pitchsustain, int(pitchdecay * rate1))
82     wholething = newenvelope * pitch
83     #print(lfo waveform)
84     return wholething
85 def adsrenv(duration):
86     totalls = int(duration * rate1)
87     attackss = int(attack * rate1)
88     decayss = int(decay * rate1)
89     releasess = int(release * rate1)
90
91     env = np.zeros(totalls)
92     env[:attackss] = np.linspace(0, 1, attackss)
93     env[attackss:attackss+decayss] = np.linspace(1, sustain, decayss)
94     env[decayss:] = np.linspace(sustain, 0, releasess)
95     #print(env)
96     #print(attackss)
97     #print(decayss)
98     #print(releasess)
99     #print(sustain)
100    return env
```

In 128, Col 23 Spaces: 4 UTF-8 CRLF Python 3.11.1 64-bit 2:53 PM 3/31/2023

Visual Studio Code interface showing code for question 8 part 1. The code implements a synthesis function that generates a waveform based on note and duration parameters. It uses np.linspace to create a time array, np.sin for oscillation, and np.pi for pi. It also uses argparse to parse command-line arguments for note and duration.

```
def fmssynthesis(note, duration):
    carfreq = car * 2 ** ((note - 69) / 12)
    moddd = carfreq * modfreq
    modamp = carfreq * modind / moddd

    t = np.linspace(0, duration, int(duration * rate1), endpoint=False)
    modulator_waveform = np.sin(2 * np.pi * moddd * t) * modamp
    carrier_waveform = np.sin(2 * np.pi * carfreq * t + modulator_waveform)

    return carrier_waveform
```

Visual Studio Code interface showing code for question 8 part 2. This part adds audio output using PyAudio. It opens a stream, writes the waveform to it, and then stops and closes the stream. The code includes imports for np, np.float32, pyaudio, and np.pi.

```
waveform *= env
# t = np.linspace(0, duration, int(sample_rate * duration), False)
# freq = 440
# waveform = np.sin(2 * np.pi * freq * t)
audio = pyaudio.PyAudio()
stream = audio.open(format=pyaudio.paFloat32, channels=1, rate=rate1, frames_per_buffer=buffer1, output=True)
# print("open")
# stream = audio.open(format=8,
#                      channels=1,
#                      rate=rate1,
#                      output=True,
#                      stream_callback=(variable) buffer1: Literal[1024])
for i in range(0, len(waveform), buffer1):
    #print("write")
    stream.write((waveform[i:i+buffer1].astype(np.float32)).tobytes())
    # while stream.is_active():
    #     pass
stream.stop_stream()
stream.close()
audio.terminate()
```

For question 8 I had to add the LFO and pitch envelopes to the implementation. The synthesizer is still mono-phonic, but can still approximate the existing patch. I added LFO and pitch envelopes to an existing envelope using a single function. The input parameters are the envelope that I am adding information to. The rest of the needed information is declared-initialized elsewhere. I first calculated the LFO waveform as a wave with the specified frequency and depth. I then multiplied this waveform with the input envelope to apply the LFO modulation. Next, I generate a pitch envelope as a linear ramp from 1.0 to the specified sustain

level. The first decay time seconds of the envelope are used for the ramp. The pitch envelope is then multiplied with the LFO-modulated envelope to apply the pitch modulation. Finally, the modified envelope with both LFO and pitch envelopes applied is returned. This function is useful for creating software synthesizers, sound effects, or other audio processing tasks, and can make the resulting sound more expressive and dynamic. It is used to create a similar sound to a specific patch from the Yamaha DX21 synthesizer. The replicated patch is talked about in the previous question report but LFO and Pitch Envelopes are powerful tools that can be used to add movement, variation, and complexity to a DX21 synthesizer patch. By experimenting with different settings and combinations, you can create a wide range of sounds that are more unique and expressive than what was possible with the code without these two elements

File Edit Selection View Go Run Terminal Help

q5.py - a4 - Visual Studio Code

EXPLORER A4

- > screenshots
- a4.py 2
- c4 patch 1 adsr.mp3
- c4 patch 1.mp3
- c4 patch 2.mp3
- g4 patch 1.mp3
- g4 patch 2.mp3
- q5.py
- test.py
- test.wav

q5.py > ...

```
114
115
116     parser = argparse.ArgumentParser(description='Real-time synth with single sinusoidal oscillator and ADSR env')
117     parser.add_argument('note', type=int, help='MIDI note number (0-127)')
118     args = parser.parse_args()
119
120     # start_time = time.time()
121
122     # while (time.time() - start_time) < 4:
123     #     print(time.time() - start_time)
124
125     note = args.note
126     note2=note+5
127     note3=note2+5
128     note4=note3+5
129
130     def compute(note):
131         waveform = fm synthesis(note, duration)
132         env = adsrenv(duration) * outputnote
133         env = q8envelopes(env)
134         waveform *= env
135         return waveform
136     waveform=compute(note)
137     waveform2=compute(note2)
138     waveform3=compute(note3)
139     waveform4=compute(note4)
140
141     # t = np.linspace(0, duration, int(sample_rate * duration), False)
142     # freq = 440
143     # waveform = np.sin(2 * np.pi * freq * t)
144     audio = pyaudio.PyAudio()
```

Ln 180, Col 20 Spaces: 4 UTF-8 CRLF Python 3.11.1 64-bit 1:27 PM 4/1/2023

File Edit Selection View Go Run Terminal Help

q5.py - a4 - Visual Studio Code

EXPLORER A4

- > screenshots
- a4.py 2
- c4 patch 1 adsr.mp3
- c4 patch 1.mp3
- c4 patch 2.mp3
- g4 patch 1.mp3
- g4 patch 2.mp3
- q5.py
- test.py
- test.wav

q5.py > ...

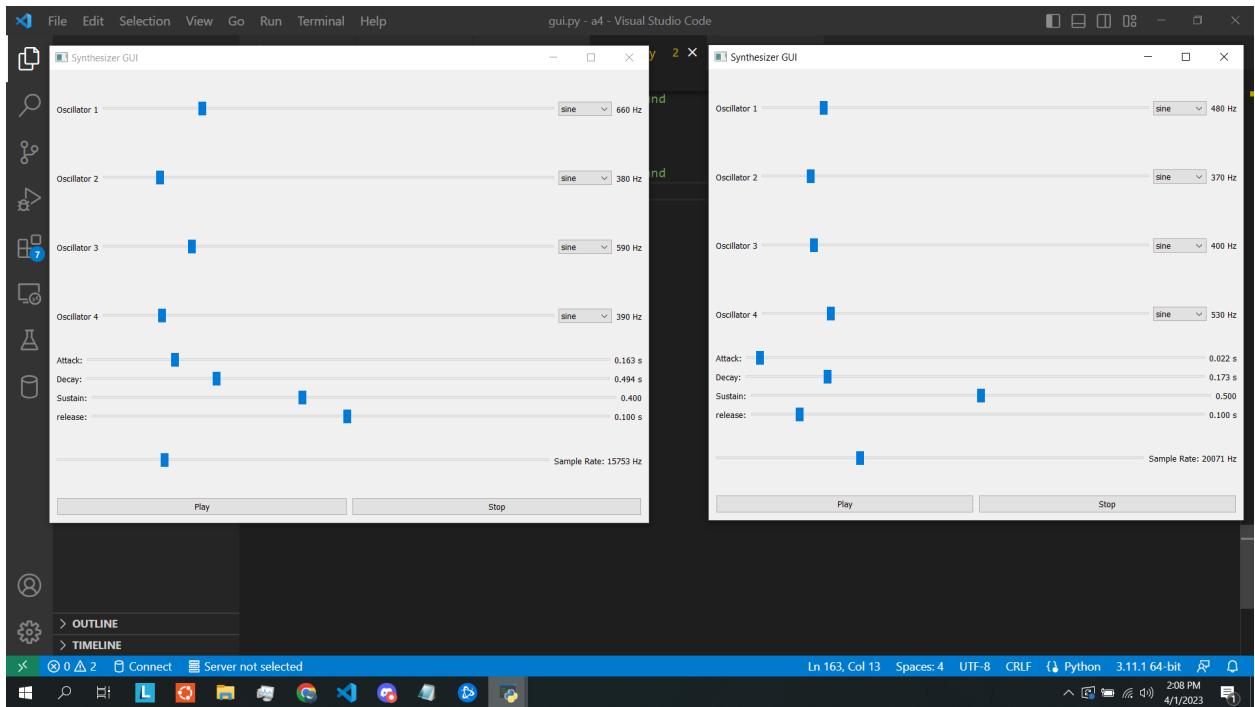
```
134     return waveform
135     waveform=compute(note)
136     waveform2=compute(note2)
137     waveform3=compute(note3)
138     waveform4=compute(note4)
139
140     # t = np.linspace(0, duration, int(sample_rate * duration), False)
141     # freq = 440
142     # waveform = np.sin(2 * np.pi * freq * t)
143     audio = pyaudio.PyAudio()
144     stream = [None] * 4
145
146     def playsound(channelnum, waveform):
147         stream[channelnum] = audio.open(format=pyaudio.paFloat32, channels=1, rate=rate1, frames_per_buffer=buffer1, o
148         # print("open")
149         # stream = audio.open(format=8,
150         #                     channels=1,
151         for i in range(0, len(waveform), buffer1):
152             #print("write")
153             stream[channelnum].write((waveform[i:i+buffer1].astype(np.float32)).tobytes())
154             #stream[channelnum].write((waveform[i:i+buffer1].astype(np.float32)).tobytes()+(b'\x00\x01\x02\x03\x04\x05'
155             # while stream.is_active():
156             #     pass
157             stream[channelnum].close()
158
159     def polyphony():
160         # create a thread for each voice
161
162         threads = []
163         thread = threading.Thread(target=playsound, args=(0, waveform))
164         threads.append(thread)
```

Ln 180, Col 20 Spaces: 4 UTF-8 CRLF Python 3.11.1 64-bit 1:28 PM 4/1/2023

The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Editor:** The main editor window displays the file `q5.py`. The code implements polyphony using threads. It defines a function `polyphony()` which creates four separate threads, each running the `playsound()` function on different voices (0, 1, 2, 3). The code uses the `threading` module to manage these threads.
- Explorer:** On the left, the Explorer sidebar shows a folder named "A4" containing files: `a4.py`, `c4 patch 1 adsr.mp3`, `c4 patch 1.mp3`, `c4 patch 2.mp3`, `c4 patch 1.mp3`, `c4 patch 2.mp3`, `q5.py`, and `test.py`.
- Status Bar:** ShowsLn 180, Col 20, Spaces: 4, UTF-8, CRLF, Python 3.11.1 64-bit, 1:28 PM, 4/1/2023.

For question 9 we had to extend the synthesizer to support polyphony using voice allocation. In order to avoid conflicting sounds when using polyphony, each sound gets assigned a separate voice or channel. The process of doing that for this question is called voice allocation. I decided to implement polyphony and voice allocation through the use of threads. Threads are separate sequences of code that can run concurrently with other threads. By assigning each sound to a separate thread, we can ensure that multiple sounds can be played simultaneously without interfering with each other. First, I needed to create a thread for each sound that we wanted to play using the inputted note numbers. Each thread runs the function `playsound()` that plays the sound on a specific voice or channel. Next, we need to ensure that each thread is synchronized so that they do not interfere with each other. Finally, we need to ensure that the threads are properly cleaned up when they finish playing their sound. We use the `join()` method to wait for each thread to finish before exiting the program. I decided the number of voices to use by investigating how many voices I could have simultaneously without audio artifacts on my computer. Then once I hear a weird echoing noise then I know I've gone too far and have too many voices such that they can not play in unison/simultaneously.



```

File Edit Selection View Go Run Terminal Help gui.py - a4 - Visual Studio Code
gui.py > SynthesizerGUI > stop_sound
1 from PyQt5.QtCore import Qt, QThread
2 from PyQt5.QtWidgets import QApplication, QMainWindow, QWidget, QVBoxLayout, QHBoxLayout, QLabel, QSlider, QPushButton
3 #import threading
4 #import tkinter as tk
5
6 class SynthesizerGUI(QMainWindow):
7     def __init__(self):
8         super().__init__()
9         #window = tk.Tk()
10        # Create main widget and layout
11        self.main_widget = QWidget(self)
12        self.main_layout = QVBoxLayout(self.main_widget)
13        self.main_widget.setLayout(self.main_layout)
14        #greeting = tk.Label(text="Hello, Tkinter")
15        # Create oscillator widgets and layouts
16        self.osc_labels = []
17        #greeting.pack()
18        self.freq_sliders = []
19        self.waveform_combos = []
20        self.freq_labels = []
21        #window.mainloop()
22        for i in range(4):
23            # Create oscillator label
24            label = QLabel(f"Oscillator {i+1}")
25            self.osc_labels.append(label)
26            #label = tk.Label(text="Hello, Tkinter")
27            # Create frequency slider
28            freq_slider = QSlider(Qt.Horizontal)
29            freq_slider.setMinimum(20)
30            freq_slider.setMaximum(3000)
31            freq_slider.setValue(1000)
32            self.freq_sliders.append(freq_slider)
33            # Create waveform dropdown
34            waveform_combo = QComboBox()
35            waveform_combo.addItem("sine")
36            waveform_combo.addItem("square")
37            waveform_combo.addItem("triangle")
38            waveform_combo.addItem("pulse")
39            self.waveform_combos.append(waveform_combo)
40            # Create ADSR envelope controls
41            attack_slider = QSlider(Qt.Horizontal)
42            attack_slider.setMinimum(0.01)
43            attack_slider.setMaximum(1.0)
44            attack_slider.setValue(0.1)
45            decay_slider = QSlider(Qt.Horizontal)
46            decay_slider.setMinimum(0.01)
47            decay_slider.setMaximum(1.0)
48            decay_slider.setValue(0.5)
49            sustain_slider = QSlider(Qt.Horizontal)
50            sustain_slider.setMinimum(0.01)
51            sustain_slider.setMaximum(1.0)
52            sustain_slider.setValue(0.5)
53            release_slider = QSlider(Qt.Horizontal)
54            release_slider.setMinimum(0.01)
55            release_slider.setMaximum(1.0)
56            release_slider.setValue(0.1)
57            # Create play and stop buttons
58            play_button = QPushButton("Play")
59            stop_button = QPushButton("Stop")
60            # Add controls to layout
61            layout = QHBoxLayout()
62            layout.addWidget(label)
63            layout.addWidget(freq_slider)
64            layout.addWidget(waveform_combo)
65            layout.addWidget(attack_slider)
66            layout.addWidget(decay_slider)
67            layout.addWidget(sustain_slider)
68            layout.addWidget(release_slider)
69            layout.addWidget(play_button)
70            layout.addWidget(stop_button)
71            self.main_layout.addLayout(layout)
72
73 if __name__ == "__main__":
74     app = QApplication(sys.argv)
75     window = SynthesizerGUI()
76     window.show()
77     sys.exit(app.exec())

```

The screenshot shows the Visual Studio Code interface with the code for the `gui.py` file. The code uses PyQt5 to create a main window (`QMainWindow`) containing four oscillators. Each oscillator is represented by a label, a frequency slider, a waveform dropdown, and an ADSR envelope section with attack, decay, sustain, and release sliders. The code also includes play and stop buttons. The code is written in Python and follows a standard object-oriented structure with a constructor (`__init__`) and a main loop (`if __name__ == "__main__":`).

gui.py - a4 - Visual Studio Code

```
41 waveform_combo = QComboBox()
42 waveform_combo.addItem("sine")
43 waveform_combo.addItem("square")
44 waveform_combo.addItem("sawtooth")
45 waveform_combo.addItem("triangle")
46 self.waveform_combos.append(waveform_combo)
47 # Create frequency label
48 freq_label = QLabel("440 Hz")
49 self.freq_labels.append(freq_label)
50 #label = tk.Label(text="Hello, Tkinter", fg="white", bg="black")
51 # Create oscillator layout
52 oscillator_layout = QHBoxLayout()
53 oscillator_layout.addWidget(label)
54 oscillator_layout.addWidget(freq_slider)
55 oscillator_layout.addWidget(waveform_combo)
56 oscillator_layout.addWidget(freq_label)
57 self.main_layout.addLayout(oscillator_layout)
58
59 # Create ADSR envelope widgets and layout
60 self.attack_slider = QSlider(Qt.Horizontal)
61 self.attack_slider.setMinimum(0)
62 self.attack_slider.setMaximum(1000)
63 self.attack_slider.setValue(10)
64
65 # Create decay slider and layout
66 self.decay_slider = QSlider(Qt.Horizontal)
67 self.decay_slider.setMinimum(0)
68 self.decay_slider.setMaximum(1000)
69 self.decay_slider.setValue(100)
70 self.decay_slider.setSingleStep(1)
71 self.decay_slider.valueChanged.connect(lambda value: self.set_adsr_label("decay", value))
72 self.decay_label = QLabel("0.100 s")
73 decay_layout = QHBoxLayout()
74 decay_layout.addWidget(QLabel("Decay:"))
75 decay_layout.addWidget(self.decay_slider)
76 decay_layout.addWidget(self.decay_label)
77 attack_layout.addWidget(self.attack_label)
78 button = tk.Button(
79     text="Click me!",
80     width=25,
81     height=5,
82     bg="blue",
83     fg="yellow",
84 )
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
```

gui.py - a4 - Visual Studio Code

```
77 attack_layout.addWidget(self.attack_label)
78 button = tk.Button(
79     text="Click me!",
80     width=25,
81     height=5,
82     bg="blue",
83     fg="yellow",
84 )
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
```

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows a folder named "A4" containing files: screenshots, a4.py, c4 patch 1 adsr.mp3, c4 patch 1.mp3, c4 patch 2.mp3, g4 patch 1.mp3, g4 patch 2.mp3, gui.py (selected), q5.py, test.py, and test.wav.
- Code Editor:** Displays the contents of the "gui.py" file. The code uses Tkinter to create a window titled "Hello Python" with a size of 300x200+10+20. It includes a play button and a stop button. The main loop starts with `window=Tk()`. A function `set_freq_label` is defined to update a frequency label based on a value and index.
- Status Bar:** Shows the following information: Line 157, Column 25, Spaces: 4, UTF-8, CRLF, Python 3.11.1 64-bit, 2:31 PM, 4/1/2023.

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows a folder named "A4" containing files: screenshots, a4.py, c4 patch 1 adsr.mp3, c4 patch 1.mp3, c4 patch 2.mp3, g4 patch 1.mp3, g4 patch 2.mp3, gui.py (selected), q5.py, test.py, and test.wav.
- Code Editor:** Displays the contents of the "gui.py" file. The code uses PyQt5 to create a window titled "Synthesizer GUI". It includes a checkbox labeled "Tennis" and a variable `v2`. The `if __name__ == '__main__':` block creates an application, a main window, and shows it.
- Status Bar:** Shows the following information: Line 157, Column 25, Spaces: 4, UTF-8, CRLF, Python 3.11.1 64-bit, 2:31 PM, 4/1/2023.

For question 10 I had to add a graphical-user interface for the synthesizer. For this I used PyQt5. The code creates a `SynthesizerGUI` class that extends PyQt5's `QMainWindow`. In the constructor of the class, we set up the basic layout of the GUI with four oscillators, an ADSR envelope, and a sample rate slider. The oscillators are created using `QComboBox` widgets that allow the user to select between different waveforms. The GUI also has a play button that uses the `sounddevice` library. The sample rate slider in the GUI allows the user to adjust the sample rate. The code updates the sample rate value whenever the slider is moved, and regenerates

the waveform with the new sample rate value. This updating whenever the slider is moved is true for all the sliders. This SynthesizerGUI class provides a simple way for users to generate and play custom synthesized sounds with different waveform shapes, ADSR envelopes, and sample rates. I also made it such that you can have two instances of the gui open at the same time for polyphony, you can add more than two but just for showing that I can, I chose to show two. In terms of steps, I first had to Import necessary libraries and modules, including PyQt5 and sounddevice, then make/define a SynthesizerGUI class and in this class set up the basic layout of the GUI. This contains widgets for waveform shapes, ADSR envelope values, and the sample rate. Next was connecting the sound playing method to a QPushButton widget in the GUI so that the user can play or stop the generated sound whenever they would like. Lastly, creating a QApplication object and two SynthesizerGUI objects, and run the application using the app.exec_() method. This way I can get two gui windows.