1)
Our script takes in two arrays xt and yt that represent two functions that we would like to find the convolution of. It also takes in a delta variable which represents the sampling interval, or in other words, how far apart (time) each point in the xt and yt arrays are. As an additional input, users can specify where the array begins with respect to time. So for example, if a user wanted to enter a signal, xt, that was a value of 1 for 103.5 <= t <= 104.5 with a sampling delta of 0.5, they could simply enter xt = [1 1 1] with the x_lowerbound variable set to 103.5. This can also be repeated with yt and it's lower bound variable y_lowerbound.

The script then calculates the approximated convolution of the two signals and produces a graph that shows this resulting approximated signal. So how does it do the convolution? The first step taken is calculating the range of t values that result in none zero summations. This is done by finding the bounds of xt and yt, then finding the bottom time limit which is the time shift resulting in only one element of yt overlapping with xt's first value. Finding the top time limit is done by finding the time shift resulting in only one element of yt overlapping with xt's last value. Based on the delta, we then create an array that holds all the time values (t_convolution) that must be plugged into our approximation. To compute each approximation, yt has zero's the length of xt added onto each side so that we have one array that we can grab the values that would line up with xt for every given shift that produces non-zero values. To start the convolution, the flipped yt array is shifted one to the right so that one value of the original yt function overlaps with xt and the final numbers (length of xt) are assigned to a new array (yt_resizedAndBeautiful) as these are the values that correspond to the xt for the first shift. Each element in xt and yt_resizedAndBeautiful are multiplied and then the results are summed to calculate the approximation for the first time value. This result is assigned to a new array (zt) in its first spot. This approximation process is repeated until there is a corresponding zt value for every time value in the t_convolution array. Note, the difference in each approximation is the yt array with the zero's added (yt_convolution) is shifted one more to the right meaning the values put into yt_resizedAndBeautiful are also shifted one to the right. Final, t_convolution and zt are plotted displaying the approximated convolution!

2a)
Our function takes a value of k for a given signal and can output the associated ak value. We do this by creating a midpoint approximation function that estimates the integral of the given function. The inputs to our function are: k, angular frequency (w), and the increment size (dx). We calculate the period of the signal using the angular frequency, and then we define our time vector in terms of the

calculated period.
We find the midpoint of each of the dx sections and then pass that value into a separate function that holds our signal to calculate the y-point on the curve at that time, t. We then multiply this result by the width in order to get the area of the small section. Then, we store each area in an array, then cumulatively sum the array to find the total area under the curve. The result is a complex number that approximates ak for the given k value.

2b)
Our function takes an array of k values and their associated ak values and plots the function using the Continuous Time Fourier Series synthesis equation. The inputs to our function are: array of k values, array of ak values (must be the same length as the array of k values), angular frequency (w), and a vector storing the values of time (x-axis).
Our synthesis function calculates the number of k values (and ak values). Using the time vector, we calculate the y-value at every point of the signal corresponding to each ak (and k) value, and add it to the new synthesized signal. This is done by looping through the k values. Our synthesized signal is then plotted on a time vs. amplitude graph.