

PYTHON CODE AS RAN:

```
from itertools import combinations
from collections import Counter
```

```
input_sequence =
```

```
"ATGAAAAACCCAAAAAGAAATCCGGAGGATTCCGGATTGTCAATATGCTAAACGCGGAGTAGCCCGTGTGAG
CCCCTTTGGGGGCTTGAAGAGGCTGCCAGCCGGACTTCTGCTGGGTCATGGGCCCATCAGGATGGTCTTGCG
ATTCTAGCCTTTTTGAGATTCACGGCAATCAAGCCATCACTGGGTCTCATCAATAGATGGGGTTCAGTGGGGAAA
AAAGAGGCTATGGAAATAATAAAGAAGTTCAAGAAAGATCTGGCTGCCATGCTGAGAATAATCAATGCTAGGAAG
GAGAAGAAGAGACGAGGCGCAGATACTAGTGTGGAATTGTTGGCCTCCTGCTGACCACAGCTATGGCAGCGG
AGGTCACTAGACGTGGGAGTGCACTATATGTACTTGGACAGAAACGACGCTGGGGAGGCCATATCTTTTCCA
ACCACATTGGGGATGAATAAGTGTTATATACAGATCATGGATCTTGGACACATGTGTGATGCCACCATGAGCTAT
GAATGCCCTATGCTGGATGAGGGGGTGAACACAGATGACGTCGATTGTTGGTGCAACACGACGTCAACTTGGG
TTGTGTACGGAACCTGCCATCACAAAAAGGTGAAGCACGGAGATCTAGAAGAGCTGTGACGCTCCCCTCCCAT
TCCACTAGGAAGCTGCAAACGCGGTCGCAAACCTGGTTGGAATCAAGAGAATACACAAAGCACTTGATTAGAGT
CGAAAAATTGATATTCAGGAACCCCTGGCTTCGCGTTAGCAGCAGCTGCCATCGCTTGGCTTTTGGGAAGCTCAA
CGAGCCAAAAAGTCATATACTTGGTCATGATACTGCTGATTGCCCCGGCATAACAGCATCAGGTGCATAGGAGTC
AGCAATAGGGACTTTGTGGAAGGTATGTCAGGTGGGACTTGGGTTGATGTTGTCTTGGACATGGAGGTTGTGT
CACCGTAATGGCACAGGACAAACCGACTGTCGACATAGAGCTGGTTACAACAACAGTCAGCAACATGGCGGAG
GTAAGATCCTACTGCTATGAGGCATCAATATCGGACATGGCTTCGGACAGCCGCTGCCAACACAAGGTGAAGC
CTACCTTGACAAGCAATCAGACACTCAATATGTCTGCAAAAGAACGTTAGTGACAGAGGCTGGGGAAATGGAT
GTGGACTTTTTGGCAAAGGGAGCCTGGTGACATGCGCTAAGTTTGCATGCTCCAAGAAAATGACCGGGAAGAGC
ATCCAGCCAGAGAATCTGGAGTACCGGATAATGCTGTGAGTTCATGGCTCCCAGCACAGTGGGATGATCGTTAA
TGACACAGGACATGAACTGATGAGAATAGAGCGAAGGTTGAGATAACGCCCAATTCACCAAGAGCCGAAGCCA
CCCTGGGGGGTTTTTGAAGCCTAGGACTTGATTGTGAACCGAGGACAGGCCTTGACTTTTCAGATTTGTATTACT
TGACTATGAATAACAAGCACTGGTTGGTTCACAAGGAGTGGTTCCACGACATTCCATTACCTTGGCAGCGTGGG
GCAGACACCGGAATCCCACTGGAACAACAAGAAGCACTGGTAGAGTTCAAGGACGCACATGCCAAAAGGC
AAACTGTCGTGGTTCTAGGGAGTCAAGAAGGAGCAGTTCACACGGCCCTTGCTGGAGCTCTGGAGGCTGAGAT
GGATGGTGCAAAGGGAAGGCTGTCTCTGGCCACTTGAAATGTGCGCTGAAAATGGATAAACTTAGATTGAAGG
GCGTGTCACTACTCCTTGTGTACCGCAGCGTTCACATTCACCAAGATCCCGGCTGAAACACTGCACGGGACAGTC
ACAGTGGAGGTACGTACGCAGGGACAGATGGACCTTGCAAGGTTCCAGCTCAGATGGCGGTGGACATGCAAA
CTCTGACCCCAGTTGGGAGGTTGATAACCGCTAACCCCGTAATCACTGAAAGCACTGAGAACTCTAAGATGATG
CTGGAACCTTGATCCACCATTGTTGGGACTCTTACATTGTGATAGGAGTCGGGGAGAAGAAGATCACCCACCACTG
GCACAGGAGTGGCAGCACCATTTGAAAAGCATTGTAAGCCACTGTGAGAGGTGCCAAGAGAATGGCAGTCTTG
GGAGACACAGCCTGGGACTTTGGATCAGTTGGAGGCGCTCTCAACTCATTGGGCAAGGGCATCCATCAAATTTT
TGGAGCAGCTTTCAAATCATTGTTTGGAGGAATGTCCTGGTTCTCACAAATTCTCATTGGAACGTTGCTGATGTG
GTTGGGTCTGAACACAAAGAATGGATCTATTTCCCTTATGTGCTTGGCCTTAGGGGGAGTGTTGATCTTCTTATC
CACAGCTGTCTCTGCTGATGTGGGGTGCTCGGTGACTTCTCAAAGAAGGAGACGAGATGCGGTACAGGGGTG
TTCGTCTATAACGACGTTGAAGCCTGGAGGGACAGGTACAAGTACCATCCTGACTCCCCCGTAGATTGGCAGC
AGCAGTCAAGCAAGCCTGGGAAGATGGTATCTGTGGGATCTCCTCTGTTTCAAGAATGGAACATCATGTGGA
GATCAGTAGAAGGGGAGCTCAACGCAATCCTGGAAGAGAATGGAGTTCAACTGACGGTCGTTGTGGGATCTGTA
AAAAACCCCATGTGGAGAGGTCCACAGAGATTGCCCGTGCCTGTGAACGAGCTGCCCCACGGCTGGAAGGCTT
GGGGGAAATCGTACTTCGTGAGAGCAGCAAAGACAAATAACAGCTTTGTGCTGGATGGTGACACACTGAAGGAA
TGCCCACTCAAACATAGAGCATGGAACAGCTTTCTTGTGGAGGATCATGGGTTTCGGGGTATTTCACTAGTGTG
TGGCTCAAGGTTAGAGAAGATTATTCATTAGAGTGTGATCCAGCCGTTATTGGAACAGCTGTTAAGGGAAAGGAG
GCTGTACACAGTGTCTAGGCTACTGGATTGAGAGTGAGAAGAATGACACATGGAGGCTGAAGAGGGCCCATCT
GATCGAGATGAAAACATGTGAATGGCCAAAGTCCACACATTGTGGACAGATGGAATAGAAGAGAGTGATCTGA
```

TCATACCCAAGTCTTTAGCTGGGCCACTCAGCCATCACAATACCAGAGAGGGCTACAGGACCCAAATGAAAGGG
CCATGGCACAGTGAAGAGCTTGAAATTCGGTTTGAGGAATGCCAGGCACTAAGGTCCACGTGGAGGAAACATG
TGGAACAAGAGGACCATCTCTGAGATCAACCACTGCAAGCGGAAGGGTGATCGAGGAATGGTGCTGCAGGGAG
TGCACAATGCCCCCACTGTCGTTCCGGGCTAAAGATGGCTGTTGGTATGGAATGGAGATAAGGCCCAGGAAAGA
ACCAGAAAGTAACCTTAGTAAGGTCAATGGTGAAGTGCAGGATCAACTGATCACATGGATCACTTCTCCCTTGGAGT
GCTTGTGATTCTGCTCATGGTGCAGGAAGGGCTGAAGAAGAGAATGACCACAAAGATCATCATAAGCACATCGA
TGGCAGTGCTGGTAGCTATGATCCTGGGAGGATTTTCAATGAGTGACCTGGCTAAGCTTGCAATTTTGATGGGT
GCCACCTTCGCGGAAATGAACACTGGAGGAGATGTAGCTCATCTGGCGCTGATAGCGGCATTCAAAGTCAGACC
AGCGTTGCTGGTATCTTTCATCTTCAGAGCTAATTGGACACCCCGTGAAAGCATGCTGCTGGCCTTGGCCTCGT
GTCTTTTGCAAACCTGCGATCTCCGCCTTGAAGGCGACCTGATGGTTCTCATCAATGGTTTTGCTTTGGCCTGGT
TGGCAATACGAGCGATGGTTGTTCCACGCACTGATAACATCACCTTGGCAATCCTGGCTGCTCTGACACCACTG
GCCCCGGGGCACACTGCTTGTGGCGTGGAGAGCAGGCCTTGCTACTTGCGGGGGGTTTATGCTCCTCTCTCTGA
AGGGAAAAGGCAGTGTGAAGAAGAACTTACCATTTGTCATGGCCCTGGGACTAACCGCTGTGAGGCTGGTCTGA
CCCCATCAACGTGGTGGGACTGCTGTTGCTCACAAGGAGTGGGAAGCGGAGCTGGCCCCCTAGCGAAGTACTC
ACAGCTGTTGGCCTGATATGCGCATTGGCTGGAGGGTTCGCCAAGGCAGATATAGAGATGGCTGGGCCCATGG
CCGCGTCCGGTCTGCTAATTGTCAGTTACGTGGTCTCAGGAAAGAGTGTGGACATGTACATTGAAAGAGCAGGT
GACATCACATGGGAAAAAGATGCGGAAGTCACTGGAAACAGTCCCCGGCTCGATGTGGCGCTAGATGAGAGTG
GTGATTTCTCCCTGGTGGAGGATGACGGTCCCCCATGAGAGAGATCATACTCAAGGTGGTCTGATGACCATC
TGTGGCATGAACCAATAGCCATACCCTTTGAGCTGGAGCGTGGTACGTATACGTGAAGACTGGAAAAAGGAG
TGGTGCTCTATGGGATGTGCCTGCTCCCAAGGAAGTAAAAAGGGGGAGACCACAGATGGAGTGTACAGAGTA
ATGACTCGTAGACTGCTAGGTTCAACACAAGTTGGAGTGGGAGTTATGCAAGAGGGGGTCTTTCACACTATGTG
GCACGTCACAAAAGGATCCGCGCTGAGAAGCGGTGAAGGGGAGACTTGATCCATACTGGGGAGATGTCAAGCAG
GATCTGGTGTCACTACTGTGGTCCATGGAAGCTAGATGCCGCTGGGACGGGCACAGCGAGGTGCAGCTCTTGG
CCGTGCCCCCGGAGAGAGAGCGAGGAACATCCAGACTCTGCCCGGAATATTTAAGACAAAGGATGGGGACAT
TGGAGCGGTTGCGCTGGATTACCCAGCAGGAACCTCAGGATCTCCAATCCTAGACAAGTGTGGGAGAGTGATAG
GACTTTATGGCAATGGGGTCGTGATCAAAAATGGGAGTTATGTTAGTGCCATCACCCAAGGAGGAGGGAGGAA
GAGACTCCTGTTGAGTGCTTCGAGCCTTCGATGCTGAAGAAGAAGCAGCTAACTGTCTTAGACTTGCATCCTGG
AGCTGGGAAAACCAGGAGAGTTCTTCTGAAATAGTCCGTGAAGCCATAAAAAACAAGACTCCGTACTGTGATCTT
AGCTCCAACCAGGGTTGTCGCTGCTGAAATGGAGGAAGCCCTTAGAGGGGCTTCCAGTGCGTTATATGACAAACAG
CAGTCAATGTCACCCACTCTGGAACAGAAATCGTCGACTTAATGTGCCATGCCACCTTCACTTCACGTCTACTAC
AGCCAATCAGAGTCCCCAACTATAATCTGTATATTATGGATGAGGCCCACTTCACAGATCCCTCAAGTATAGCAG
CAAGAGGATACATTTCAACAAGGGTTGAGATGGGCGAGGCGGCTGCCATCTTCATGACCGCCACGCCACCAGG
AACCCGTGACGCATTTCCGGACTCCAACCTACCAATTATGGACACCGAAGTGGAAAGTCCCAGAGAGAGCCTGGA
GCTCAGGCTTTGATTGGGTGACGGATCATTCTGAAAAACAGTTTGGTTTGTCCAAGCGTGAGGAACGGCAAT
GAGATCGCAGCTTGTCTGACAAAGGCTGGAAAACGGGTCAACAGCTCAGCAGAAAGACTTTTGAGACAGAGTT
CCAGAAAACAAAACATCAAGAGTGGGACTTTGTCGTGACAACTGACATTTAGAGATGGGCGCCAACCTTTAAAGC
TGACCGTGTATAGATTCCAGGAGATGCCTAAAGCCGGTCATACTTGATGGCGAGAGAGTCACTTGGCTGGAC
CCATGCCTGTACACATGCCAGCGCTGCCAGAGGAGGGGGCGCATAGGCAGGAATCCCAACAAACCTGGAGA
TGAGTATCTGTATGGAGGTGGGTGCGCAGAGACTGACGAAGACCATGCACACTGGCTTGAAGCAAGAATGCTC
CTTGACAATATTTACCTCCAAGATGGCCTCATAGCCTCGCTCTATCGACCTGAGGCCGACAAAGTAGCAGCCATT
GAGGGAGAGTTCAAGCTTAGGACGGAGCAAAGGAAGACCTTTGTGGAACCTCATGAAAAGAGGAGATCTTCTGT
TTGGCTGGCCTATCAGGTTGCATCTGCCGGAATAACCTACACAGATAGAAGATGGTGCTTTGATGGCACGACCA
ACAACACCATAATGGAAGACAGTGTGCCGGCAGAGGTGTGGACCAGACACGGAGAGAAAAGAGTGCTCAAACC
GAGGTGGATGGACGCCAGAGTTTGTTCAGATCATGCGGCCCTGAAGTCATTCAAGGAGTTTGCCGCTGGGAAAA
GAGGAGCGGCTTTTGAGTGATGGAAGCCCTGGGAACACTGCCAGGACACATGACAGAGAGATTCCAGGAAGC
CATTGACAACCTCGCTGTGCTCATGCGGGCAGAGACTGGAAGCAGGCCTTACAAAGCCGCGGCGGCCCAATTG
CCGGAGACCCTAGAGACCATTATGCTTTTGGGGTTGCTGGGAACAGTCTCGCTGGGAATCTTTTTCGTCTTGATG
AGGAACAAGGGCATAGGGAAGATGGGCTTTGGAATGGTGAAGTCTTGGGGCCAGCGCATGGCTCATGTGGCTCT

CGGAAATTGAGCCAGCCAGAATTGCATGTGTCCTCATTGTTGTGTTCCATTGCTGGTGGTGCTCATACCTGAGC
CAGAAAAGCAAAGATCTCCCCAGGACAACCAATGGCAATCATCATCATGGTAGCAGTAGGTCTTCTGGGCTTG
ATTACCGCCAATGAACTCGGATGGTTGGAGAGAACAAAGAGTGACCTAAGCCATCTAATGGGAAGGAGAGAGGA
GGGGGCAACCATAGGATTCTCAATGGACATTGACCTGCGGCCAGCCTCAGCTTGGGCCATCTATGCTGCCTTGA
CAACTTTCATTACCCCAGCCGTCCAACATGCAGTGACCACTTCATACAACAACACTCTTAATGGCGATGGCCA
CGCAAGCTGGAGTGTTGTTTGGTATGGGCAAAGGGATGCCATTCTACGCATGGGACTTTGGAGTCCCGCTGCTA
ATGATAGGTTGCTACTCACAATTAACACCCCTGACCCTAATAGTGCCATCATTTTCTCGTGGCGCACTACATG
TACTTGATCCCAGGGCTGCAGGCAGCAGCTGCGCGTGCTGCCAGAAGAGAACGGCAGCTGGCATCATGAAGA
ACCCTGTTGTGGATGGAATAGTGGTGACTGACATTGACACAATGACAATTGACCCCCAAGTGAGAAAAAGATG
GGACAGGTGCTACTCATAGCAGTAGCCGTCTCCAGCGCCATACTGTCGCGGACCGCCTGGGGGTGGGGGGAG
GCTGGGGCCCTGATCACAGCGGCAACTTCCACTTTGTGGGAAGGCTCTCCGAACAAGTACTGGAACCTCTAC
AGCCACTTCACTGTGTAACATTTTTAGGGGAAGTTACTTGGCTGGAGCTTCTAATCTACACAGTAACAAGAAA
CGCTGGCTTGGTCAAGAGACGTGGGGGTGGAACAGGAGAGACCTGGGAGAGAAATGGAAGGCCCGCTTGAA
CCAGATGTCGGCCCTGGAGTTCTACTCCTACAAAAAGTCAGGCATCACCGAGGTGTGCAGAGAAGAGGCCCGC
CGCGCCCTCAAGGACGGTGTGGCAACGGGAGGCCATGCTGTGTCCCGAGGAAGTGCAAAGCTGAGATGGTTG
GTGGAGCGGGGATACCTGCAGCCCTATGGAAGGTCATTGATCTTGGATGTGGCAGAGGGGGCTGGAGTTACT
ACGCCGCCACCATCCGCAAAGTTCAAGAAGTGAAAGGATACAAAAAGGAGGCCCTGGTCATGAAGAACCCATG
TTGGTGCAAAGCTATGGGTGGAACATAGTCCGTCTTAAGAGTGGGGTGACGTCTTTCATATGGCGGCTGAGCC
GTGTGACACGTTGCTGTGTGACATAGGTGAGTCATCATCTAGTCCTGAAGTGGAAGAAGCACGGACGCTCAGAG
TCCTCTCCATGGTGGGGGATTGGCTTGAAAAAGACCAGGAGCCTTTTGTATAAAAGTGTTGTGCCCATACACCA
GCACTATGATGGAACCCCTGGAGCGACTGCAGCGTAGGTATGGGGGAGGACTGGTCAGAGTGCCACTCTCCCCG
CAACTCTACACATGAGATGTACTGGGTCTCTGGAGCGAAAAGCAACACCATAAAAAAGTGTGTCCACCACGAGCC
AGCTCCTCTTGGGGCGCATGGACGGGCCAGGAGGCCAGTGAAATATGAGGAGGATGTGAATCTCGGCTCTGG
CACGCGGGCTGTGGTAAGCTGCGCTGAAGCTCCCAACATGAAGATCATTGGTAACCGCATTGAAAGGATCCGCA
GTGAGCACGCGGAAACGTGGTTCTTTGACGAGAACCACCCATATAGGACATGGGCTTACCATGGAAGCTATGAG
GCCCCACACAAGGGTCAGCGTCCTCTCTAATAAACGGGGTTGTCAGGCTCCTGTCAAACCCCTGGGATGTGGT
GACTGGAGTCACAGGAATAGCCATGACCGACACCACCCGTATGGTCAGCAAAGAGTTTTCAAGGAAAAAGTGG
ACACTAGGGTGCCAGACCCCCAAGAAGGCACTCGTCAGGTTATGAGCATGGTCTCTTCTGGTTGTGGAAAGAG
CTAGGCAAACACAAACGGCCACGAGTCTGTACCAAAGAAGAGTTCATCAACAAGGTTTCGTAGCAATGCAGCATT
AGGGGCAATATTTGAAGAGGAAAAAGAGTGGAAGACTGCAGTGGAAGCTGTGAACGATCCAAGGTTCTGGGCT
CTAGTGACAAGGAAAGAGAGCACCACCTGAGAGGAGAGTGCCAGAGTTGTGTGTACAACATGATGGGAAAAA
GAGAAAAAGAAACAAGGGGAATTTGGAAGGCCAAGGGCAGCCGCGCCATCTGGTATATGTGGCTAGGGGCTAG
ATTTCTAGAGTTGGAAGCCCTTGATTCTTGAACGAGGATCACTGGATGGGGAGAGAGAACTCAGGAGGTGGTG
TTGAAGGGCTGGGATTACAAAGACTCGGATATGTCCTAGAAGAGATGAGTCGCATACCAGGAGGAAGGATGTAT
GCAGATGACACTGCTGGCTGGGACACCCGCATCAGCAGGTTTGATCTGGAGAATGAAGCTCTAATCACCAACCA
AATGGAGAAAGGGCACAGGGCCTTGGCATTGGCCATAATCAAGTACACATACCAAAACAAAGTGGTAAAGGTCC
TTAGACCAGCTGAAAAAGGGAAGACAGTTATGGACATTATTTTCGAGACAAGACCAAAGGGGGAGCGGACAAGTT
GTCACCTACGCTCTTAACACATTTACCAACCTAGTGGTGCAACTCATTCCGAATATGGAGGCTGAGGAAGTTCTA
GAGATGCAAGACTTGTGGCTGCTGCGGAGGTGAGAGAAAGTGACCAACTGTTGCAGAGCAACGGATGGGATA
GGCTCAAACGAATGGCAGTCAGTGAGATGATTGCGTTGTGAAGCCAATTGATGATAGGTTTGCACATGCCCTC
AGGTTCTTGAATGATATGGGAAAAAGTTAGGAAGGACACACAAGAGTGGAACCCCTCAACTGGATGGGACAACCTG
GGAAGAAGTTCCGTTTTGCTCCCACCACTTCAACAAGCTCCATCTCAAGGACGGGAGGTCCATTGTGGTTCCCT
GCCGCCACCAAGATGAACTGATTGGCCGGGGCCGCTCTCTCAGGGGCGGGATGGAGCATCCGGGAGACTG
CTTGCCATAGCAAAATCATATGCGCAAATGTGGCAGCTCCTTTATTTCCACAGAAGGGACCTCCGACTGATGGCCA
ATGCCATTTGTTTCATCTGTGCCAGTTGACTGGGTTCCAAGTGGGAGAACTACCTGGTCAATCCATGGAAAGGGA
GAATGGATGACCACTGAAGACATGCTTGTGGTGTGGAACAGAGTGTGGATTGAGGAGAACGACCACATGGAAG
ACAAGACCCCAAGTTACGAAATGGACAGACATTCCCTATTTGGGAAAAAGGGAAGACTTGTGGTGTGGATCTCTC
ATAGGGCACAGACCGCGCACCCACCTGGGCTGAGAACATTAACAAACAGTCAACATGGTGCGCAGGATCATAG

```
GTGATGAAGAAAAGTACATGGACTACCTATCCACCCAAGTTCGCTACTTGGGTGAAGAAGGGTCTACACCTGGA
GTGCTG" # DNA coding sequence to be analyzed (coding sequence so its length should be a multiple of 3)
```

```
number_of_mutations = 1 # Number of mutations to consider
```

```
# Define a function that translates a DNA sequence to a protein sequence using the universal codon table
```

```
def translate_dna_sequence(dna_sequence):
```

```
    # The universal codon table
```

```
    table = {
```

```
        'ATA':'I', 'ATC':'I', 'ATT':'I', 'ATG':'M',
        'ACA':'T', 'ACC':'T', 'ACG':'T', 'ACT':'T',
        'AAC':'N', 'AAT':'N', 'AAA':'K', 'AAG':'K',
        'AGC':'S', 'AGT':'S', 'AGA':'R', 'AGG':'R',
        'CTA':'L', 'CTC':'L', 'CTG':'L', 'CTT':'L',
        'CCA':'P', 'CCC':'P', 'CCG':'P', 'CCT':'P',
        'CAC':'H', 'CAT':'H', 'CAA':'Q', 'CAG':'Q',
        'CGA':'R', 'CGC':'R', 'CGG':'R', 'CGT':'R',
        'GTA':'V', 'GTC':'V', 'GTG':'V', 'GTT':'V',
        'GCA':'A', 'GCC':'A', 'GCG':'A', 'GCT':'A',
        'GAC':'D', 'GAT':'D', 'GAA':'E', 'GAG':'E',
        'GGA':'G', 'GGC':'G', 'GGG':'G', 'GGT':'G',
        'TCA':'S', 'TCC':'S', 'TCG':'S', 'TCT':'S',
        'TTC':'F', 'TTT':'F', 'TTA':'L', 'TTG':'L',
        'TAC':'Y', 'TAT':'Y', 'TAA':'_', 'TAG':'_',
        'TGC':'C', 'TGT':'C', 'TGA':'_', 'TGG':'W',
```

```
    }
```

```
    protein_sequence = ""
```

```
    # Make sure the DNA sequence length is a multiple of three (ie it is an open reading frame)
```

```
    if len(dna_sequence)%3 == 0:
```

```
        # Pull out each codon in the DNA sequence
```

```
        for i in range(0, len(dna_sequence), 3):
```

```
            codon = dna_sequence[i:i + 3]
```

```
            # convert the codon to its corresponding amino acid using the above table
```

```
            protein_sequence += table[codon]
```

```
        return protein_sequence
```

```
# Define two functions that together output all possible DNA sequences of a given length
```

```
# For example, these will return 16 sequences if the length is 2 (AA, AT, AG, AC, TA, TT, TG, TC, GA, GT, GG, GC,
CA, CT, CG and CC)
```

```
def _all_sequences(bases, prefix, n, sequence_length):
```

```
    # If we have a sequence of the correct length then return this sequence
```

```
    if sequence_length == 0:
```

```
        possible_bases.append(prefix)
```

```
        return possible_bases
```

```
    # This loop generates the actual sequences
```

```
    for i in range(n):
```

```
        new_prefix = prefix + bases[i]
```

```
        _all_sequences(bases, new_prefix, n, sequence_length-1)
```

```

def all_sequences(sequence_length):
    # Define all possible bases
    bases = ["A", "T", "C", "G"]
    # Call the above function
    _all_sequences(bases, "", len(bases), sequence_length)

sequence_length = len(input_sequence)
marked_positions = []

# First, we need to generate all possible mutants of the input DNA sequence given the requested number of mutations
# We will do this in two steps

# In step 1 we will calculate all possible position(s) to mutate given the input DNA sequence and the number of mutations
# Then we'll mark the position(s) within the input sequence with an X and store the results in marked_positions
# For example: if the DNA sequence was ATGC and the number of mutations was 1 then we would store the following sequences in marked_positions:
# XTGC, AXGC, ATXC and ATGX
for positions in combinations(range(sequence_length), number_of_mutations):
    n = list(input_sequence)
    for position in positions:
        n[position] = "X"
    marked_positions.append(n)

# In step 2 we will substitute the marked_positions within each sequence with an A,T,C or G such that all possible combinations of sequence are generated.
# All combinations includes regenerating the input sequence
# As we don't want the input sequence, we will also remove all occurrences of the input sequence to generate the final set of mutant DNA sequences
possible_bases = []
all_sequences(number_of_mutations)

mutant_DNA_sequences = []
for position in marked_positions:
    for base in possible_bases:
        input_sequence_list = list(input_sequence)
        position_index = 0
        base_index = 0
        while position_index < len(position):
            if position[position_index] == "X":
                input_sequence_list[position_index] = base[base_index]
                base_index = base_index + 1
                position_index = position_index + 1
            mutant_DNA_sequences.append("".join(input_sequence_list))

```

```

# Remove any occurrence of the input sequence from our list of mutant sequences
set_of_mutant_DNA_sequences = set(mutant_DNA_sequences)
set_of_mutant_DNA_sequences.remove(input_sequence)
mutant_DNA_sequences = list(set_of_mutant_DNA_sequences)

# Next, we will translate each mutant DNA sequence into its corresponding protein sequence
proteins = []
for i in range(0, len(mutant_DNA_sequences)):
    proteins.append(translate_dna_sequence(mutant_DNA_sequences[i]))

# Now we can count how many of the proteins there are and how many of these proteins lack an internal stop codon
final_proteins = Counter(proteins)
stop = "_"
final_protein_count = 0
for protein in final_proteins:
    if not stop in protein:
        final_protein_count += 1

# Print all the output to screen
print()
print("Length of input DNA sequence: " + str(len(input_sequence)))
print()
print("Number of mutant DNA sequences: " + str(len(mutant_DNA_sequences)))
print()
print("Total number of protein sequences: " + str(len(final_proteins)))
print()
print("Number of protein sequences lacking internal STOP codons: " + str(final_protein_count))
print()

```

OUTPUT FROM THIS CODE:

Length of input DNA sequence: 10269

Number of mutant DNA sequences: 30807

Total number of protein sequences: 20921

Number of protein sequences lacking internal STOP codons: 19816