In [1]:
```
#Para reproducir audio en Jupyter N.
import numpy as np
from IPython.display import Audio, display
from scipy.fft import fft, fftshift
import warnings

warnings.simplefilter("ignore")
```

In [2]:
```
#cantidad de puntos por segundo
framerate = 44100
#cinco segundos de audio
t = np.linspace(0,5,framerate*5)
f = 440
data = np.sin(2*np.pi*f*t)
# data
```

In [3]:
```
def simple_wave(f, t, framerate, A = 1):
    '''Funcion para crear una onda senoidal
        inputs:
        f = frecuencia, int
        t = tiempo, int
        A = amplitud, int
        return:
        wave = onda senoidal, np.array
    '''
    t = np.linspace(0, t, framerate*t)
    return np.sin(2*np.pi*f*t)

la440 = simple_wave(440, 5, 44100)
```

In [4]:
```
Audio(data=la440, rate=44100)
```

Out[4]:

0:00 / 0:05

Distancia entre notas occidentales: intervalos de media nota.

$$y = n_0 * 2^{\frac{n}{12}}$$

In [5]:
```
#Escala musical
n_0 = 440
notas = [440*2**(n/12) for n in range(0, 13)]
# notas
```

In [6]:
```
#función senoidal para cada nota
funciones_nota = []
for note_frequency in notas:
     funciones_nota.append(simple_wave(note_frequency, 2, 44100))
# funciones_nota
```

In [7]:
```
#encontrar notas de la escala pentatónica mayor de c
# la, La#, si, do, do#, re, re#, mi, fa, fa#, sol, sol#, la
```

In [8]:
```
index_notas = [0,4,5,7,11,12]
scala_pentatonica = [funciones_nota[i] for i in index_notas]
# scala_pentatonica
```

In [9]:
```
Audio(data=scala_pentatonica[1], rate = 44100)
```

Out[9]:

0:00 / 0:02

In [10]:
```
#juntamos las notas de la pentatónica
audio_concatenado = np.concatenate(scala_pentatonica)
Audio(audio_concatenado, rate=44100)
```

Out[10]:

0:00 / 0:12

Con grabaciones reales

In [11]:
```
import matplotlib.pyplot as plt
plt.style.use('seaborn-darkgrid')
%matplotlib inline
```

In [12]:

```
def fourier_calculation(y, framerate):
    '''Aplicando fast fourier transform'''
    yf = fft(y)
    N = len(y)
    yf = 2.0/N *np.abs(yf[0:N//2])
    xf = np.linspace(0.0, 1/2*framerate, N//2)
    return yf, xf

def plot_multiple(y, fs = 44100, plot_type = 'plotly'):
    '''Plotea primero la onda en dominio de tiempo, luego el espectro (FFT) y luego el espectrograma'''

    #Creo el espacio para plotear, una "figura" vacia
    plt.figure(figsize=(15,15))
    x = np.linspace(0, len(y)/fs, len(y))
    #Dividido en 3 filas y 1 columna, ploteo la onda en el 1er espacio
    plt.subplot(3, 1, 1)
    #plt.xlim([0, 0.005])
    plt.plot(x, y)
    #Pongo titulo al eje y
    plt.ylabel('Amplitude')
    #Grilla de fondo
    plt.grid()

    #FFT
    #n = len(w) = duración * framerate
    yf, xf = fourier_calculation(y, fs)
    #Plot FFT
    plt.subplot(3, 1, 2)
    #Ploteo las frecuencias positivas y sus valores, con un color RGBA
    plt.xlabel('Freq (Hz)')
    plt.ylabel('|Y(freq)|')
    plt.xlim([0,2000])
    plt.plot(xf, yf)


    #plot spectogram
    if plot_type == 'scipy':
        plt.subplot(3, 1, 3)
        f, t, Sxx = signal.spectrogram(y, fs, scaling='density')
        plt.pcolormesh(t, f, Sxx, shading='gouraud', cmap='jet_r')
        #plt.specgram(y, Fs =fs)
        plt.ylabel('Frequency [Hz]')
        plt.xlabel('Time [sec]')
        plt.ylim([0,2000])
        plt.show()
    else:
        plt.subplot(3, 1, 3)
        a = plt.specgram(y, Fs=fs, cmap='jet')
        plt.ylim([0,3000])
        plt.ylabel('Frequency [Hz]')
        plt.xlabel('Time [sec]')
        plt.colorbar()
        plt.show()
```

In [13]:

```
from scipy.io import wavfile
import scipy.io
```

In [14]:

```
samplerate, data = wavfile.read('fondo.wav')
print(f"number of channels = {data.shape[1]}")

length = data.shape[0] / samplerate
print(f"length = {length}s")
```

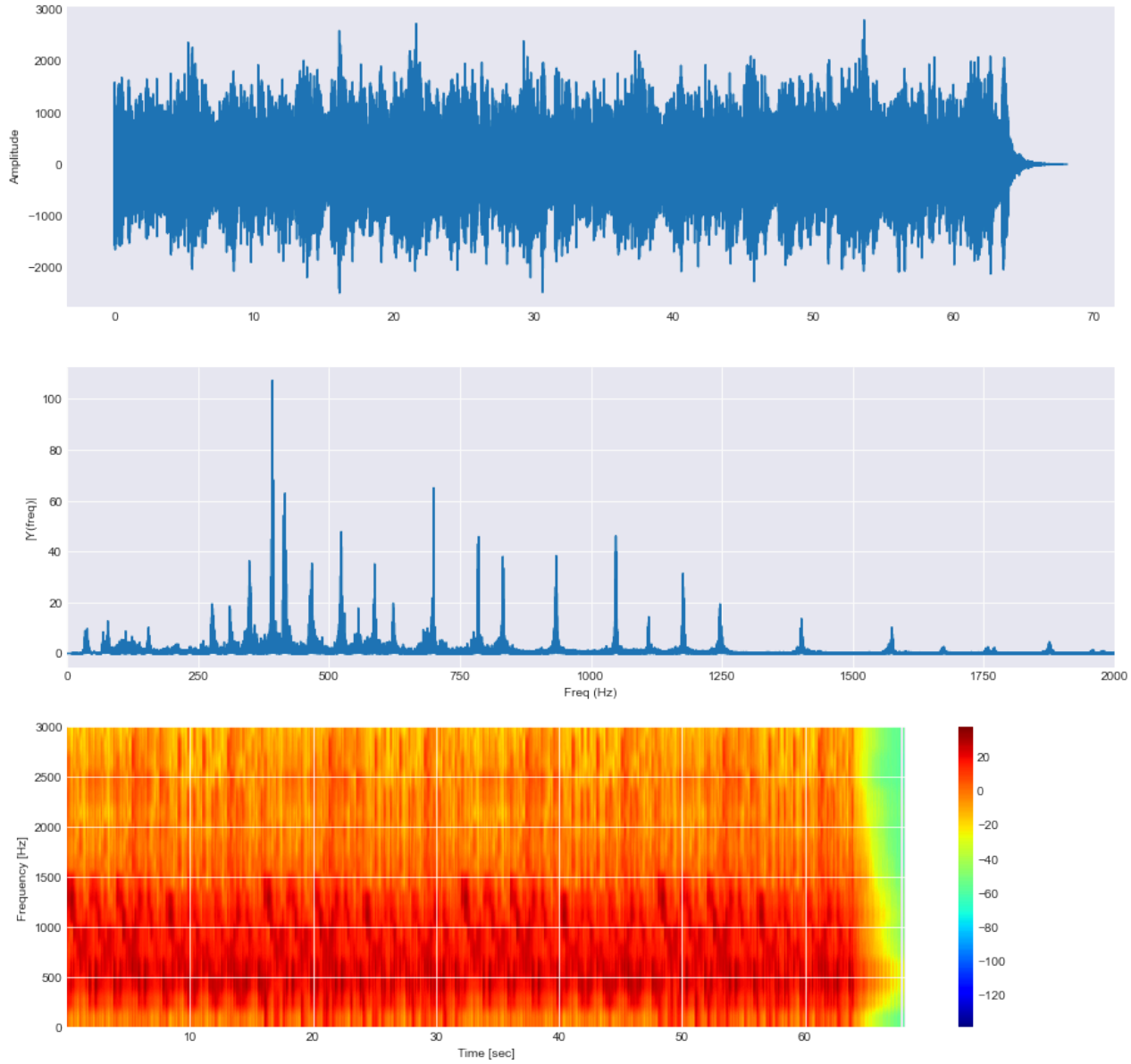
```
number of channels = 2
length = 68.15056689342404s
```

In [15]:

```
#stereo to mono
data = data.sum(axis=1) / 2
```

In [16]:

```
1 plot_multiple(data, samplerate)
```



Posibles aplicaciones

- Speech to face (https://speech2face.github.io/)
- Fire detection (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6929188/pdf/sensors-19-05093.pdf)