Program Design Methods and Intro to Programming Python Final Project

Python Guitar Tuner

Student Information:

Student Name: Raphael Reynldi

Student ID: 2440071973

Class Information:

Class: L1BC

Lecturer's Name: Ida Bagus Kerthyayana

Introduction

The classes for Introduction to Programming and Program Design Methods taught at Binus International were designed to help develop coding skills for the students who attended by teaching them the concepts of the programming language of Python. During the period of the first semester, students both experienced and new to programming had the chance to understand the fundamentals of programming and learn the ins and outs of Python. At the end of this program, students were expected to have understood the concepts and ideas taught to them so that they could apply the knowledge in a more realistic situation. Thus, students were tasked to create a final project they wanted using the knowledge that they've gotten and the ideas that they have produced.

After the task and the project specifications were given to us, I really wanted to produce something that was related to a passion of mine so I decided to choose a topic that was related to music. Music as a whole is a broad choice for an idea so after some thought, I decided to settle on something simple yet very important to me as a musician. I decided to create a guitar tuner that would be tuned to E standard. This idea had me very interested because tuners are tools that are always used everytime a musician is going to perform. I've always had an interest in how to manipulate data from an audio signal so I thought that this was a good way to start getting into it.

Project Specification

The function of this program:

The function of this program is to cater towards guitarists in particular to help them tune their guitar to E standard tuning. The project aims to help the problem that has always affected guitarists which is a defective tuner. A clip on tuner while very handy and helpful at first, will start losing its accuracy the moment it has fallen to the ground once and twice. As a musician, the last thing they prioritize the most when preparing for a performance, is the condition of the tool they're using. Especially if it's a small tool like a clip on tuner. Since all of the processing of my Python tuner is digital, no matter how many times it's been accidentally thrown or has fallen to the ground, the accuracy will still remain the same the moment it was created. Thus, the consistency that the tuner is able to achieve is something it has over clip on tuners. The target audience of this project is for guitarists that play their instruments at home. The aim of this project is to help guitarists tune their guitars to E standard consistently by taking their audio signal and processing it through the code.

Solution Design

Design/Plan:

Right off the bat when I chose to tackle this idea, I needed a way to be able to take an input from an audio signal and find a way for it to be processed data. After some time looking around and seeing what I could use, I found a method called Discrete Fourier transform or DFT for short which allows me to take samples within an audio input and convert it into data that can be processed into cosine functions that oscillate at different frequencies. Fortunately, I didn't need to define it when doing the project because there is already a function in Numpy, which is a module in Python which automatically converts the audio signal into a DFT. Furthermore, I found an algorithm which is able to process the matrix of a DFT and compute said matrix into a frequency domain. This is huge because most of the theory surrounding musical notes is based off of frequency which creates a note from a given pitch. The definition of a frequency is the reciprocal of a duration based on its period of a repeating event. The formula of finding a frequency based on its period is 1 divided by the period in seconds. The definition of a pitch is how said frequency is perceived as a sound. A note is a pitch that is defined with a letter. Using the Fast Frontier Transformation algorithm which factorizes DFT matrices into sparse zero factors which allows the audio to process the signal at a much faster rate. What I plan to do with all of this audio processing is to find the maximum frequency of the given buffer, and then determine if the frequency is near to the note selected by the user or not. Obviously it is very hard to get tunings to the exact frequency and even the human ear can't detect these small little things so I decided to not include an output which states that the guitar is tuned. Rather, I will showcase an output which shows if the user is below above the frequency of a given note and how far in hertz they are to the given note. I will have a simple GUI for the users to navigate to the note they want to tune to. By doing this, the user can check each individual notes that they are tuning to.

The modules I will be using to achieve my project will be Pyaudio, Numpy, Tkinter, Threading, and os. The Pyaudio module will be used to receive the input coming from the user. The Numpy module is used to process a lot if not all of the data received from the input coming from the Pyaudio. The Tkinter module will be used to display a GUI for the user to navigate to the note they want to tune to. The Threading module will be used to run two while loop codes which I will showcase later on as Tkinter will crash when there is more than one loop happening at once. The answer to this problem for me will be the Threading module which will allow me to run two codes containing loops simultaneously. The os module will be used to clear out the terminal because the data is constantly being pushed to the terminal.

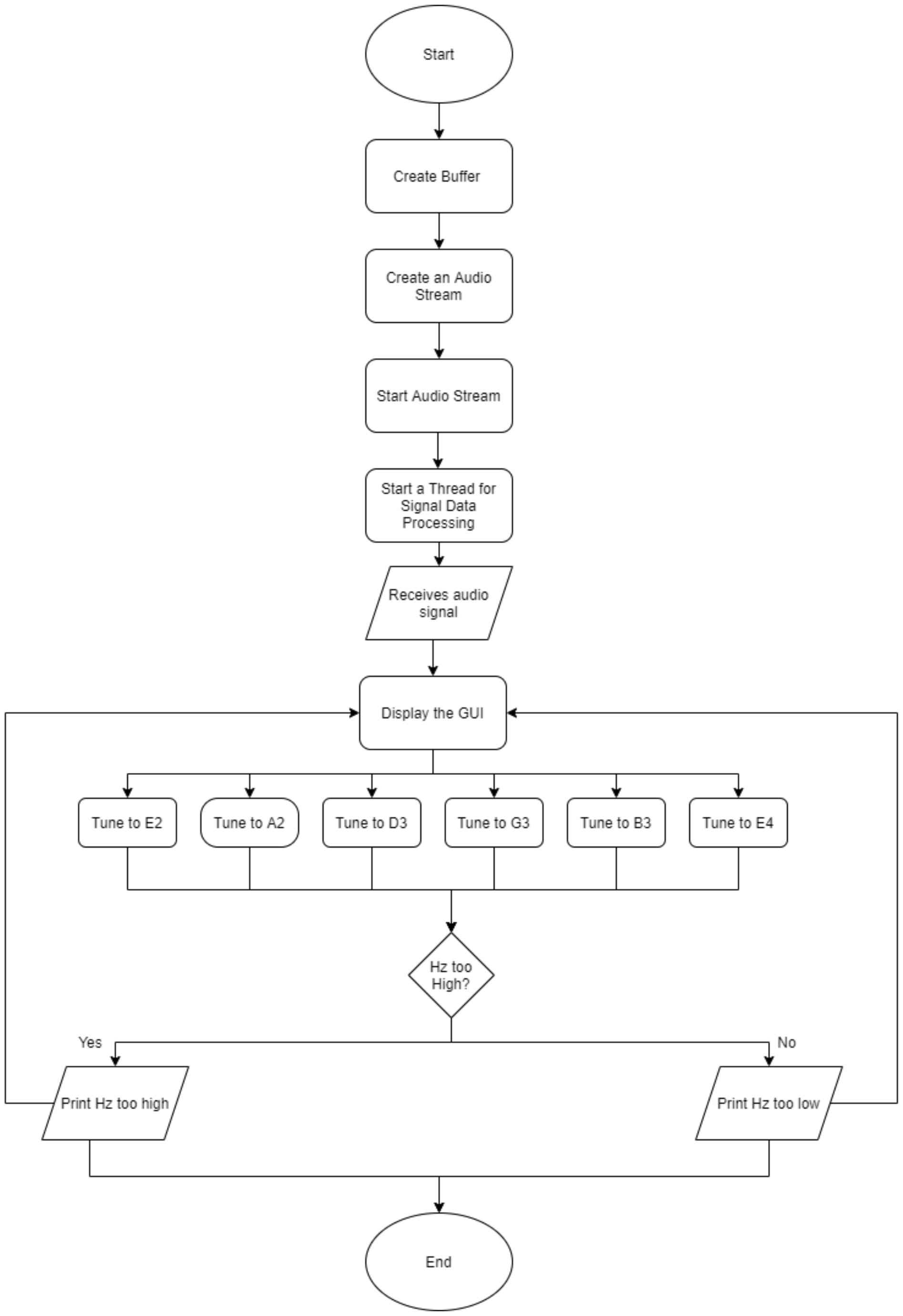


Image 1. Flowchart of the main logic behind the project

A discussion of what was implemented and how it works

The project is made up of only one file titled main.py. However, inside the main.py includes all of the modules used, all of the functions, the GUI class, and the audio processing function.

```
      from tkinter import *
      import pyaudio
      import numpy as np
      import os
      import threading
```

These are the modules that I used for the project. The Tkinter module is used to create the GUI. The Pyaudio module is used to receive an audio signal from the user's microphone. The Numpy module is used to process the signal and convert the signal into DFT and use the FFT algorithm to process it. The os module is used to clear out the terminal as more data is being printed out. The threading module is used to run two loops simultaneously to overcome an issue with running both Tkinter and a while loop together.

```
#Frequency of each note in Hz
E2_freq = 247.00
A2_freq = 220.00
D3_freq = 290.00
G3_freq = 194.00
B3_freq = 733.00
E4_freq = 329.00
```

These are the frequency of each of the notes on the guitar string. These values will be used later on to determine if the user is in tune or not.

```
Sample_Rate = 44100      # Sampling frequency in Hz
Chunk = 21050             # The number of frames in the buffer
Samples_Per_Buffer = 44100 # Used determine how much data that can be processed
```

The Sample_Rate value will be used when receiving the audio signal and is used to determine how many samples are being recorded. The higher the number, the higher quality the sample rate is. The Chunk value is the amount of frames that can be received in the buffer. The Samples_Per_Buffer value is used to determine how much data the zero array can hold. The higher the amount, the more data the array can hold.

```
# Creating a zero array to be used as a buffer when receiving input
buffer = np.zeros(Samples_Per_Buffer, dtype=np.int32)

# Defining the audio stream and starting it
stream = pyaudio.PyAudio().open(format=pyaudio.paInt32,
                                channels=1,
                                rate=Sample_Rate,
                                input=True,
                                frames_per_buffer=Chunk)

stream.start_stream()
```

The buffer value is used to store a zero matrix based on the value of the Samples_Per_Buffer value. The data type of the zero matrix array is an int32 which essentially means that it will return integer values. The stream value is used to call in the pyaudio module of PyAudio().open which allows the microphone of the user to be used. The Format of the data is in integers. The audio signal will have one channel which means that the audio will be in mono. The rate is the value of Sample_Rate which will give us a sampling frequency of the value stored in Hz. The input is the function that allows the user to give an audio signal into the program. The frames_per_buffer function allows the user to store a set amount of data from a continuous audio signal. The stream.start_stream() function triggers the audio stream and allows input of the microphone.

Start_Stream() Function

```
# A function for the input to be processed to find the max frequency of a buffer
def start_stream():
  while stream.is_active:
    buffer[-Chunk:] = np.frombuffer(stream.read(Chunk), np.int32) # Appending new
buffers and removing old buffers

    FFT = np.fft.fft(buffer) # Using the FFT algorithm to process the DFT

    Freq = (np.abs(FFT).argmax()) # Finding the maximum frequency

    print("The guitar frequency is {:f} Hz".format(Freq))
```

The start_stream() function is used for the threading function. But basically this function contains how the audio is being processed. The old data gets put at the end while the new data gets amended by a new clean buffer. Then the buffer is converted into data that contains frequency through the FFT algorithm which allows us to read said data. After that, using the argmax function which returns a maximum value in conjunction with the abs function from Numpy which allows us to get rid of the redundant values allows us to find the highest frequency in the buffer. I used someone else’s code when trying to receive the new buffers and I have credited him in my references.

```
# Threading Process to run the GUI and the audio processing simultaneously
Thread = threading.Thread(target=start_stream, daemon = True)
Thread.start()
```

The Threading process allows the start_stream() function to run simultaneously along side the Tkinter GUI.

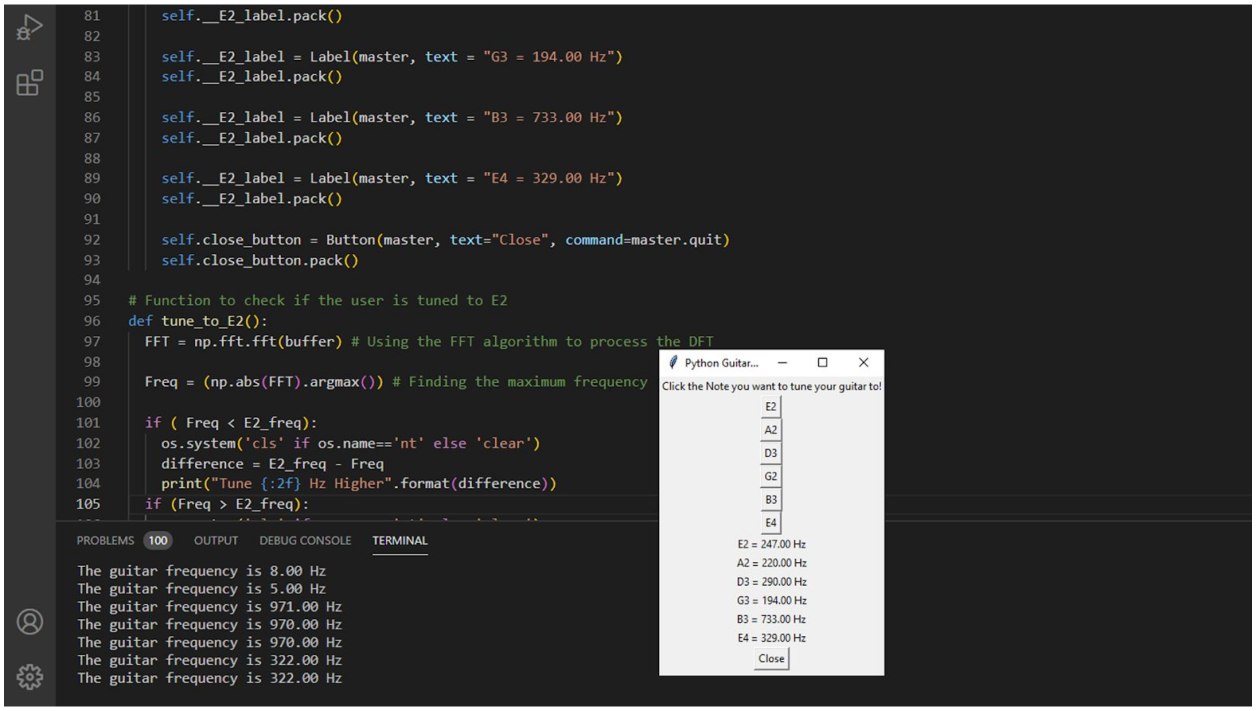
```python
# The GUI for each of the Notes
class TunerGUI:
  def __init__(self, master,):
    self.__master = master
    self.__master.title("Python Guitar Tuner")
    self.__master.geometry = ("600 x 600")

    self.__label = Label(master, text="Click the Note you want to tune your
guitar to!")
    self.__label.pack()

    self.__e2_button = Button(master, text = "E2", command=tune_to_E2)
    self.__e2_button.pack()

    self.__a2_button = Button(master, text = "A2", command=tune_to_A2)
    self.__a2_button.pack()

    self.__d3_button = Button(master, text = "D3", command=tune_to_D3)
    self.__d3_button.pack()

    self.__G3_button = Button(master, text = "G2", command=tune_to_G3)
    self.__G3_button.pack()

    self.__B3_button = Button(master, text = "B3", command=tune_to_B3)
    self.__B3_button.pack()

    self.__E4_button = Button(master, text = "E4", command=tune_to_E4)
    self.__E4_button.pack()

    self.__E2_label = Label(master, text = "E2 = 247.00 Hz")
    self.__E2_label.pack()

    self.__E2_label = Label(master, text = "A2 = 220.00 Hz")
    self.__E2_label.pack()

    self.__E2_label = Label(master, text = "D3 = 290.00 Hz")
    self.__E2_label.pack()

    self.__E2_label = Label(master, text = "G3 = 194.00 Hz")
    self.__E2_label.pack()

    self.__E2_label = Label(master, text = "B3 = 733.00 Hz")
    self.__E2_label.pack()

    self.__E2_label = Label(master, text = "E4 = 329.00 Hz")
    self.__E2_label.pack()

    self.close_button = Button(master, text="Close", command=master.quit)
    self.close_button.pack()
```

The Class TunerGUI is used to showcase the gui of the program and can be used to navigate which note the user wants to tune into. It also has the frequencies of each note shown in the app so that the user can tune to what note they desire. It has 6 buttons that correspond to each of the notes and when pressed takes the buffer of what the user has and goes through and checks if the frequency is tuned to the desired note.

```python
# Function to check if the user is tuned to E2
def tune_to_E2():
  buffer[-Chunk:] = np.frombuffer(stream.read(Chunk), np.int32) # Appending new buffers and removing old buffers

  FFT = np.fft.fft(buffer) # Using the FFT algorithm to process the DFT

  Freq = (np.abs(FFT).argmax()) # Finding the maximum frequency

  if ( Freq < E2_freq):
    os.system('cls' if os.name=='nt' else 'clear')
    difference = E2_freq - Freq
    print("Tune {:2f} Hz Higher".format(difference))
  if (Freq > E2_freq):
    os.system('cls' if os.name=='nt' else 'clear')
    difference = Freq - E2_freq
    print("Tune {:2f} Hz Lower".format(difference))

# Function to check if the user is tuned to A2
def tune_to_A2():
  buffer[-Chunk:] = np.frombuffer(stream.read(Chunk), np.int32) # Appending new buffers and removing old buffers

  FFT = np.fft.fft(buffer) # Using the FFT algorithm to process the DFT

  Freq = (np.abs(FFT).argmax()) # Finding the maximum frequency

  if ( Freq < A2_freq):
    os.system('cls' if os.name=='nt' else 'clear')
    difference = A2_freq - Freq
    print("Tune {:2f} Hz Higher".format(difference))
  if (Freq > A2_freq):
    os.system('cls' if os.name=='nt' else 'clear')
    difference = Freq - A2_freq
    print("Tune {:2f} Hz Lower".format(difference))

# Function to check if the user is tuned to D3
def tune_to_D3():
  buffer[-Chunk:] = np.frombuffer(stream.read(Chunk), np.int32) # Appending new
buffers and removing old buffers

  FFT = np.fft.fft(buffer) # Using the FFT algorithm to process the DFT

  Freq = (np.abs(FFT).argmax()) # Finding the maximum frequency

  if ( Freq < D3_freq):
    os.system('cls' if os.name=='nt' else 'clear')
    difference = D3_freq - Freq
    print("Tune {:2f} Hz Higher".format(difference))
  if (Freq > D3_freq):
    os.system('cls' if os.name=='nt' else 'clear')
    difference = Freq - D3_freq
    print("Tune {:2f} Hz Lower".format(difference))

# Function to check if the user is tuned to G3
def tune_to_G3():
  buffer[-Chunk:] = np.frombuffer(stream.read(Chunk), np.int32) # Appending new
buffers and removing old buffers

  FFT = np.fft.fft(buffer) # Using the FFT algorithm to process the DFT

  Freq = (np.abs(FFT).argmax()) # Finding the maximum frequency

  if ( Freq < G3_freq):
    os.system('cls' if os.name=='nt' else 'clear')
    difference = G3_freq - Freq
    print("Tune {:2f} Hz Higher".format(difference))
  if (Freq > G3_freq):
    os.system('cls' if os.name=='nt' else 'clear')
    difference = Freq - G3_freq
    print("Tune {:2f} Hz Lower".format(difference))

# Function to check if the user is tuned to B3
def tune_to_B3():
  buffer[-Chunk:] = np.frombuffer(stream.read(Chunk), np.int32) # Appending new
buffers and removing old buffers
```

```
  FFT = np.fft.fft(buffer) # Using the FFT algorithm to process the DFT

  Freq = (np.abs(FFT).argmax()) # Finding the maximum frequency

  if ( Freq < B3_freq):
    os.system('cls' if os.name=='nt' else 'clear')
    difference = B3_freq - Freq
    print("Tune {:2f} Hz Higher".format(difference))
  if (Freq > B3_freq):
    os.system('cls' if os.name=='nt' else 'clear')
    difference = Freq - B3_freq
    print("Tune {:2f} Hz Lower".format(difference))

# Function to check if the user is tuned to E4
def tune_to_E4():
  buffer[-Chunk:] = np.frombuffer(stream.read(Chunk), np.int32) # Appending new
buffers and removing old buffers

  FFT = np.fft.fft(buffer) # Using the FFT algorithm to process the DFT

  Freq = (np.abs(FFT).argmax()) # Finding the maximum frequency

  if ( Freq < E4_freq):
    os.system('cls' if os.name=='nt' else 'clear')
    difference = E4_freq - Freq
    print("Tune {:2f} Hz Higher".format(difference))
  if (Freq > E4_freq):
    os.system('cls' if os.name=='nt' else 'clear')
    difference = Freq - E4_freq
    print("Tune {:2f} Hz Lower".format(difference))

root = Tk()
my_gui = TunerGUI(root)
root.mainloop()
```

The remaining functions are used to determine if the guitar is tuned to the specified frequency. It goes through the latest chunk of the buffer when pressed and goes through a bunch of if statements. The if statements checks if the maximum frequency of the note is lower and will take the difference of the frequency of the selected note and minus it with the frequency of the buffer. If it doesn’t meet the criteria, it will cycle to the next if statement and check if the frequency is higher than the selected note. If the selected note is higher, then it will subtract the current frequency with the selected note and print the difference between the two and tell the user to print lower.



```
        self.__E2_label.pack()

        self.__E2_label = Label(master, text = "G3 = 194.00 Hz")
        self.__E2_label.pack()

        self.__E2_label = Label(master, text = "B3 = 733.00 Hz")
        self.__E2_label.pack()

        self.__E2_label = Label(master, text = "E4 = 329.00 Hz")
        self.__E2_label.pack()

        self.close_button = Button(master, text="Close", command=master.quit)
        self.close_button.pack()

# Function to check if the user is tuned to E2
def tune_to_E2():
    FFT = np.fft.fft(buffer) # Using the FFT algorithm to process the DFT

    Freq = (np.abs(FFT).argmax()) # Finding the maximum frequency

    if ( Freq < E2_freq):
        os.system('cls' if os.name=='nt' else 'clear')
        difference = E2_freq - Freq
        print("Tune {:2f} Hz Higher".format(difference))
    if (Freq > E2_freq):
```

```
The guitar frequency is 8.00 Hz
The guitar frequency is 5.00 Hz
The guitar frequency is 971.00 Hz
The guitar frequency is 970.00 Hz
The guitar frequency is 970.00 Hz
The guitar frequency is 322.00 Hz
The guitar frequency is 322.00 Hz
```

Click the Note you want to tune your guitar to!

E2 = 247.00 Hz
A2 = 220.00 Hz
D3 = 290.00 Hz
G3 = 194.00 Hz
B3 = 733.00 Hz
E4 = 329.00 Hz

A Screenshot of me trying to tune my guitar to E2. Using the terminal to monitor my guitar’s frequency. The terminal is able to monitor if my frequency went up or down.

References

1. https://people.csail.mit.edu/hubert/pyaudio/docs/
2. https://www.mathworks.com/help/matlab/ref/fft.html
3. https://newt.phys.unsw.edu.au/jw/notes.html
4. https://docs.python.org/3/library/tkinter.html
5. https://github.com/mzucker/python-tuner/blob/master/tuner.py