Task 1

For task 1 I started by reading about the Vigenere cipher on wikipedia and experimenting with encoding and decoding on https://www.dcode.fr/vigenere-cipher. Once I had a good understanding I started coding. I created a function that took the plaintext and cipher key and for each letter in the plaintext it would add the appropriate index of the key only if the character was in the alphabet. I also checked if the character was uppercase or lowercase and after adding the appropriate index of the key I would check for overflow and subtract -26 to bring it back into the ASCII range. Below the function is the main function which takes input and puts it into the function.

Task 2

For task 2 I did basically the same thing as for task 1 with the difference that I had to change it so that instead of adding the cipher key to the plaintext I subtracted the cipher key from the ciphertext. I also made sure that instead of checking for overflow I checked for underflow after subtracting the cipher key.

Task 3

For task 3 I did a lot of reading about frequency analysis which started by reading about the different methods on wikipedia and then looking more in depth on different websites about the different methods. I decided to use the method that calculated the chi squared value for the different caesar ciphers of the frequency of the ciphertext against the overall frequency of the english alphabet and then returned which caesar cipher got the lowest chi squared value. This is represented in my code by 2 functions. The getKey function takes the ciphertext and the cipher key length and for each index of the cipher key it will get all the indices in the ciphertext that get encrypted with the index of the cipher key and create a frequency array where the index of the array is a letter of the alphabet and the value is the percentage of occurrence in the ciphertext. Then the frequency array gets sent to the freqAnalysis function which calculates the chi square value of the array for each caesar cipher against the english frequencies which are represented by an array. The alphabet value for the caesar cipher with the lowest chi square value is then sent to

the getKey function which will put all the alphabet values together to form the key. This alphabet value is determined by how many times the array was shifted over as each shift corresponds to encrypting the ciphertext by the next letter.

Task 4

For task 4 I started by looking on wikipedia for methods of getting the key length based on the ciphertext. I found Kasaki's examination and Friedman's test and researched both methods until I decided upon Friedman's test due to more people recommending it. In the task4 code that i have there are two functions, getKeyLen() and getIOC(), where getKeyLen() relies on getIOC(). getKeyLen() starts off by reformatting the ciphertext to only capital uppercase letters and making sure that the max length of the cipher is no more than the ciphertext but this code serves no purpose because we can't tell the cipher key when it's the same length due to how a one-time pad works. Next, for each key length from 1 to 100 (since the professor specified 100 will be the max), I split the ciphertext into the respective parts that correspond to each cipher key index (for example 'helloprofessor' for the key 'hi' would be split into 'hlorfso' and 'elpoesr'). For each split I got the index of coincidence using the getIOC function which works by using the formula in Friedman's test. Then I averaged the index of coincidence values and added them to a table. I then iterated through the table in terms of key length (the lower key length first) and picked the first value that fit within the range of 0.0667 +-0.006. The 0.0667 value is the index of coincidence value for the english language and then I used 0.006 to get the values that were just within the range. I decided upon 0.006 after testing a while because I found that if it was too big then it would pick up a shorter key length and if it was too short then it would pick up a longer key length. I decided to pick the first occurrence because I found that later occurrences that fit the range occurred on n * key_length lengths where n is a whole number. This method works really well for medium to long ciphertexts but experiences trouble with short ciphertexts since there aren't enough letters to perform an accurate Friedman test. I tested Kasaki's examination with short ciphertexts and they also experienced this issue so I left it out of my code.