Python Code for Question 1

```
import sys

# Function to find all open reading frames (ORFs) in a given genome
sequence
def find_genes(sequence):
    """
    Find all open reading frames (ORFs) in the given genome sequence.
    Returns a list of found genes.
    """
    genes = []  # List to store found genes
    start_codon = 'ATG'  # Start codon
    stop_codons = ['TAA', 'TAG', 'TGA']  # Stop codons

    # Loop through the sequence to find start codons
    for i in range(len(sequence) - 2):
        if sequence[i:i+3] == start_codon:  # Check for start codon
            # Look for stop codon
            for j in range(i, len(sequence) - 2, 3):
                if sequence[j:j+3] in stop_codons:
                    genes.append(sequence[i:j+3])  # Add gene to list
                    break  # Exit the loop after finding the first stop
codon
    return genes  # Return the list of found genes

# Main function to handle command line arguments and read input file
def main():

    input_file = sys.argv[1]  # Get the input file from command line
arguments

    try:
        codonMin = sys.argv[2]

    except:
        CodonMin = 20

    with open(input_file, 'r') as file:  # Open the input FASTA file
        sequence = ''  # Initialize an empty string for the genome
sequence
        for line in file:  # Read the file line by line
            if not line.startswith('>'):  # Ignore header lines
```

```
                sequence += line.strip()  # Append sequence lines to
the genome sequence

    genes = find_genes(sequence)  # Call the function to find genes
    print("Found genes:", genes)  # Output the found genes
    print(len(genes))

# Entry point of the script
if __name__ == "__main__":
    main()  # Run the main function
```

Python Code for Question 2

```
import sys

# Function to find all open reading frames (ORFs) in a given genome
sequence
def find_genes(sequence):
    """
    Find all open reading frames (ORFs) in the given genome sequence.
    Returns a list of found genes.
    """
    genes = []  # List to store found genes
    start_codon = 'ATG'  # Start codon
    stop_codons = ['TAA', 'TAG', 'TGA']  # Stop codons

    # Loop through the sequence to find start codons
    for i in range(len(sequence) - 2):
        if sequence[i:i+3] == start_codon:  # Check for start codon
            # Look for stop codon
            for j in range(i, len(sequence) - 2, 3):
                if sequence[j:j+3] in stop_codons:
                    genes.append(sequence[i:j+3])  # Add gene to list
                    break  # Exit the loop after finding the first stop
codon
    return genes  # Return the list of found genes
```

```
# Function to compute reverse complement of a DNA sequence
def reverse_complement(sequence):
    """
    Return the reverse complement of the given DNA sequence.
    Skip any ambiguous bases.
    """
    complement = {'A': 'T', 'T': 'A', 'C': 'G', 'G': 'C'}  #
Complementary base pairs
    reverse_seq = sequence[::-1]  # Reverse the sequence
    return ''.join([complement.get(base, 'N') for base in reverse_seq])
# Replace with 'N' if no complement found

# Function to find genes in all three reading frames
def find_genes_in_frames(sequence):
    """
    Find genes in all three reading frames of a given sequence.
    """
    all_genes = []
    for frame in range(3):  # Check the three reading frames
        all_genes.extend(find_genes(sequence[frame:]))  # Find genes in
the current reading frame
    return all_genes

# Main function to handle command line arguments and read input file
def main():
    input_file = sys.argv[1]  # Get the input file from command line
arguments

    try:
        codonMin = int(sys.argv[2])
    except:
        codonMin = 20

    # Read the genome sequence from the input FASTA file
    with open(input_file, 'r') as file:
        sequence = ''  # Initialize an empty string for the genome
sequence
        for line in file:
            if not line.startswith('>'):  # Ignore header lines
                sequence += line.strip()  # Append sequence lines to
the genome sequence
```

```
    # Find genes in both forward and reverse complement
    forward_genes = find_genes_in_frames(sequence)
    reverse_genes = find_genes_in_frames(reverse_complement(sequence))

    # Combine genes from both forward and reverse strands
    all_genes = forward_genes + reverse_genes

    print("Found genes:", all_genes)  # Output the found genes
    print(len(all_genes))


# Entry point of the script
if __name__ == "__main__":
    main()  # Run the main function
```

Python Code for Question 3 and 4

```
import sys

# Genetic code dictionary for translating codons into amino acids
genetic_code = {
    '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',
}
```

```
# Function to translate a sequence of codons into a protein sequence
def translate_sequence(seq):
    protein = []
    for i in range(0, len(seq), 3):
        codon = seq[i:i+3]
        if codon in genetic_code:
            amino_acid = genetic_code[codon]
            if amino_acid == '_':  # Stop translation if a stop codon
is found
                break
            protein.append(amino_acid)
    return ''.join(protein)

# Function to find ORFs in all six reading frames and translate them
def find_orfs(sequence):
    orfs = set()  # Use a set to store distinct protein sequences
    frames = [sequence, sequence[1:], sequence[2:]]  # Forward frames

    # Process each forward reading frame
    for frame in frames:
        orfs.update(find_orfs_in_frame(frame))

    # Process the reverse complement frames
    reverse_sequence = reverse_complement(sequence)
    reverse_frames = [reverse_sequence, reverse_sequence[1:],
reverse_sequence[2:]]
    for frame in reverse_frames:
        orfs.update(find_orfs_in_frame(frame))

    return orfs

# Function to find ORFs in a single reading frame
def find_orfs_in_frame(frame):
    start_codon = 'ATG'
    stop_codons = ['TAA', 'TAG', 'TGA']
    orfs = set()

    for i in range(len(frame) - 2):
        if frame[i:i+3] == start_codon:
            for j in range(i, len(frame) - 2, 3):
                codon = frame[j:j+3]
                if codon in stop_codons:
                    # Translate the ORF and add it to the set
```

```
                    protein = translate_sequence(frame[i:j+3])
                    if protein:
                        orfs.add(protein)
                    break
    return orfs

# Function to generate the reverse complement of a DNA sequence
def reverse_complement(sequence):
    complement = {'A': 'T', 'T': 'A', 'C': 'G', 'G': 'C'}
    reverse_seq = sequence[::-1]  # Reverse the sequence
    return ''.join([complement[base] for base in reverse_seq if base in
complement])

# Main function to handle command line arguments and read input file
def main():
    input_file = sys.argv[1]  # Get the input file from command line
arguments

    with open(input_file, 'r') as file:  # Open the input FASTA file
        sequence = ''  # Initialize an empty string for the genome
sequence
        for line in file:
            if not line.startswith('>'):  # Ignore header lines
                sequence += line.strip()

    # Find and print all distinct ORFs (translated proteins)
    orfs = find_orfs(sequence)
    for protein in orfs:
        print(protein)

# Entry point of the script
if __name__ == "__main__":
    main()  # Run the main function
```

Python Code for Question 5

```
import sys

# Genetic code dictionary for translating DNA sequences to protein
genetic_code = {
    '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':'_', 'TGA':'_'
}

# Function to translate a DNA sequence to a protein
def translate_sequence(seq):
    protein = []
    for i in range(0, len(seq), 3):
        codon = seq[i:i + 3]
        if len(codon) < 3:
            break
        protein.append(genetic_code.get(codon, 'X'))  # 'X' for unknown
codon
        if protein[-1] == '_':  # Stop codon
            break
    return ''.join(protein)

# Function to find ORFs longer than the specified minimum length
def find_orfs(dna_sequence, min_length=100):
    orfs = []
    stop_codons = ['TAA', 'TAG', 'TGA']

    # Search in all 3 frames
    for frame in range(3):
        for i in range(frame, len(dna_sequence) - 2, 3):
            codon = dna_sequence[i:i + 3]
```

```
            if codon == 'ATG':  # Start codon
                for j in range(i, len(dna_sequence) - 2, 3):
                    stop_codon = dna_sequence[j:j + 3]
                    if stop_codon in stop_codons:
                        orf_length = (j - i + 3) // 3  # Calculate
length in codons
                        if orf_length >= min_length:

orfs.append(translate_sequence(dna_sequence[i:j + 3]))
                        break
    return orfs

# Function to find ORFs from both strands
def find_orfs_in_both_strands(dna_sequence, min_length=100):
    # Complementary strand
    complement = {'A':'T', 'T':'A', 'C':'G', 'G':'C'}
    reverse_complement = ''.join([complement[base] for base in
dna_sequence[::-1]])

    # Find ORFs in both the original and reverse complement strands
    orfs = find_orfs(dna_sequence, min_length) +
find_orfs(reverse_complement, min_length)
    return orfs

# Main function to load DNA from a file and find ORFs
def main():
    if len(sys.argv) < 2:
        print("Usage: python gene_finder.py <input_file> [min_length]")
        sys.exit(1)

    input_file = sys.argv[1]
    min_length = int(sys.argv[2]) if len(sys.argv) > 2 else 100  #
Default to 100 codons if not specified

    with open(input_file, 'r') as f:
        dna_sequence = ''.join([line.strip() for line in f if not
line.startswith('>')])  # Ignore header lines

    orfs = find_orfs_in_both_strands(dna_sequence, min_length)

    for orf in orfs:
        print(orf)
```

```
if __name__ == "__main__":
    main()
```

Python Code for Question 6

```
import sys

# Default RBS sequence and search parameters
RBS_SEQUENCE = "AGGAGG"
MIN_ORF_LENGTH = 100  # minimum ORF length in codons
UPSTREAM_WINDOW = 20   # window to search upstream of the start codon

def has_rbs(sequence, start_index, window=UPSTREAM_WINDOW,
rbs_seq=RBS_SEQUENCE):
    """
    Check if the upstream region contains a ribosome binding site
(RBS).

    :param sequence: full genome sequence
    :param start_index: index of the start codon
    :param window: number of bases to look upstream of the start codon
    :param rbs_seq: sequence of the ribosome binding site to search for
    :return: True if an RBS is found, False otherwise
    """
    # Ensure we don't go out of bounds
    upstream_start = max(0, start_index - window)
    upstream_region = sequence[upstream_start:start_index]

    # Check if the RBS sequence is in the upstream region
    return rbs_seq in upstream_region

def filter_orfs_by_rbs(orfs, genome_sequence):
    """
    Filter ORFs based on the presence of a Shine-Dalgarno sequence
upstream.

    :param orfs: list of ORFs (start, end positions)
    :param genome_sequence: full genome sequence
    :return: list of ORFs with valid RBS
```

```
    """
    valid_orfs = []
    for orf in orfs:
        start, end = orf
        if has_rbs(genome_sequence, start):
            valid_orfs.append(orf)
    return valid_orfs

def find_orfs(genome_sequence):
    # Your logic for finding ORFs goes here
    # e.g., finding start codons (ATG) and stop codons (TAA, TAG, TGA)
    orfs = []
    # Example ORFs found (start, end positions)
    orfs.append((100, 600))  # Just an example
    return orfs

def main():
    genome_file = sys.argv[1]
    with open(genome_file, 'r') as f:
        genome_sequence = f.read()

    # Find ORFs
    orfs = find_orfs(genome_sequence)

    # Filter ORFs by length (default length is 100 codons, can be
parameterized)
    orfs = [orf for orf in orfs if (orf[1] - orf[0]) / 3 >=
MIN_ORF_LENGTH]

    # Filter ORFs based on the presence of an RBS
    valid_orfs = filter_orfs_by_rbs(orfs, genome_sequence)

    # Output the valid ORFs (could be printed or written to a file)
    for orf in valid_orfs:
        print(f"Valid ORF: Start: {orf[0]}, End: {orf[1]}")

if __name__ == '__main__':
    main()
```