CSC 314**, Exam II Review**

**Outline**

1. Bioinformatics Databases
   1. GenBank – searching and understanding sequence records
   2. GenPept (Protein) – searching and understanding sequence records
   3. UCSC Genome Browser and Table Browser
2. Sequence alignments
   1. Calculating alignment scores using BLOSUM-62 matrix and gap penalties
   2. BLOSUM vs PAM matrices
   3. Dynamic programming
      1. Needleman-Wunsch global alignment method
      2. Smith-Watterman local alignment method
3. Biopython
   1. Working with sequence (Seq) objects
   2. Working with SeqRecord objects (GenBank or FASTA format)
      1. Accessing feature types, locations, and qualifiers
      2. Extracting a sequence (a region)
4. Additional Python concepts
   1. List comprehension
   2. Regular expressions

**Note:** This review is not comprehensive, but contains several practice problems to help prepare for Exam II. In addition to these practice problems, you should make sure to understand all labs and course notes since the first exam.

1. ***Sequence Database questions***

Look at the GenBank entry with accession number NM\_001185098.1 to answer the questions below:

* 1. What is the length of this sequence?
  2. When was the sequence last modified?
  3. How many exons does this gene have?
  4. What position marks the beginning and end of the poly-adenylation signal sequence, and what is this sequence?
  5. What are the first nine nucleotides in the coding sequence (CDS), and the corresponding amino acids (you can give the 1-letter amino acid codes)?

1. ***UCSC Genome / Table Browser***

You can view SARS-Cov2, the virus which causes COVID-19, here: <https://genome.ucsc.edu/cgi-bin/hgTracks?db=wuhCor1&lastVirtModeType=default&lastVirtModeExtraState=&virtModeType=default&virtMode=0&nonVirtPosition=&position=NC_045512v2%3A1%2D29903&hgsid=944856447_taX3FbgH6PMvp6v3qdJkMejgfZHI>

* 1. Hide all tracks and then view the following tracks only, using Pack view:
     1. NCBI genes
     2. Uniprot Protein Products

Note: SARS-Cov2 has 2 genes (Orf1a and Orf1ab) that each code for multiple proteins.

Hover over the protein *Pol* and indicate the function of this protein.

1. ***Sequence Alignments***

For (a) and (b), use a linear gap penalty of 4 points, a match score of +5 points, and a mismatch score of -1 point.

* 1. Find the optimal global alignment and optimal global alignment score for the words *handy* and *say.* You must show your dynamic programming matrix to receive credit.
  2. Find the optimal local alignment and optimal local alignment score between the words *stars* and *that*. You must show your dynamic programming matrix to receive credit.
  3. Using the BLOSUM-62 matrix, a gap opening penalty of 5, and a gap extension penalty of 1, find the score of the *semiglobal* alignment given below (Recall that semiglobal alignments do not penalize gaps at the beginning or end of the alignment).

F R I D A - - Y

- - P – A R T Y

1. ***Coding questions***

***Coding question #1***

The regular expression corresponding to a potential coding sequence (CDS) is given by (without the quotes): "ATG(?:.{3})\*?(?:TAG|TAA|TGA)"

Note: it is not necessary to understand the regular expression (this is much more

advanced than the examples covered in class). For an explanation, put the regular

expression into <https://regex101.com/> and look at the description below:

* ATG – the start codon, ATG
* (?:.{3})\*?- any number of codons (0 or more) (will match as few times as possible, i.e., *non-greedy*)
* (?:TAG|TAA|TGA)– matches any of the stop codons

Technical note #1: Normally parentheses denote a *capturing group*, and the match to this pattern is returned; to prevent this, we use (?:) which makes the group *non-capturing*, so that the pattern matching the entire regular expression is returned.

Technical note #2: the asterisk (\*) will match the preceding pattern as many times as possible, which is known as *greedy* evaluation. To match a pattern as *few* times as possible, follow the asterisk with a question mark (\*?), which is known as *lazy* evaluation.

Technical note #3: the regular expression above will not find a potential CDS if it is within a larger one. For example, ATGATGTGA contains two potential CDSs (ATGATGTGA and ATGTGA), but only the larger one will be returned. However, we could use a *positive lookahead* (?=) to handle these cases as well.

**Question:** Write Python code to prompt the user to enter a DNA sequence. The code

then uses the regular expression above to find each potential CDS. If no CDS is

found, the code outputs “Your sequence does not contain a potential CDS”

***Coding question #2***

Suppose that the header of a FASTA sequence is of the form: *Human\_GeneName* (e.g., *Human\_TP53*). If the object *ids* contains a list of headers, use list comprehension to create a list that contains only the gene names.