**Program cs03** CS 5245: Data Programming

Fall 2022: Updated September 23, 2022 R. M. Parry

---

Read chapter 5 and 10 in Fundamentals of Python Programming and complete the following exercises.

**Complete computer setup by following the directions here:**
`http://cs.appstate.edu/~rmp/cs5245/setup.pdf`

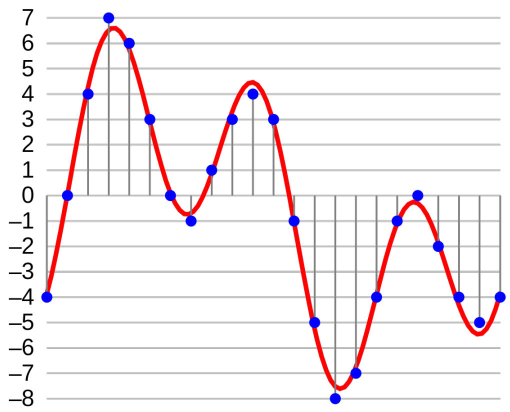
Create a folder called `cs03` to store your files for this lab.

Complete the following activities using the **exact variable names** and save the files using the **exact file names** as specified.

To test your programs, run them from IDLE and check their output.

## 1 audio

Working with audio. Digial audio stores an acoustic waveform as a sequence of numbers. For compact disc quality audio, each number is stored as a 16-bit integer between -32768 and 32767 and these numbers are recorded 44,100 times per second. Sound causes a microphone to vibrate, sending a voltage signal to your computer's sound card which then converts them into the digital format. Similarly, the sound card converts digital audio into a voltage signal that causes your speakers to vibrate creating sound. A Python `list` is a perfectly reasonable way to store digital audio!



To read and write sound files on your computer, I am providing two helper functions in audio_helpers.py. Download the `audio_helpers.py` file and store it in the same folder as your other assignment files. The following code segment, reads an audio file from the hard drive and stores it into a different filename:

```python
from audio_helpers import load, save

rate, data = load('old_name.wav')
save('new_name.wav', rate, data)
```

The `rate` is an integer indicating the number of audio samples (numbers) per second. The audio data in `data` is a `list`.

For example:

```
from audio_helpers import load, save

# read the audio file as a list
rate, data = load('file_name.wav')
# do some processing or visualization...
data[0] = 0
# save the file as an edited version
save('edited.wav', rate, data)
```

In addition, we will want to use pyplot to plot the audio signals. For example:

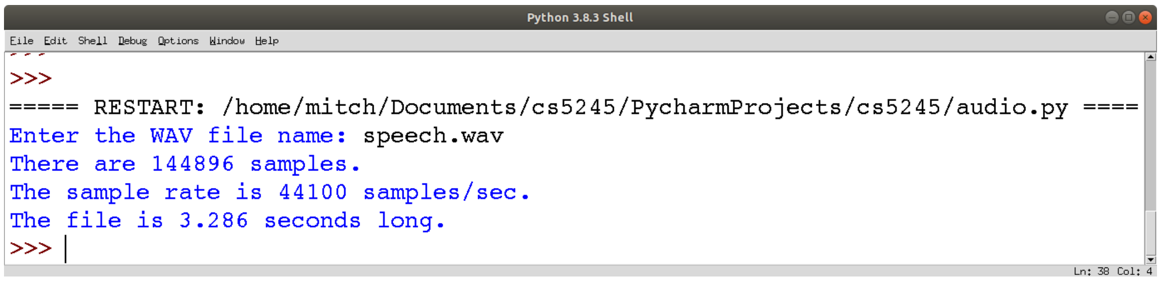
```
import matplotlib.pyplot as plt

x = [...]
y = [...]

plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('y')
plt.title('title')
plt.show()
```

Now that you can read and write WAV files and convert the data to and from lists, we can start manipulating the data. First, download the following file and store it in your `cs03` folder:

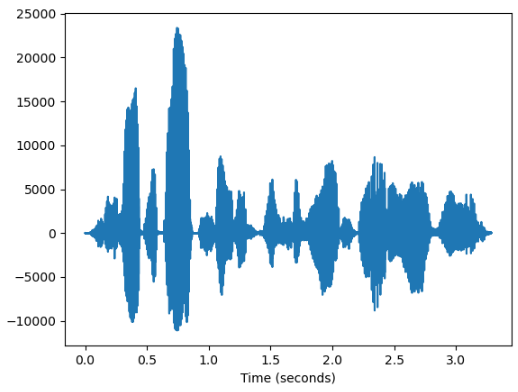
http://cs.appstate.edu/~rmp/cs5245/speech.wav

In a file called `audio.py`,

1. Prompt the user for the name of a WAV file.
2. Print the number of samples, sample rate, and the recording's length in seconds (using 3 decimal places).
3. Plot waveform in a figure where the x-axis represents time (in seconds) and the y-axis represents the sound pressure level in the audio `data`. Notice that you will need to create a list of times so that correspond to the timing of each audio sample. Basically, starting at zero and taking $1/n$ size steps in time ($n$ is the sample rate).
4. Web-CAT will check that each sample's $(x, y)$ coordinate and the print out.
5. Running the program should look like this:



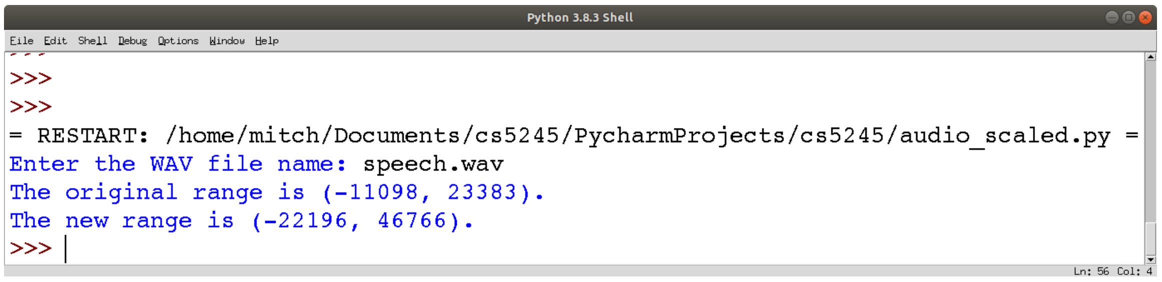
```
>>>
===== RESTART: /home/mitch/Documents/cs5245/PycharmProjects/cs5245/audio.py ====
Enter the WAV file name: speech.wav
There are 144896 samples.
The sample rate is 44100 samples/sec.
The file is 3.286 seconds long.
>>> 
```



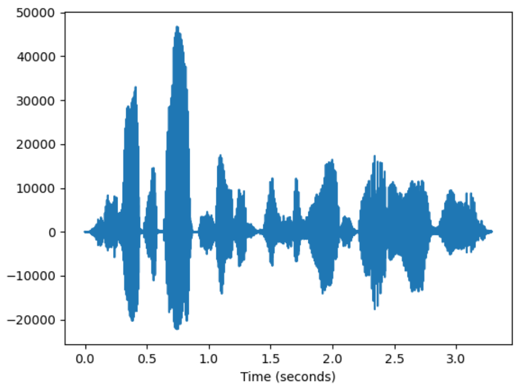
## 2 audio_scaled

In a file called `audio_scaled.py`:

1. Prompt the user for the name of a WAV file.
2. Print the minimum and maximum amplitude of the audio `data`.
3. Create a new list that contains each sample at twice its original amplitude.
4. Print the minimum and maximum amplitude of the new audio `data`.
5. Plot the new waveform in a figure like before.
6. It should look like this:



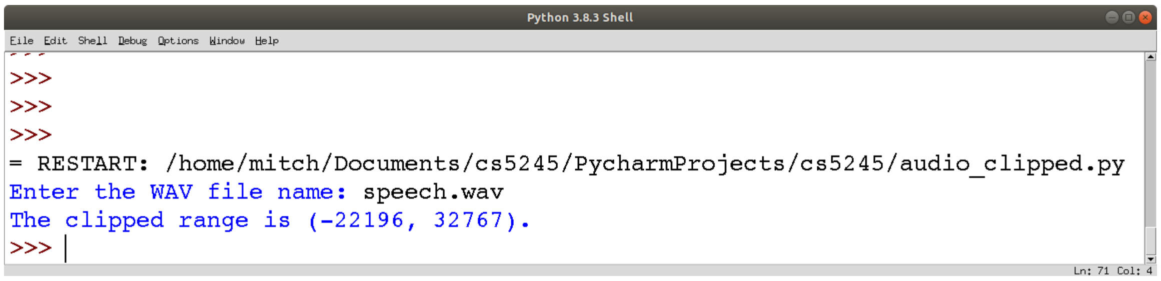
```
>>>
>>>
= RESTART: /home/mitch/Documents/cs5245/PycharmProjects/cs5245/audio_scaled.py =
Enter the WAV file name: speech.wav
The original range is (-11098, 23383).
The new range is (-22196, 46766).
>>>
```



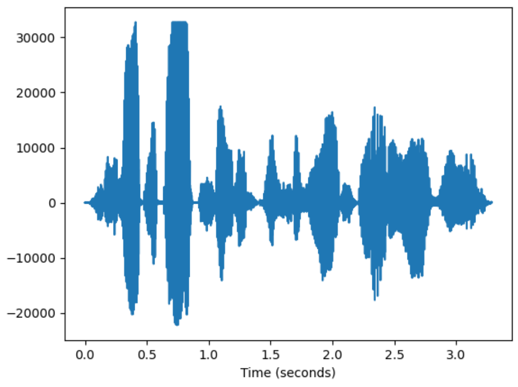
## 3 `audio_clipped`

You might have noticed that the maximum amplitude after scaling was greater than the allowable range for 16-bit integers [-32768, 32767]. In order to successfully save this as a new file, we'll need to clip the values that are greater than 32767 or less than -32768. In a file called `audio_clipped.py`:

1. Prompt the user for the name of a WAV file.
2. Create a new list that contains each sample at twice its original amplitude.
3. Clip the new list so that amplitudes greater than 32767 get replaced by 32767 and amplitudes less than -32768 get replaced by -32768.
4. Print the minimum and maximum amplitude of the clipped audio `data`.
5. Plot the new waveform in a figure like before.
6. Save the scaled and clipped audio data to a new file with `_clipped` appended to the file name. That is, if the original file name is `speech.wav`, the clipped file name would be `speech_clipped.wav`. You can use string slicing (much like list slicing) to get a substring and concatenate the the part you want.
7. Use your headphones to listen to the difference between the original WAV file and the clipped one. (Try double-clicking on the file in your file browser.)
8. It should look like this:

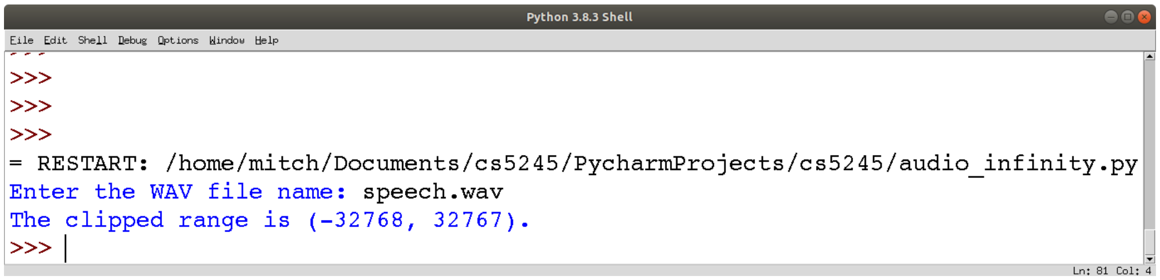


```
>>>
>>>
>>>
= RESTART: /home/mitch/Documents/cs5245/PycharmProjects/cs5245/audio_clipped.py
Enter the WAV file name: speech.wav
The clipped range is (-22196, 32767).
>>> 
```

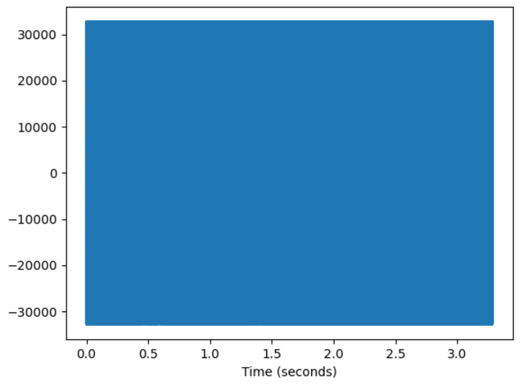


## 4 `audio_infinity`

To infinity! Let's try amplifying the sound way up. In a file called `audio_infinity.py`, prompt for the file name from the user, scale it by a really big number, say 100000 (sort of like infinity). After scaling and clipping there should only be three possible data values: -32768, 0, or 32767. Print the clipped range, and plot the new figure. Save the new waveform with `_infinity.wav` appended to the original file name (as before). What do you think it will sound like? Use your headphones to find out. It should look like this:



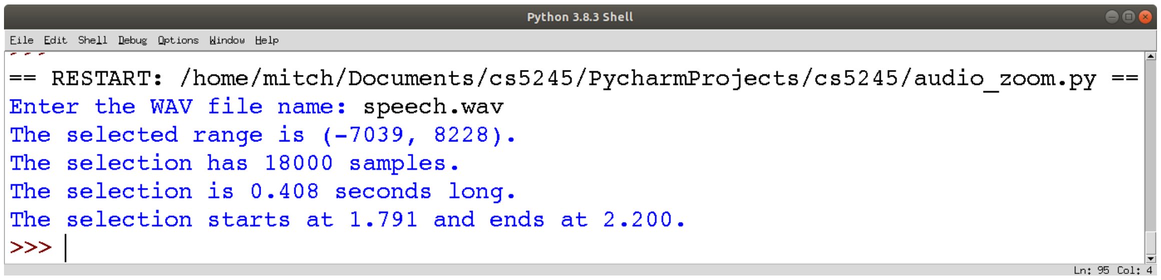
```
>>>
>>>
>>>
= RESTART: /home/mitch/Documents/cs5245/PycharmProjects/cs5245/audio_infinity.py
Enter the WAV file name: speech.wav
The clipped range is (-32768, 32767).
>>> 
```



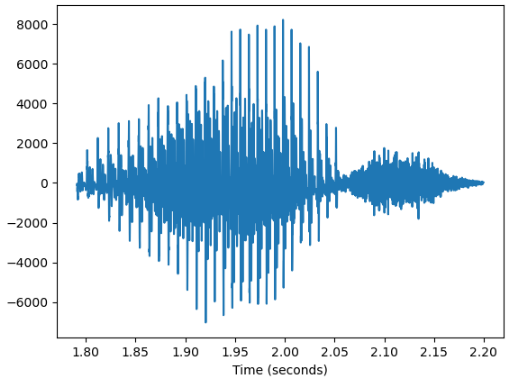
## 5 `audio_zoom`

Zooming in on part of the audio. In a file called `audio_zoom.py`, prompt the user for the name of the WAV file and use `list` slicing to create a list containing only the elements starting at index 79000 and continuing up to but not including index 97000. Do the same for the x-coordinates (in seconds).

1. Prompt the user for the name of the WAV file.
2. Print the range of amplitudes in the new signal.
3. Print the number of samples in the new signal.
4. Print the length (in seconds) of the new signal.
5. Print the start time and end time of the new signal.
6. Plot the new signal.
7. Save the signal as a new file with `_zoom.wav` concatenated to the end of it (as before) and listen to it.
8. It should look like this:



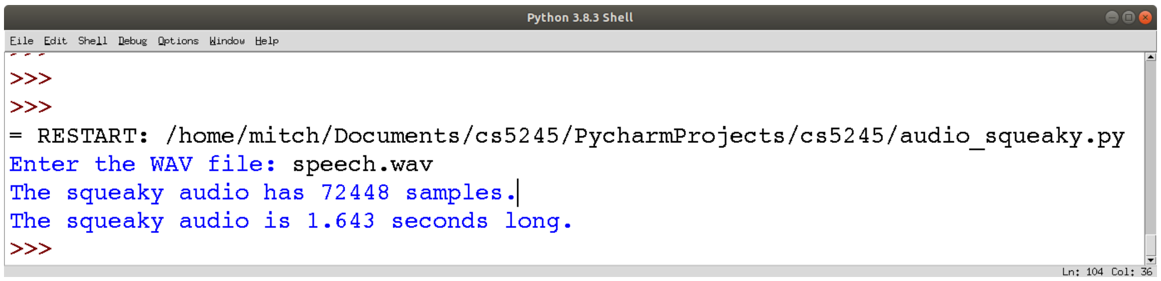
```
== RESTART: /home/mitch/Documents/cs5245/PycharmProjects/cs5245/audio_zoom.py ==
Enter the WAV file name: speech.wav
The selected range is (-7039, 8228).
The selection has 18000 samples.
The selection is 0.408 seconds long.
The selection starts at 1.791 and ends at 2.200.
>>> 
```



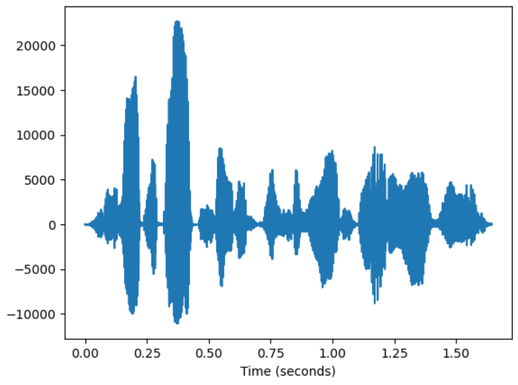
## 6 audio_squeaky

Increasing pitch by skipping samples. In a file called `audio_squeaky.py`, prompt the user for the name of a WAV file and create a list of amplitudes that contains every other sample in the original. For example, if the original data contained the sequence 1, 2, 3, 4, 5; the new sequence would only contain 1, 3, 5. The list of times for the x-axis will only go half as long (starting at zero and going at the same $1/n$ rate for the shortened audio sequence. Because the resulting sequence will be played back at the same rate, the pitch doubles to one octave higher.

1. Prompt the user for the name of the WAV file.
2. Print the number of samples and length in seconds for the squeaky signal.
3. Plot the squeaky signal.
4. Save the signal with `_squeaky.wav` concatenated to the original file name and listen to it.
5. It should look like this:



```
>>>
>>>
= RESTART: /home/mitch/Documents/cs5245/PycharmProjects/cs5245/audio_squeaky.py
Enter the WAV file: speech.wav
The squeaky audio has 72448 samples.
The squeaky audio is 1.643 seconds long.
>>>
```

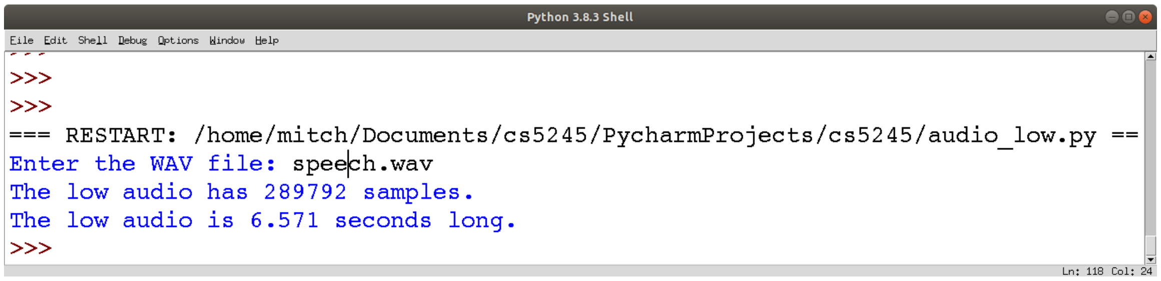


## 7 audio_low

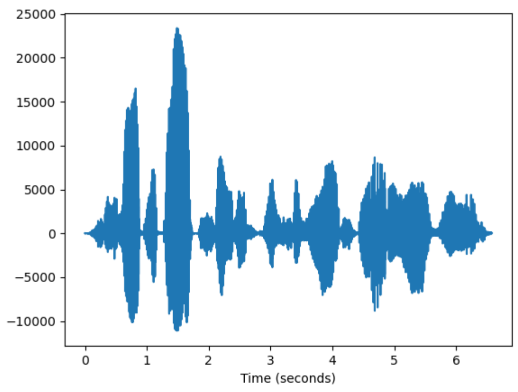
Decreasing pitch by duplicating samples.

In a file called `audio_low.py`, prompt the user for the name of a WAV file and create a list of amplitudes that contains every sample from the original duplicated. For example, if the the original data contained the sequence 1, 2, 3, 4, 5; the new sequence would be 1, 1, 2, 2, 3, 3, 4, 4, 5, 5. The list of times for the x-axis will go twice as long as the original using the same spacing. Because the resulting sequence will be played back at the same rate, the pitch reduces by half to an octave lower.

1. Prompt the user for the name of the WAV file.
2. Print the number of samples and length in seconds for the slow signal.
3. Plot the low signal
4. Save the low signal as `speech_low.wav` and listen to it.
5. It should look like this:



```
>>>
>>>
=== RESTART: /home/mitch/Documents/cs5245/PycharmProjects/cs5245/audio_low.py ==
Enter the WAV file: speech.wav
The low audio has 289792 samples.
The low audio is 6.571 seconds long.
>>>
```



**Submit to Web-CAT to compute your score!**

1. Create a ZIP file for your `cs03` folder by right-clicking the folder and selecting:
   - **Send to** → **Compressed (zipped) folder** on Windows
   - **Compress Items** on MacOS
   - **Compress** on Linux

   You should find the new ZIP file in the same directory where `cs03` resides.
2. Login to `http://webcatvm.cs.appstate.edu:8080/Web-CAT` and submit your ZIP file for grading. You may submit as many times as you want before the deadline.