In the first task I created the function ‘Sampling\_Func’ in order to solve the task that was asked. The parameters of the function are the time period, T, the frequency at which we want to sample, v, the function we are evaluating, f, and an optional parameter, d, which decides which code shall be executed. I created a graph which shows the shape of the function and the frequency at which samples were taken, which are denoted by the ‘x’ symbol on the graph. When I run the code for the two frequencies given, 8Hz and 50Hz, I can see that the ‘x’ on the graph that denotes where the samples taken appear more frequently at 50Hz than 8Hz because the number of samples taken is equal to the time period, T, multiplied by the sampling frequency.

For the first and second DIY tasks in the fourier analysis section, I created a function ‘Dft\_Func’ which takes the sample points, x, the time period, T, and the optional variable, d, which decides if a specific block of code should be executed. The function takes a set of data points, x, and calculates the coefficients of the fourier analysis transform using the equations given in the powerpoint. If the optional variable, d, is equal to 0 then a graph will be plotted using the coefficients calculated before. This graph shows the frequencies which are most present in the function that is to be evaluated. When using the data points given in the course materials, the graph that is shown is frequency vs magnitude and there are straight lines that go upwards at specific frequencies and have different magnitudes, this shows that the function works.

For the third DIY task in the discrete fourier analysis section I combined the ‘Sampling\_Func’ and ‘Dft\_Func’ using the optional variable, d, stated above. This variable allows me to create sample points for the function that is going to be evaluated and then compute the fourier analysis and output a graph of frequency vs magnitude. The d variable in the ‘Sampling\_Func’ will call the ‘Dft\_Func’ if it is equal to 1. When using the functions given by the powerpoint and inputting them into the function to create the frequency vs magnitude graph I get a graph that is very weird and doesn’t have straight lines like in DIY task 1 & 2 but it is clear which frequencies are present. I think this is because there are not enough sampled points that the function is being evaluated at.

The Fourth DIY task for the discrete fourier analysis section was very straight forward. I used the inbuilt fast fourier transform, ‘fft’, function. I used the sample data points which were given in the course materials and evaluated them using ‘fft()’, then I took the absolute values of the output and plotted them on a graph. The graph looks exactly the same as the one I got for DIY task 1 & 2, where there are clear straight lines showing which frequencies are present.

In the task that asks to implement the Cooley-Tuckey method I created a function, ‘ctfft’, which takes just the sample data points as a parameter. The function then checks if the logarithm to the base 2 of the size of the array of data points is an integer, because if it is not then then the method will not work. In this method I had to get the odd and even indexes from the array that holds the data points, and to do this I used recursion. The even and odd indexes are then stored in another array and then they are used in further computation. This method is another way of doing the discrete fourier transform except it is much faster computationally then the one that is coded in tasks 1 & 2. When I use a data set of the size 2^7 I get a successful output, an array filled with complex numbers.

I tried implementing the Beethoven task but unfortunately the size of the audio file was too large even hence I compressed it using a website from the internet to 1% of the original size. I could not figure out how to reduce the noise from the graph when I passed the data points into ‘Dft\_Func’.