Odd Semester (2022)



**BINUS UNIVERSITY**

**BINUS INTERNATIONAL**

**Assignment Cover Letter**

**(Individual Work)**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | |  | |  | |  |
| **Student Information**: **Surname** | | | | | **Given Names**    **Brenda** | | **Student ID Number**  **2201841702** | |  |
| 1. | | **Spears** |  | |  |
|  |  |  |
| **Course Code** | **: COMP6502** |  |  | | **Course Name** | | **: Introduction to Programming** | |  |
| **Class** | **: L1CC** |  |  | | **Name of Lecturer(s)** | | **:Jude Joseph Lamug Martinez** | |  |
|  |  |  |  | |  | |  | |  |
| **Major** | **: CS** |  |  | |  | |  | |  |
| **Title of Assignment**  (if any) | : Audio Spectrum Analyzer | |  |  | |  | |  | |
| **Type of Assignment**    **Submission Pattern** | **: Final Project** |  |  | |  | |  | |  |
| **Due Date** | **: 23-11-18** |  |  | | **Submission Date** | | **: 23-11-18** | |  |

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer’s instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

# Plagiarism/Cheating

BiNus International seriously regards all forms of plagiarism, cheating and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

# Declaration of Originality

By signing this assignment, I understand, accept and consent to BiNus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student: (Name of Student)

1. Brenda Spears

**“The Audio Spectrum Analyzer”**

**Name : Brenda Spears**

**ID : 2201841702**

1. **Description**

**The function of this program:**

The purpose of this program is to stream real time audio data from a microphone into python using PyAudio. It converts the data to binary then using Python’s Struct module to convert binary data to integer data. Then create the Numpy array using the Numpy module. Compute the FFT using the Scipy module. Using Matplotlib, the program creates and output a waveform graph and a spectrum graph.

**II.a. Design/Plan**

Start

**Project’s Hierarchy Chart**

AudioStream Class

init\_plots(self) function

start\_plot(self) function

Output “Stream Start”

Not self.pause

Yes

Input data

Converts data to data\_int

Create data\_np array

Draw next frame

frame\_count += 1

No

Output “Average Frame Rate”

exit\_app(self) function

onClick(self) function

End

Yes

**II.b. Explanation of Each Function Inside the Class**

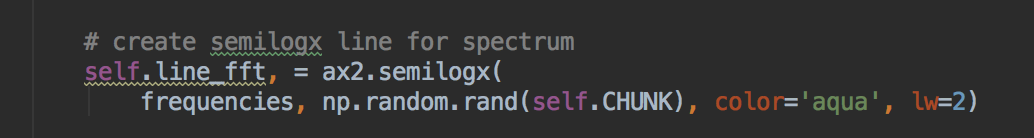
*FinalCode.py*

* **init\_plots (self) :**
* Creates the variables for plotting. “samples” for the audio waveform graph and “frequencies” for the spectrum graph.
* Creates the Matplotlib figure and axes.
* Connects with the onClick function. It ends the loop when user clicked on any part of the Matplotlib page.
* Creates/Plot object with random data for the audio waveform graph.
* Creates the semilogx line for the spectrum. Semilogx module compares the measured intensity of the sound with the given level on a logarithmic scale . The x-axis is scaled to its logarithmic form.
* Formats the audio waveform and audio spectrum axes, the titles, lables, and limits.
* **stats\_plot (self) :**
* Prints the “Stream Started” text.
* Takes real time audio as its input and converts it to binary data.
* Converts the collected data into integer as Matplotlib plots integer values.
* Creates a Numpy array and offset it by 128 so that the audio waveform graph will have its positive and negative amplitude. We use slicing to remove the unnecessary data that is returning to 0, making the plotted line connects.
* Computes FFT using Scipy.fftpack. It converts the data\_int into a set of complex numbers to plot the audio spectrum (algorithm). The “np.abs” module is used to convert the imaginary values and returns the magnitudes. When plotted, it shows two frequencies, the negative frequency and positive frequency of the sine wave, therefore we use slicing to take the first half of it (algorithm).
* Instead of erasing and replacing every frames, it updates the frames, up to 40 frames per second making it a smoother wave.
* Increments the frame\_count which is later used to calculate the average frame rate when the cycle/loop ended.
* When user ends the loop, it calculates the average frame rate and prints it after being formatted to 0 decimal place (whole number).
* **exit\_app (self) :**
* Prints “Stream Closed”
* Ends/Break the loop, stops streaming the real time audio input.
* **onClick (self, event) :**
* Changes the self.pause to True so that the while loop in start\_plot (self) function will end.
* This occur when user clicked anywhere on the page.

**Class Diagram**

|  |
| --- |
| **AudioStream** |
| **- CHUNK**  **- FORMAT**  **- CHANNELS**  **- RATE**  **- pause**  **- p**  **- stream**  **- init\_plots( )**  **- start\_plot( )** |
| **- init\_plots( )**  **- start\_plot( )**  **- exit\_app( )**  **- onClick( )** |
|  |

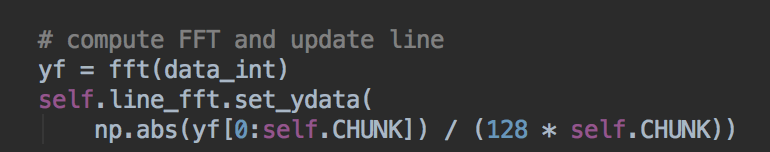
**III.a. Lessons that Have Been Learned**

1. ***The use of “ semilogx ”:***

I did some research, and I found out that we can convert real values to its logarithmic form using this module knows as “semilogx” .

1. ***Colors in Matplotlib:***

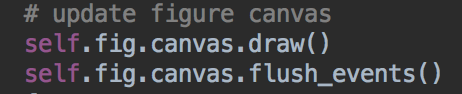
When I plotted the graph, I find it really dull and boring when it is only using the primary colours. I did some researches, and apparently, matplotlib provides numerous colours we can use for the labels, face colors, inside or outside the graph, the lines plotted and many more.

1. ***The existence of Scipy:Screen Shot 2018-11-23 at 01.01.32.png***

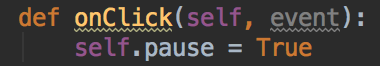
Scipy is a module that conducts advanced mathematical calculations like calculus. Using Scipy, I managed to calculate the audio spectrum using the formula in Scipy.

1. ***The use of PyAudio:Screen Shot 2018-11-23 at 01.14.16.pngScreen Shot 2018-11-23 at 01.14.29.png***

Doing this project, I was able to use PyAudio to stream the real-time audio data I need to create the graphs. PyAudio allows user to connect audio with coding, using the audio in different ways.

1. ***Update the graph:***

When doing this project, it requires the program to to take different samples of audio, plotting the data to different frames. Instead of erasing and replacing every frames, I found out that we can use those lines to update the frames up to 40 frames per second, making it a smooth moving graph.

1. ***Closing/Ending the program :Screen Shot 2018-11-23 at 01.24.51.png***

I found a fancy way in terminating the program where user could end it when clicking on any part on the Matplotlib page.

**III.b. Problem that Have Been Overcome**

Creating this program is not as easy as I thought it would be. Even though the code did not reach 200 lines, the codes used to put this project together is extremely complicated and tricky. I found some codes through the internet in order to build it, but different operating systems uses different commands, which is hard as it requires trial and errors to fix them. When installing PyAudio, I had a really hard time as that module only works with python version 3.6 and below, having to know that my python is at version 3.7. I tried many different ways to install it, by downgrading my python, to illegally installing it. Other huge problem I overcome was converting the backend I was using. There are also some codes that became redundant. Understanding the formula to make the waveform and spectrum graph was also very difficult with the fact that when using the formula, it returns some data that we don’t need.

https://bastibe.de/2013-05-30-speeding-up-matplotlib.html

**Resources :**

- <https://bastibe.de/2013-05-30-speeding-up-matplotlib.html> (speeding up the Matplotlib graph)

- <https://github.com/markjay4k/Audio-Spectrum-Analyzer-in-Python/blob/master/audio_spectrum.py> (background code with the algorithms)

- <https://stackoverflow.com> (website I used when I was trying to fix the errors)

**V. Source Code**

*FinalCode.py*

**import** matplotlib

matplotlib.use(**'TkAgg'**)

**import** matplotlib.pyplot **as** plt

**import** numpy **as** np

**import** pyaudio

**import** struct

**from** scipy.fftpack **import** fft

**import** time

**class** AudioStream(object):

**def** \_\_init\_\_(self):

*# stream constants*

self.CHUNK = 1024 \* 2 *# Samples per Frame*

self.FORMAT = pyaudio.paInt16 *# Audio Format*

self.CHANNELS = 1 *# Channel for Microphone*

self.RATE = 44100 *# Samples per Second*

self.pause = **False**

*# stream object to get data from microphone*

self.p = pyaudio.PyAudio() *# PyAudio class instance*

self.stream = self.p.open(

format=self.FORMAT,

channels=self.CHANNELS,

rate=self.RATE,

input=**True**,

output=**True**,

frames\_per\_buffer=self.CHUNK )

self.init\_plots()

self.start\_plot()

**def** init\_plots(self):

*# variables for plotting*

samples = np.arange(0, 2 \* self.CHUNK, 2) *# Samples (Waveform)*

frequencies = np.linspace(0, self.RATE, self.CHUNK) *# Frequencies (spectrum)*

*# create matplotlib figure and axes*

self.fig, (ax1, ax2) = plt.subplots(2, figsize=(15, 7))

self.fig.canvas.mpl\_connect(**'button\_press\_event'**, self.onClick)

*# create a line object with random data*

self.line, = ax1.plot(samples, np.random.rand(self.CHUNK), color = **'aqua'**, lw=2)

*# create semilogx line for spectrum*

self.line\_fft, = ax2.semilogx(

frequencies, np.random.rand(self.CHUNK), color=**'aqua'**, lw=2)

*# format waveform axes*

ax1.set\_title(**'Audio Waveform (Top) & Audio Spectrum (Bottom)'**)

ax1.set\_xlabel(**'Samples'**)

ax1.set\_ylabel(**'Volume'**)

ax1.set\_ylim(0, 255)

ax1.set\_xlim(0, 2 \* self.CHUNK)

ax1.set\_facecolor(**'k'**)

ax2.set\_facecolor(**'k'**)

plt.setp(

ax1, yticks=[0, 128, 255],

xticks=[0, self.CHUNK, 2 \* self.CHUNK],

)

plt.setp(ax2, yticks=[0, 1],)

*# format spectrum axes*

ax2.set\_xlim(20, self.RATE / 2)

self.fig.patch.set\_facecolor(**'rosybrown'**)

*# show axes*

plt.show(block=**False**)

**def** start\_plot(self):

print(**'Stream Started'**)

frame\_count = 0

start\_time = time.time()

**while not** self.pause:

*# Binary Data*

data = self.stream.read(self.CHUNK,exception\_on\_overflow = **False**)

*# Convert data to integers*

data\_int = struct.unpack(str(2 \* self.CHUNK) + **'B'**, data)

*# Create Numpy Array, offset by 128*

data\_np = np.array(data\_int, dtype=**'b'**)[::2] + 128

self.line.set\_ydata(data\_np) *# Sets data to the audio waveform graph.*

*# compute FFT and update line*

yf = fft(data\_int)

self.line\_fft.set\_ydata(

np.abs(yf[0:self.CHUNK]) / (128 \* self.CHUNK))

*# update figure canvas*

self.fig.canvas.draw()

self.fig.canvas.flush\_events()

frame\_count += 1

**else**:

self.fr = frame\_count / (time.time() - start\_time)

print(**'average frame rate = {:.0f} FPS'**.format(self.fr))

self.exit\_app()

**def** exit\_app(self):

print(**'Stream Closed'**)

self.p.close(self.stream)

**def** onClick(self, event):

self.pause = **True**

*# Calling the Class (main program)*

AudioStream()