Biomedical Software Engineering

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

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

My teaching assistants from 2018, Ryan and Nicole, wrote an example solution which I revised slightly. It is posted on GitHub. I encourage you to review it and feel free to ask questions or make comments about it.

Grading process

I strongly believe that student programming assignments should be both executed and read. Execution determines which parts work and which parts don't. And reading enables graders to provide feedback on the structure and style of your program. Execution might be unable to provide this feedback. For example, a student program might duplicate a lot of code or use an unnecessarily complex algorithm. These might not be detected by execution but could be detected by reading.

We do this to try to provide you with the most fair, comprehensive, actionable and educationally beneficial program evaluation that we can.

You will find that your programs contain some comments that begin with '`# APG:`' or '`# AG:`'. These are all attempts to provide advice. Some of them identify problems with your program that lose points, while others simply offer coding advice that wasn't required by the assignment.

Feedback

Recommendations

Here are some recommendations that we offered multiple students.

Implementation

1. Code that uses constants instead of small integers is easier to read. E.g., in many places your programs used 3 for the size of a codon. It would be better to define CODON_SIZE (or even CO_SZ if you want a smaller name) as 3, and use that instead. This improves readability because the value 3 could appear for many reasons and a program is easier to understand if the explicit meaning of an integer is clear. This applies to potentially ambiguous strings too.
2. One can replace code like 'if start_pos == []:' with 'if start_pos:'. see True/False evaluations in the Google Python Style Guide.

3. Generally, functions should return strings or other data rather than print so that they can be reused by other code without generating output. Then whether to generate output and the destination of output can be controlled independently. We'll talk about logging later, which is a general technique to address this. In the meantime, a function could print optionally under the control of a keyword parameter that is false by default, like debug=False.
4. Some of you used functions, which are very helpful. How big should a function or method be? A function or method should perform a single task, and do it completely and well. Good examples of functions in this assignment could include these (the solution program included 1, 3, & 4):
    1. process a mini-chromosome
    2. analyze the results obtained by processing a mini-chromosome
    3. process all the DNA, a list or other iterable of mini-chromosomes
    4. output the results of processing the DNA
    5. check that the input is legal, e.g., that DNA is an iterable, and that mini-chromosomes contain valid nucleotides
5. Avoid doing similar work multiple times, and separate work that is distinct. For example, this program needed to scan each mini-chromosome to find its START and STOP codons. Additional coding and computing can be avoided by finding the proteins during this scan. Also, DNA failures in which a scan ends with an incomplete protein should be recorded during a scan. Other types of analysis can be done separately, such as the number of encoded proteins; the length in amino acids of the shortest, and longest protein; the mean protein length in amino acids; and the total amount of non-coding DNA in nucleotides. Processing that can be done separately should be done separately, which enables you to make that code more concise, and clear, and easier to debug by testing or reading.
6. Use data structures and computer science concepts to simplify your code. For example, these approaches could help this program:

    a *state machine*, that represents whether a scan of a min-chromosome is looking for a start codon or looking for a stop codon while recording a protein

    code invariants, which are data properties that should always be true at a particular line of code; Python provides the assert statement to implement these. E.g., if a while loop scanning a mini-chromosome checks that at least 3 nucleotides remain, then the invariant that "the next 3 nucleotides can be interpreted as a codon holds".
7. Since strings are immutable, lists are better for storing data that changes. E.g., if protein is a sequence of AAs, then you could use

    protein = []

    # iterate over DNA

```
            # find another AA
            protein.append(AA)
    # process the protein, e.g., to convert it to a string use
    ".join(protein)
    # see the string method join
```

8. More generally, use the methods available on built-in data structures, like append() and join() above. They're useful and general purpose. Learn about them by reading the documentation on these data structures. Find them by googling for general functionality, like "add element to list python" or "connect elements in list python as a string".
9. Read your code. We can find many mistakes by reading our code. Think about what it does.
10. Avoid spelling mistakes in comments, or in variable names (or use an editor that finds them).
11. Write *general purpose code*. It should handle arbitrary input (more on that in an upcoming class). The comments should make sense for other people using other computers. E.g., trans_trans should work for arbitrarily many mini-chromosomes, proteins that start any codon in a mini-chromosome, etc.Think about and test general input.

Style

1. Python style uses ALL_CAPS names for constants; e.g., see the Global variables and Naming in the Google Python Style Guide.
2. More generally, follow the standard Python naming.

BMSE logistics

1. Ask for help if you need it.
2. Ask for an extension if you need it.