Cairo University

Faculty of Engineering -
Cairo UniversityCredit Hours
System HEM

2024

Spring 24 SBES375
Bioinformatics I

Assignment - 3

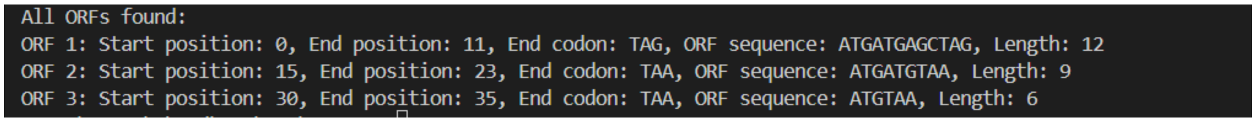
Prepared by:

Ahmed Gehad Mohamed Aly (ID:1200387)

1. Open Reading Frames Detection:

Create a function that finds all the ORFs in a given DNA sequence. Return a list of ORFs along with their start and end positions.

Output:



```
All ORFs found:
ORF 1: Start position: 0, End position: 11, End codon: TAG, ORF sequence: ATGATGAGCTAG, Length: 12
ORF 2: Start position: 15, End position: 23, End codon: TAA, ORF sequence: ATGATGTAA, Length: 9
ORF 3: Start position: 30, End position: 35, End codon: TAA, ORF sequence: ATGTAA, Length: 6
```

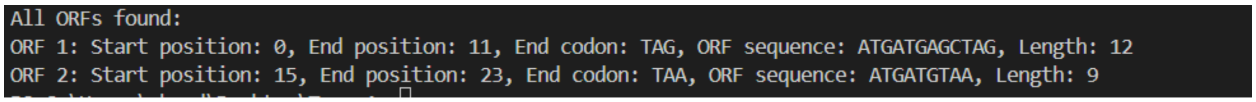
Code_1 Explanation:

- We Define start codon (ATG) and stop codons (TAA, TAG, TGA)
- Initialize an empty list to store ORFs
- Convert DNA sequence to a Seq object
- iterates through the DNA sequence to find ORFs. It starts by initializing a loop variable i to 0.
- We first look for the start condon by using this line of code: start_pos = sequence.find(start_codon, i) break if no start codon is found in the whole sequence
- If we find a start codon we do another for loop to find stop codon
- Here we Extract the ORF sequence in this line of code: orf_sequence = sequence[start_pos:end_pos + 1]
- Note we search by each 3 letters after one another we are not search letter by letter for the codon sequence we searching by each 3 letters.
- Store ORF information: Information about the ORF, including its start position, end position, stop codon, sequence appended to the all_orfs list in this line of code: all_orfs.append((start_pos, end_pos, codon, orf_sequence, orf_length))
- Function returns the list of all ORFs: return all_orfs
- At the end I gave an example sequence I made to use as a test example for the function and printed the output of the ORF and their index values

Modify the function to filter out ORFs that are too short (less than a specified length).

I chose Specified length filtered out to be (sequence ORF<=6)

Output



```
All ORFs found:
ORF 1: Start position: 0, End position: 11, End codon: TAG, ORF sequence: ATGATGAGCTAG, Length: 12
ORF 2: Start position: 15, End position: 23, End codon: TAA, ORF sequence: ATGATGTAA, Length: 9
```

Code_2 Explanation:

Just added to code 1 the following:

- Calculates the length of each sequence orf_length = len(orf_sequence)
- Added an if condition to check if the length of the sequence is less than 6 or not if less than 6, then we will not add this ORF sequence to our all_orfs list:

```
if orf_length > 6:
   all_orfs.append((start_pos, end_pos, codon, orf_sequence, orf_length
    i = end_pos + 1
    break
```

2. What is the difference between FASTA and FASTQ?

FASTA Format:

FASTA is a straightforward file format used for representing biological sequence data. It consists of two main components:

Sequence ID: Begins with the ">" symbol, followed by a unique identifier for the sequence.

Sequence Data: Represents the actual sequence of nucleotides (A, T, G, C) or amino acids.

FASTA files are commonly used for storing sequences in databases, sharing sequence information, and conducting basic sequence-based analyses.

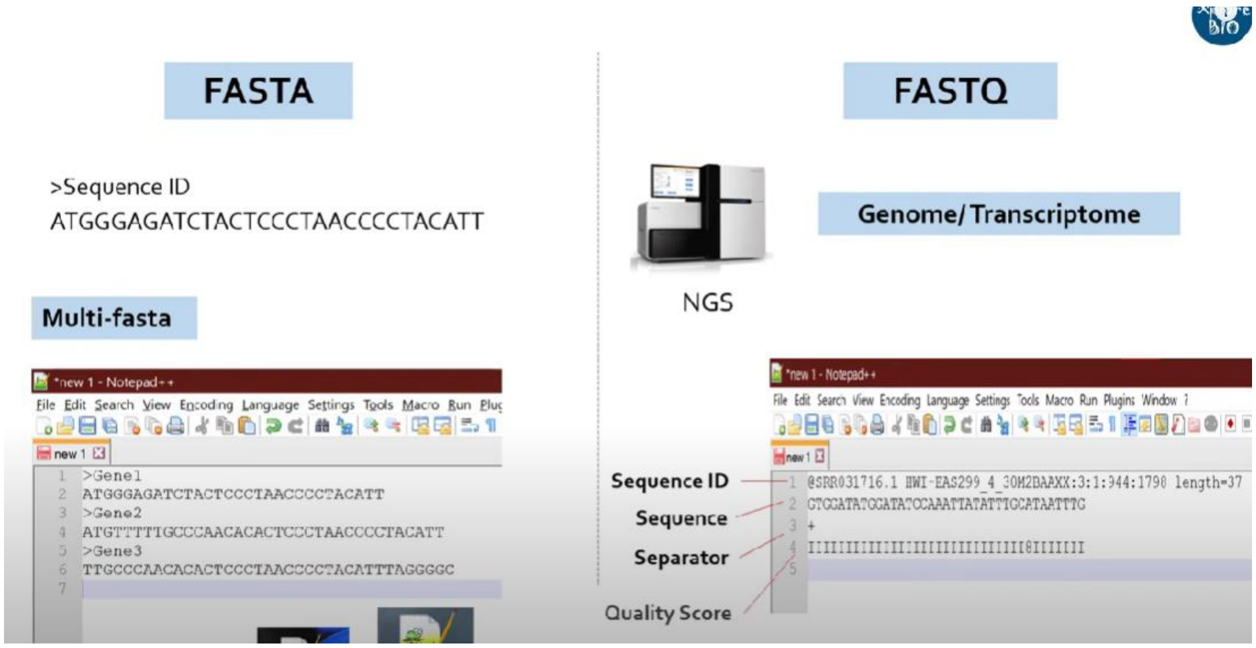
FASTQ Format:

FASTQ is an enhanced file format that includes quality information along with sequence data. It comprises four lines per sequence record:

Sequence ID: Starts with the "@" symbol, followed by the identifier.

Sequence Data: Represents the sequence of nucleotides or amino acids.

Separator Line: Denoted by the "+" symbol.



```
>Sequence ID
ATGGGAGATCTACTCCCTAACCCCTACATT
```

```
>Gene1
ATGGGAGATCTACTCCCTAACCCCTACATT
>Gene2
ATGTTTTTGCCCAACACACTCCCTAACCCCTACATT
>Gene3
TTGCCCAACACACTCCCTAACCCCTACATTTAGGGGC
```

```
@SRR031716.1 HWI-EAS299_4_30M2BAAXX:3:1:944:1798 length=37
GTGGATATGGATATCCAAATTATATTTGCATAATTTG
+
IIIIIIIIIIIIIIIIIIIIIIIIIIIII8IIIIIII
```

Quality Scores: Represented using ASCII characters or Phred scores, indicating the quality of each base in the sequence.

FASTQ files are primarily generated by next-generation sequencers and are essential for assessing the reliability of sequencing data. Quality scores help in identifying and filtering out low-quality bases before downstream analysis.

Importance of Quality Information:

In next-generation sequencing technologies, such as Illumina sequencing by synthesis, fluorescent-labeled nucleotides are added to the growing DNA strand and captured by a sensitive camera. The process of interpreting the fluorescent signals to determine the sequence, generates base quality information. This information is crucial for:

- Identifying sequencing errors and artifacts.
- Filtering out low-quality bases to improve the accuracy of downstream analyses, such as genome assembly and mapping.

3. What should be used instead of Seq object so it can be modified?

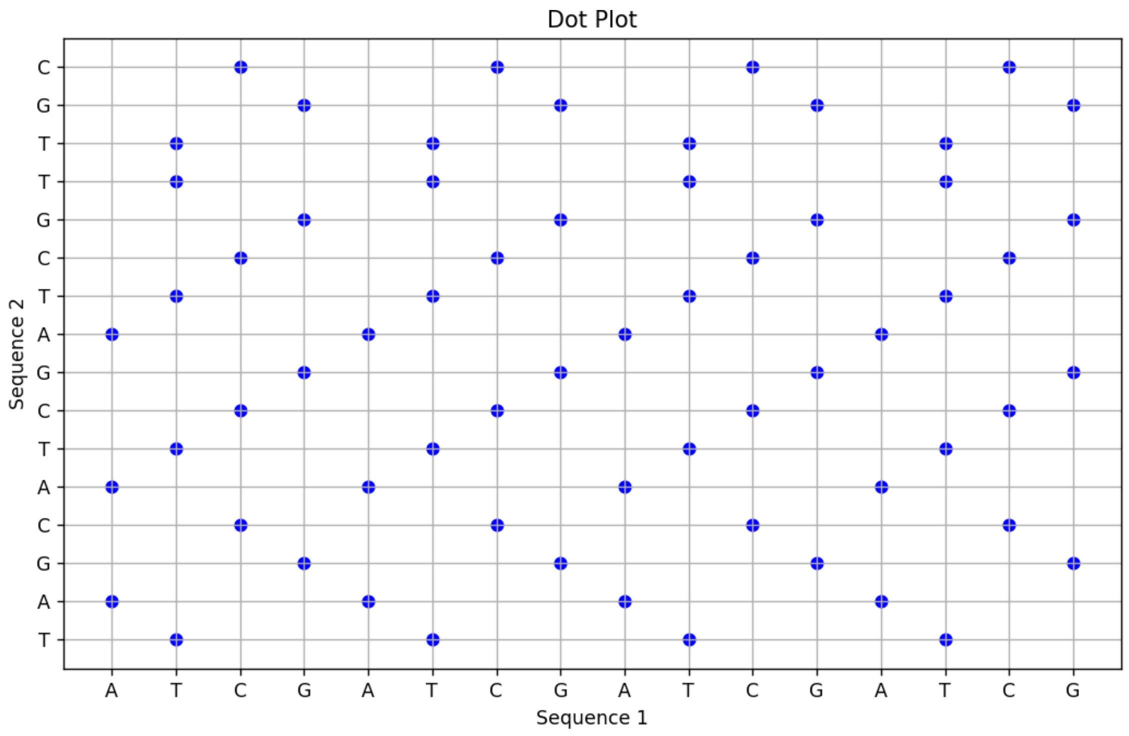
By representing the sequence as a string, you can easily modify and manipulate it.

4. Create a Python program that generates a dot plot to visualize the

Code Explanation:

- The function takes 2 Dna sequences as an input
- Creates a new graph of size (10,6) using matplotlib
- The x axes will be sequence 1 and the y will be our 2 sequence. Check the function inputs
- In the for loops we Iterate over the positions in both sequences: Nested loops iterate over each position in both sequence1 and sequence2.
- Plot a blue dot for matching nucleotides see the following code under
  if sequence1[i] == sequence2[j]:
    plt.scatter(i, j, color='blue')
- Now we add the sequence letters on the x-axis and y-axis for easier visualisation:
  plt.xticks(range(len(sequence1)), sequence1)
  plt.yticks(range(len(sequence2)), sequence2)
- Then to show the graph on matplotlib we use the plt.show() function
- At the end I added a random example of 2 sequence to put in my function to generate a visualised output

Out put:



References

https://www.youtube.com/watch?v=cJm_BGpjnWg

https://vlab.amrita.edu/?sub=3&brch=273&sim=1432&cnt=1

https://www.genome.gov/genetics-glossary/Open-Reading-Frame

https://biopython.org/wiki/Seq