Combined lab and homework assignment on sound and audio files: Part 1

If you using your own computer please install the library "simpleaudio"

The following commands will test whether the audio file can be played back on your computer. Before you execute them, a word of caution:

1) Plug in your headset and set the volume control to a minimum.

2) Hold it away from your ears and execute the following cell to play back the 35 second audio file.

3) As the song is playing bring the head phone closer to yor ears and increase the volume gradually. Your should hear a very short piece from Mussorgsky's "Pictures of an Exhibition".

```
import simpleaudio as sa

filename = 'Promenade_org.wav'
wave_obj = sa.WaveObject.from_wave_file(filename)
play_obj = wave_obj.play()
play_obj.wait_done()  # Wait until sound has finished playing
```

Combined lab and homework assignment on sound and audio files: Part 2

Execute the following commands at the beginning of for this and all following parts.

```
import matplotlib.pyplot as plt
%config InlineBackend.figure_format = 'retina'
import numpy as np
import simpleaudio as sa
from scipy.io import wavfile

def PrintArrayInfo(a,name=''):
    if (len(name)==0):
        print("Array info:")
    else:
        print("Array:", name)
    print("shape:", a.shape)
    print("dtype:", a.dtype)
    print("min, max:", a.min(), a.max())
    print()
```

Play back this WAV file:

```
filename = 'audio_file_A.wav'
wave_obj = sa.WaveObject.from_wave_file(filename)
play_obj = wave_obj.play()
play_obj.wait_done()  # Wait until sound has finished playing
```

Load and plot the audio information:

```
rate, data = wavfile.read(filename)
print("Data rate [units of Hz = 1/seconds] = ",rate)
PrintArrayInfo(data,"data")

plt.rcParams['figure.figsize'] = [15, 5]
plt.plot(data,'r-')
plt.show()
```

Zoom in and plot only the first 5000 data points.

... 2 points

There are clearly a number of frequencies hidden in this file. You can hear several of them. Now we want to use discrete Fourier transforms to identify those frequencies ν.

$$f^{(sin)}(\nu) = \int_{-\infty}^{+\infty} dt\, x(t)\, \sin(2\pi\nu t) \qquad \text{and} \qquad f^{(cos)}(\nu) = \int_{-\infty}^{+\infty} dt\, x(t)\, \cos(2\pi\nu t)\,.$$

Instead of continuous function, $f(t)$, our audio signal is provides as a series discrete points $x_{j=0\ldots n-1}$. So our sine and cosine transform assume the following discrete forms:

$$f_k^{(sin)} = \sum_{j=0}^{n-1} x_j\, \sin\left\{2\pi\frac{jk}{n}\right\} \qquad \text{and} \qquad f_k^{(cos)} = \sum_{j=0}^{n-1} x_j\, \cos\left\{2\pi\frac{jk}{n}\right\}$$

On a sheet of paper, work out how you convert between time t and index j. More importantly work ot the conversion from frequency index k to frequency ν.

Write a Python function that computes $\left[f_k^{(sin)}\right]^2 + \left[f_k^{(cos)}\right]^2$ for an arbitrary data set and frequency index k.

... 2 points

Construct a array of frequency indicies from 1100 and 1150 and call this function for every single one. Plot the result as function of frequency index.

... 2 points

Now plot the same results as function of frequency in units of Hertz. What was the frequency of the wave have that you saw in the previous plot?

... 2 points

Finally we want to speed this calculations by using use fast Fourier transform (FFT) routines that are provided in the NumyPy library. Cut and paste the following lines and analyze what happens.

```
#For FFT, see https://docs.scipy.org/doc/numpy-1.13.0/reference/routines.fft.html
ft = np.fft.fft(data)
print(ft)
print(ft.shape)

plt.xlim(0,2000)
plt.plot(abs(ft),'o-',lw=2)
plt.show()
plt.xlim(1100,1150)
plt.plot(abs(ft),'o-',lw=2)
plt.show()
```

Analyze the files "audio_file_A_SID.wav" and "audio_file_B_SID.wav". In each case, plot the Fourier spectrum to illustrate all frequencies that are present. Give the frequency in Hertz of the three waves with the largest amplitude.

```
...
```

2 points

Combined lab and homework assignment on sound and audio files: Part 3, a hearing test and three notes

Execute the following commands to play tune A with frequency 440 Hz

```
import numpy as np
import simpleaudio as sa

frequency = 440    # Play a tune A with a frequeny of 440 Hz
fs        = 44100  # 44100 samples per second
seconds   = 2      # Duration of 2 seconds

# Generate array with seconds * sample_rate steps
t = np.linspace(0, seconds, seconds * fs, False)

# Generate a sine wave
note = np.sin(frequency * t * 2 * np.pi)

# Ensure that highest value is in 16-bit range
audio = note * (2**15 - 1) / np.max( np.abs(note) )
# Convert to 16-bit data
audio = audio.astype(np.int16)

# Start playback
play_obj = sa.play_buffer(audio, 1, 2, fs)

# Wait for playback to finish before exiting
play_obj.wait_done()
```

Increase and decrease the frequency until you can no longer hear anything (except for the very beginning and end). IOn the following notebook cell, simply write "I determined my hearing range." If you wish you are welcome to report both limits. This is entirely voluntary, however. We may report discuss your collective responses without mentioning names.

...

Modify the code above to consecutively play three notes of your choice from https://www.seventhstring.com/resources/notefrequencies.html Here is a example how you can calculate the frequencies rather extracting them from table. Specify the names of your notes.

```
# calculate note frequencies
A_freq   = 440
Csh_freq = A_freq * 2 ** (4 / 12)
E_freq   = A_freq * 2 ** (7 / 12)
```

```
...    10 points
```

Combined lab and homework assignment on sound and audio files: Part 4, manipulate Mussorgsky's Promenade

Download the file 'Promenade*very*short_mono.wav' and execute the following commands.

```
import matplotlib.pyplot as plt
%config InlineBackend.figure_format = 'retina'
import numpy as np
import simpleaudio as sa
from scipy.io import wavfile
```

```
def PrintArrayInfo(a,name=''):
    if (len(name)==0):
        print("Array info:")
    else:
        print("Array:", name)
    print("shape:", a.shape)
    print("dtype:", a.dtype)
    print("min, max:", a.min(), a.max())
    print()
```

```
def Rescale(data):
    data2 = np.copy(data)

    dMin = data2.min()
    dMax = data2.max()
    dataRange = dMax-dMin  # warning: if you call this 'range', for loops will stop w
orking

    data2 = (2**15-1) * ( 2.0*(data2-dMin) / dataRange -1.0 )
    data2 = data2.astype(np.int16)

    return data2
```

```
def Play(rate,data):
    # Start playback
    play_obj = sa.play_buffer(data, 1, 2, rate)
    # Wait for playback to finish before exiting
    play_obj.wait_done()
```

```
def Write(filename,rate,data):
    wavfile.write(filename, rate, data)
```

```
rate, data = wavfile.read('Promenade_very_short_mono.wav')
Play(rate,data)
```

Plot the wave amplitudes as we have done in part 2. How long is the file in seconds?

```
...
```

Cut the data short so that only the first 5 notes are being played. This is about 4.1 seconds.

```
...
```

Fourier transform the audio data with "ft = np.fft.fft(data)" and plot the resulting frequencies spectrum in units of Hertz even though you might find it challenging to work in real units. Pick a reasonable upper frequency limit for your plot.

```
...
```

Now remove all frequencies below 1 kHz and above 3 kHz from the spectrum.

```
...
```

Fourier transform the reduced spectrum back into real time with "data2 = np.fft.ifft(ft2)" Then rescale and play the "data2.real" using the functions above.

```
data2 = np.fft.ifft(ft)
data3 = Rescale(data2.real)
Play(rate,data3)
```

Fourier transfrom the original audio file once again. Then scale all frequencies by factor 2, transform it back to real space and play it.

```
...
```

Fourier transfrom the original audio file once again but this time, scale all frequencies by factor 0.25, transform it back to real space and play it.

```
...
```