Basic Concepts of Biology

Registration for exercise slots: Meanwhile the standard exercise group in TUMonline might already have been deleted. Please register to one of the Tuesday or Thursday spots offered instead. For the in-class programming example we provide a master solution.

1.1 (E) Genetic Code

The aim of this exercise is to become familiar with the genetic code. To do so, you are asked to write a small program in Python:

Restriction: Do not use functions from the package BioPython or similar ones.

Your program should:

- implement the genetic code given in [https://en.wikipedia.org/wiki/Genetic_code#Standard_codon_tables](https://en.wikipedia.org/wiki/Genetic_code#Standard_codon_tables) as a dictionary of DNA triplets to a one letter amino acid code symbol. Remember that DNA in contrast to RNA contain T instead of U
- write a function called `codons_to_aa` that takes a DNA sequence coding for an open reading frame (ORF). An ORF is a DNA/RNA sequence that precisely codes for a protein, i. e. no leading or trailing untranslated DNA/RNA residues.
- the function `codons_to_aa` returns the corresponding amino acid sequence as a string in one letter code

1.2 (E) Amino Acid Distribution

The aim of this exercise is to become familiar with the genetic code. To do so, you are asked to write a small function in Python:

Restriction: Do not use functions from the package BioPython or similar ones.

Your function `aa_dist` should:

- take an amino acid sequence in one letter code
• return the amino acid distribution as Python dictionary with the single letter codes as keys and the frequencies as values. You might want to check out the Counter class.

1.3 (E) FASTA sequence format
The goal of this exercise is to introduce the FASTA sequence format. To do so, you are asked to write a small function in Python:

Restriction: Do not use functions from the package BioPython or similar ones.

Your function read_fasta should:
• take a filename as argument
• open and read out one sequence given in FASTA format
• return a tuple consisting of the header and the sequence body

1.4 (H) Genetic Code
You already wrote a program to translate a single ORF to an amino acid sequence. Now we will extend this:

Restriction: Do not use functions from the package BioPython or similar ones.

Write a program called orffinder.py which does the following:

(a) write a function get_orfs that takes a DNA sequence as a single string parameter
(b) if the input sequence is not a DNA sequence it raises a TypeError exception
(c) find all ORFs encoding proteins longer than 33, i.e. the resulting polypeptide has at least 34 amino acids, in all six reading frames (primary and reverse-complementary strand)
(d) the return value is a list of ORFs represented as 4-tuples containing (in this order) the position of the first DNA residue, the position of the last DNA residue (including stop codon), the translated amino acid sequence as a single string, and a flag which is True if the ORF is parsed from the reversed strand. Positions start at index 0 (first nucleotide in the primary strand) and ORFs in the reverse-complementary strand must be indexed according to their position on the primary strand (i.e. start > stop except if they are circular)
Example: (591, 704, 'MNFAKLMVRICHMRILTMKKLMNGILIFICLHFWMIG', False)
(e) if multiple ORFs overlap and end with the same stop codon return only the longest
(f) mind circular DNA, e.g. plasmoids
(g) the DNA index always has to be non-negative and less than the genome length.

1.5 (H) Amino Acid Distribution
Restriction: Do not use functions from the package BioPython or similar ones.

In this exercise we want extend your knowledge of the amino acid distribution in a bacterial genome. Write a program called aa_dist.py which provides functions called:
(a) `read_fasta(filename)` which takes a filename as parameter and reads in a file with multiple protein sequences in FASTA format

(b) `get_counts()` which counts the number of read sequences and returns the number as integer

(c) `get_average_length()` which calculates the average sequence length and returns the result as float

(d) `get_abs_frequencies()` which counts the occurrence for every amino acid over all proteins

(e) `get_av_frequencies()` which calculates the average amino acid composition over all read protein sequences

(f) the absolute amino acid frequencies are a dictionary with the single letter amino acid symbol as key and an integer value whereas the average amino acid composition has the same format but a float value.

(g) introduce counter variables and collections where they are needed

1.6 (H) Amino acid properties

The goal of this exercise is to introduce the different physicochemical properties of amino acids. To do so, you are asked to write several small functions in Python. Put them in a file called `aa_props.py`:

**Restriction:** Do not use functions from the package BioPython or similar ones.

Use the [https://en.wikipedia.org/wiki/Amino_acid#General_structure](https://en.wikipedia.org/wiki/Amino_acid#General_structure) as reference and remember that an amino acid can be in multiple categories.

All functions take a single letter amino acid code symbol and return True or False depending if they have the respective property. Implement the following functions:

- isCharged
- isPositivelyCharged
- isNegativelyCharged
- isHydrophobic
- isAromatic
- isPolar
- isProline
- containsSulfur
- isAcid
- isBasic