Biomedical Software Engineering

Mount Sinai School of Medicine
Graduate School of Biomedical Sciences
Prof. Arthur Goldberg
Spring II, 2024

Assignment 2: Write a Python program that uses the genetic code

Overview

This assignment's goals are to get you started writing Python in our course, ask you to write code that uses some of the Python features in our Python tutorial, and encourage you to start using the documentation in The Python Tutorial and The Python Standard Library.

Logistics

I welcome emailed questions about this and any other assignment. I want all students in our course to receive equal information at the same time as much as possible. To achieve this I will respond to questions by removing the name of the student who sent the question and provide the question and answer to the full class. However, if the question contains personal or sensitive information, I will not share that with the rest of the class.

All BMSE course policies apply to this assignment, including the late work policy, the plagiarism policy, and the collaboration policy. I welcome any questions about these course policies.

Hand in your assignment by emailing your Python program to me by noon on Thu, April 25. Please include your first and last names in your program's filename.

Your program

Use the genetic code

The genetic code controls the relationship between DNA codons, transcription and translation. Build a program called `trans_trans`, for transcription and translation, that inputs some DNA sequences (I'll call them mini-chromosomes) and outputs the proteins that they encode and some summary statistics.

Your program should have this structure and incorporate these features:

1. Use the "`#!/usr/bin/env python`" line so your program can be run from the command line. Please use Python 3.
2. Input and process a list of mini-chromosomes, each of which is a string composed of A, C, G, and T. I provide some example mini-chromosomes in the **Example data** section below. You can hard-code them into your program as input, for example as a list literal, or, better yet, input them from a file.
3. Hard-code a dictionary literal that maps all possible codons into their role in the genetic code in your program. I recommend that you map each codon into one of the 1 letter amino acid abbreviations or 'STOP'. Treat ATG as a special case because it

encodes for both START and methionine, and therefore cannot be fully represented in this codon dictionary.

4. Compute the values of some properties of the DNA. These include:
    a. the number of encoded proteins
    b. the length in amino acids of the shortest, and longest protein
    c. the mean protein length in amino acids
    d. the number of DNA failures (defined below)
    e. and the total amount of non-coding DNA in nucleotides
5. Iterate over the mini-chromosomes.
6. Examine each mini-chromosome and use the codon dictionary to determine and print the amino acid sequence of each encoded protein. A protein is encoded by any sequence of codons on the DNA that immediately follows a START codon and immediately precedes a STOP codon. Biologically, your program will combine transcription and translation into one step.
    a. The scans should maintain the properties of the DNA described above.
    b. Coding DNA is any DNA that encodes for a protein, plus the START and STOP codons that enclose it. All other DNA is non-coding.
    c. A DNA failure occurs if a mini-chromosome does not contain a START codon, or ends while encoding a protein, or encodes a protein of length 0.
7. After all mini-chromosomes have been processed, output the properties of the DNA listed above, with text labels that identify them. Please include the properties' units too.
8. Make your program general purpose, so it can handle arbitrary DNA, not just the DNA I provide below. Think about what other inputs you might have to handle and how it would affect the output.

Testing

An important software engineering principle is that code which has been carefully and thoroughly tested will be of higher quality than code which has not been tested. In particular, tested code is less likely to have bugs.[1] An effective and recommended way to achieve careful and thorough testing is to test each software component -- method, class, module -- while it is being built. This is called *unit testing*, which we'll cover later in the course.

In that spirit, include some test input data with your program, and the output expected when processing it. Each test -- *test case* in software engineering jargon -- should test a particular feature of your program. For example, since the program needs to calculate the minimum length of the proteins encoded by the DNA, one test case should input a bunch of proteins and check that your program computes the correct minimum length.

Make test cases that contain zero or more mini-chromosomes and some text in comments or strings associated with the mini-chromosomes that describes the feature of your program being tested and the output that should be produced when `trans_trans` processes it. For example, this DNA consists of 1 mini-chromosome:

```
test_1 = (['ATGATTTAA'],
```

[1] I'm deliberately and consciously not saying that the tested code will not contain bugs.

```
    '''Test feature: encoding of a protein. This DNA encodes for 'I'. It
  has these properties:
    Num encoded proteins: 1
    Shortest protein: 1
    Longest protein: 1
    Mean protein length: 1
    Number of DNA failures: 0
    Amount of non-coding DNA: 0''')
```

Of course, this is an especially simplistic test case, because the DNA encodes just 1 protein that's 1 AA long. Make at least six tests like this and use them to test `trans_trans`. Leave them in your program.

Don't worry about automating your tests.

Example data

To get you started, here's a little DNA for this assignment. Each row is a mini-chromosome.

```
ATGGAACAAGAATGA
ATGATTTAAATGATCTAAATGATTTAA
CATATGATTATTTAAATCATGATTATTTAGGATATGGATATTTAGATT
ATGATTATGTAA
ATTATGTAA
ATGCGTCGT
```