IQB ASSIGNMENT 1

Team Members:-

Amandeep Kaur (2018014)
Deepi Garg (2018389)
Ria Gupta (2018405)

Question1)

Python script: iqb1.py
Input: FASTA format file (Default: DNA.fa.txt)
Output: 2 FASTA format files (Default: OutRNA.fa, OutProtein.fa)

Command to run the script:-
To use default input and output files:
$ python3 iqb1.py
To provide your own input file:
$ python3 iqb1.py -i [**filename**]
To provide your own input and output files:
$ python3 iqb1.py -i [**input_file**] **-o** [**output_file1**] [**output_file2**]

The python script converts an input 5'-3' DNA sequence to a 5'-3' RNA sequence and stores it in a *FASTA* file.(Transcription)
It also translates the RNA sequence to the corresponding Protein sequence using the first reading frame. It starts translating when it encounters the START codon (ATG) and translates the sequence till the first STOP codon.

Question2)

Python script: iqb2.py
Input: PDB format file (Default: 3v44.pdb)
Output: 1 PlainText file (Default: output.txt)

Command to run the script:-
To use default input and output files:
$ python3 iqb2.py
To provide your own input file:
$ python3 iqb2.py -i [**filename**]
To provide your own input and output files:
$ python3 iqb2.py -i [**input_file**] **-o** [**output_file**]

Step1: Opening the pdb file-
We use *getopt.getopt* to parse options and arguments from command line. If the command returns an error, we print the correct format of command for file execution and exit, else we read the options provided. If the option is '*-i*', we assign the following argument to the input file, if option is '*-o*' we assign the argument to the output file.

Step2: Reading the pdb file-
We open the input and output file under the name input_file and output_file(in write mode) respectively. For each line in input_file, we split the line by spaces and create a list. Now if the first word is *'HEADER' (or 'TITLE')*= we save that line in header(or title), print it to the terminal and write to the output file. If the length of this list of words is greater than 2, and 3rd word in the line is '*RESOLUTION*', then we save the line from index 12, till end to resolution, print and write the same to output file.

Question3)

Python script: iqb3.py

Input: FASTA format file (Default: protein.fa)
Output: 2 csv format files (Default: dotplot_output.csv, sum_matrix_output.csv)

Requirements:-
One needs to install **pandas** library for python3 before using the code.

Command to run the script:-
To use default input and output files:
$ python3 iqb3.py
To provide your own input file:
$ python3 iqb3.py -i [**filename**]

Step 0: Reading the original sequences-
The *getopt.getopt* method parses command line options and parameter list. We try to find running options and arguments and save them in *opts, args*. If the command returns an error, we output the correct format of the command and exit. Else, the option is read, if option is *-i*, that is input file follows, we assign the arg to input file.
We open the input file, for each line in file, if the line starts with '>' symbol and is not a blank line, we add that line to the list *seq[]. seq[0]* and *seq[1]* gives the original sequences to be aligned. Actually, we are converting a sequence stored in the FASTA file to string data type.

Step1: Creating the dotplot-
We create a matrix named *dotplot* of dimensions (length of sequence 1 * length of sequence 2) and initialize all cells to zero.
Now we iterate over each cell, and check corresponding elements of the sequence strings. If both the elements are same (match), we place 1, else (mismatch) we place a 0 in that cell .
Converting the *dotplot* into dataframe and loading to csv file-
We create two lists, named *colnames* having elements of sequence string 1 at each index, similarly *rownames*, having elements of sequence string 2.

Then we create a dataframe *df* with elements of *rownames* as index and elements of *colnames* as columns and load the 1,0 values from dotplot created earlier.
To put the dataframe into a csv file named *dotplot_output.csv*, we use the *to_csv* command using commas as separators.

Step2: Making a sum matrix and loading it into a csv file-
We make a copy of dotplot named *sum_matrix* using *deepcopy*(so that changes in *sum_matrix* are not reflected in *dotplot*), next we fill in the *sum_matrix.* As the last column and last row are always same as that of *dotplot*, we start out iteration from index len(sequence)-2 to index (-1) reducing index by 1 at every step.
For each cell, we follow the following algorithm:

new_value(row r, column c)= value(r,c) + max{value(r+1,c+1),cells(r+1,c+2 to c_max),cells(r+2 to r_max, c+2)}

We create a dataframe *df_sum* and load it into a csv file named *sum_matrix_output* as we did with dotplot.

Step3: Backtracking to find the required alignment.
We start from the top left corner, it has the greatest value in the complete matrix. Now we move diagonally downwards to the next maximum value. If, to reach next maximum value, we proceed downwards in a column, we introduce a gap in the sequence on the top. If in a column, all values till the end of column are 0, we introduce a gap in the sequence on the left, else, append the corresponding element of the top and left sequences into *first_seq_allign* and *sec_seq_allign*. Following the above conditions, reach the end of the matrix.
The final alignments using Needleman Wunsh algorithm (with 0 gap penalty) are given by first_seq_allign and sec_seq_allign.

The final outcome (optimal alignments) are dispayed on the terminal, along with a dotplot and matrix of sums.