**Part A**

User inputs 3 codons with combinations of A, G, C, T. If the input contains unknown characters or more than 3 inputs the user is prompted to correct their statement with hints. The characters entered are checked by using a while loop that is initiated if there are any characters entered other than A, G, C, or T. The message: “Please enter a triplet of character only containing A,C,T,G (ex. ACT)” is displayed until the user enters a triplet with correctly inputted characters. The length of the triplet is also checked using an if statement inside the while loop. If the length of the triplet is more or less than three, the user is presented with the message “Input does not contain the required 3 characters.” After a correct triplet is entered, it is ran through the sequence using a while loop to find and calculate the total number of triplets in the sequence (which is the length of the matrix divided by 3). The positions of the first ten occurrences are calculated using if else statements. All of the results are written in a document called repor\_count.txt, which includes the total number of occurrences, and the positions of the first 10 occurrences. We did multiple error checks with different combinations of triplets and all come out with valid answers. The downside to this program is that it does take two minutes to execute.

**Part B**

In the first part to problem B, we created a code that found the location of the first start codon in the short sequence provided. We then searched for all the locations of the end codons. For our next calculations, we needed only the location of the first appearing start codon, so we took the minimum of those results. As far as the locations of the end codons, we only need the location of the first appearing codon after the start codon. We created a while loop that found the first positive difference between the minimum start codon location and the end codons. A review file was created to clearly and coherently display the information found. Our script was tested by comparing its results to the results we did through manual calculations. We also ran “made up” values through our script and made sure the results were correct. To the best of our knowledge, there is nothing wrong with this script. The only user error that could occur is importing the wrong file.

For the long codon the process, we began by searching for the start codon locations and sorting similar to the process above. To prevent future overlapping of codons, we replaced all the start codons with ‘\*\*\*’ by using the strrep function. Next, the end codon locations are found and replaced by ‘\*\*\*’ one at a time so no overlapping may occur. The end codon locations are then sorted. We created a while loop to look for the start and end of genes. The start codon and end codon arrays were then incremented. An elseif statement was used to check if the end codon came after the start codon. If it didn’t, the end codon was incremented by one, and it was checked again. Once an end codon that appears after the start codon was found, the locations were recorded. After the loop ran out of data to check, it displays, ‘ran out of data to check.” The results are displayed in a file called report\_longnew.txt.