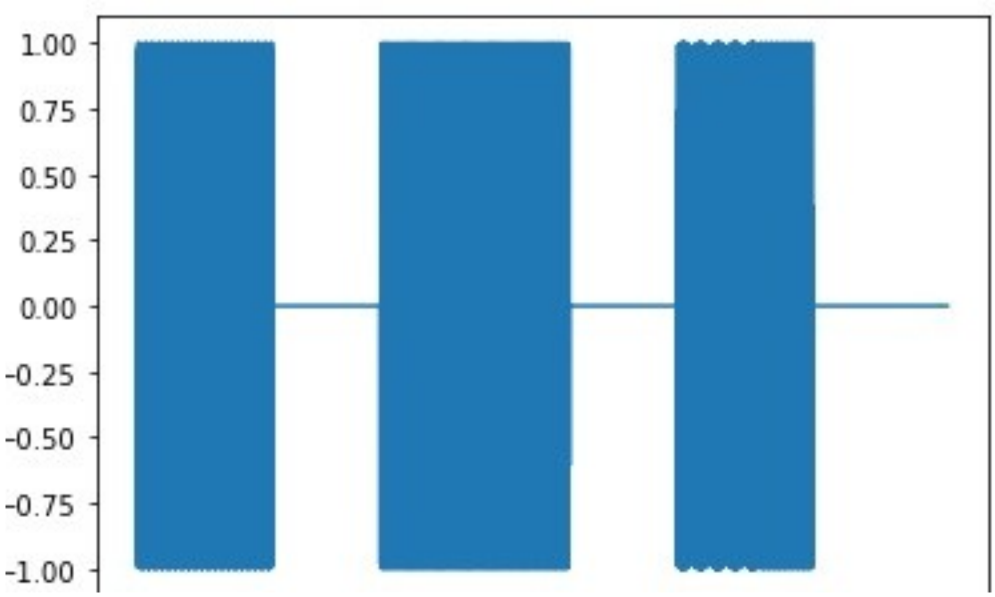
Report Signals Project

Done by:
Mayar Bahnacy 52-7251
Seif Ashraf Ahmed 52-5119

Milestone 1



First, we put the frequencies we're utilizing in an array list called "Fi" that correlates to the left hand, and then we added the frequencies that correspond to the right hand to the list "fi." However, we didn’t use the right hand, so all the frequencies in “fi” were zeroes. The time period of each frequency was then added to the array list "Ti," and the array list "ti" was made up of the values “Ti” added cumulatively. Then, using a for loop that finishes until “i” is more than the size of the "Fi" array list, we iterate over the four lists, placing them in the function, which was explained later, and summing them to "s," which we previously constructed by equating it to the function but instead of inserting the frequencies, we inserted 0. Finally, we used plt.plot to plot the song on a graph, and sd.play to play it.

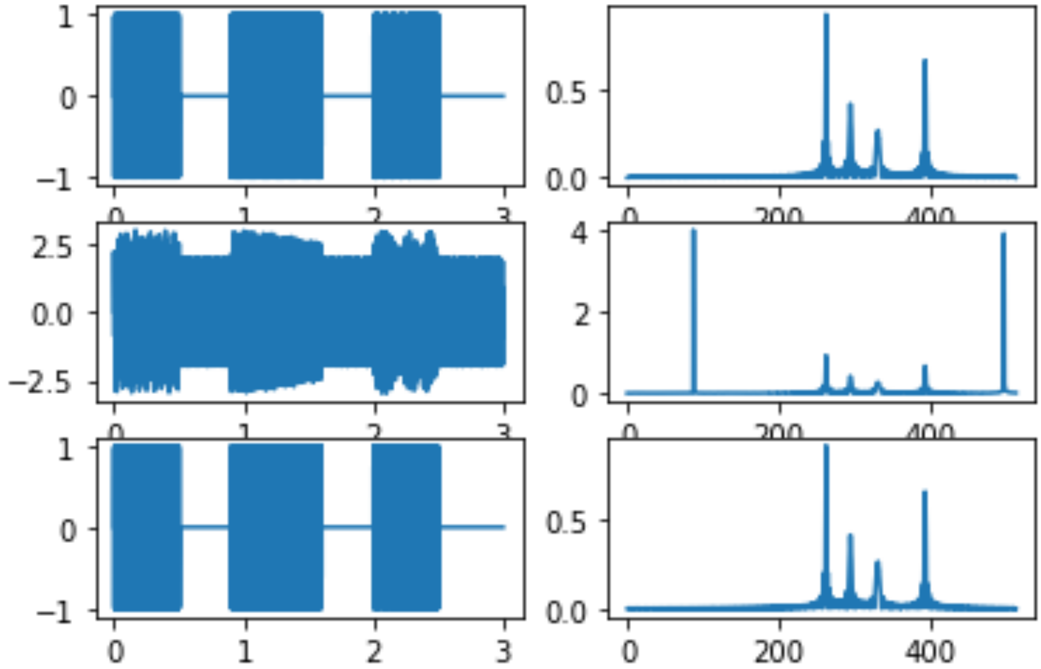
The Function:

$$x(t) = \sum_{i=1}^{N} [\sin(2\pi \mathrm{F}_i t) + \sin(2\pi f_i t)]\,[u(t - t_i) - u(t - t_i - T_i)]$$

To construct this function, we used the predefined function np.multiply to multiply the part which contains the sine and the part which contains the unit step function. Also, we used the predefined function np.pi to insert the (2πt) into the sine. Finally, we generated the unit step function by defining a method which takes the array “t”and returns 1 times the values of t which are greater than zero.

Milestone 2

The first row is the original song and its transformation in the frequency domain.
The second row is the song plus the noise and its transformation in the frequency domain.
The third row is the filtered song and its transformation in the frequency domain



First, we started off by creating an array which contains the frequency axis, which was the 3rd and 4th octave. Secondly, we transformed the signal of the original song and calculated it in the frequency domain using fft(s) and called it sf. Then, we generated two random frequencies, which are also in the 3rd and 4th octave, using np.random and used them in this function to generate the noise n = sin(2fn1πt) + sin(2fn2πt), and then we added this function to the original song to create the function xn, which is the song plus the noise. Those were the steps conducted to generate the noise.
As for the noise cancellation, first, we transformed the signal of the noisy song and calculated it in the frequency domain using fft(xn) and called it xnf. Then, we saved the rounded maximum number of sf in a variable called max and created an empty list called ind1. After that, we looped over the length of xnf, and compared each element of xnf to max. If the element was greater than the maximum, we appended its index to the empty list we created. After exiting the loop, we would call the first and second elements of ind1, which corresponds to the largest two numbers of xnf which are greater than the largest number in sf, and get their values in the frequency domain, round them up and save them in two variables. Finally, we created the function xfilter which takes the noisy song and removes the noise using the two variables we saved previously and adding them in this function xfilter = xn - (sin(2fn11πt) + sin(2fn22πt)). Also, we transformed signal of xfilter and calculated it in the frequency domain using fft(xfilter) and called it xfilterf to make sure that the noise was removed.