Biomedical Software Engineering

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

Assignment 4: Object-oriented program with unit testing

Software engineering perspective

As we've discussed, Object oriented (OO) programming is a central feature of most of the widely used modern programming languages. This assignment asks you to write an OO program in Python. In particular, please:

1. Refactor some of your existing code into one or more Python classes
2. Design and write some new Python classes
3. Employ *composition* to construct a larger program from multiple class types
4. Use unit testing to debug and analyze the classes and methods you write

One of my primary goals for this assignment is to guide you towards writing simpler classes and methods whose correctness is easier to evaluate, thereby addressing problems that some of you faced in HW 2. I also ask you to unittest them thoroughly.

Assignment overview

Write a program that includes the functionality of HW 2, *Write a Python program that uses the genetic code*, and extends it with variant calling. Implement your solution as an object-oriented program. Make classes with clear, well-defined semantics. Write classes and methods which are simple enough that another person can read and understand them.

Clarification about HW 2

Reading frames: Assume that the beginning of an input mini-chromosome is its 5' end and the other end of the mini-chromosome is its 3' end. Use the following algorithm to find the open reading frames in DNA that encode for proteins in a mini-chromosome:

```
scan next nucleotide:
   if codon at the nucleotide is the START codon:
      # start accumulating coding DNA
      scan next codon:
         if current codon is a STOP codon:
            # stop accumulating coding DNA
            break
         # accumulate current codon as coding DNA

[todo: elaborate like this]
while mini-chromosome contains a nucleotide:
   get next nucleotide
```

```
if codon at the nucleotide is the START codon:
   # start accumulating coding DNA
   while mini-chromosome contains a codon:
      get next codon
      if current codon is a STOP codon:
         # stop accumulating coding DNA
         break
      # accumulate current codon as coding DNA
```

As in HW 2, 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.

Debug and refactor your HW 2 answer

Correct the bugs in your HW 2 answer while converting it into an object-oriented program. This conversion will *refactor* your code. Martin Fowler, a leading software engineering innovator and writer, defines *refactoring* as "a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior". Refactoring is a critical, on-going activity that addresses problems with the internal design and construction of software, and thereby enables future enhancements to the software.

Create a `MiniChromosome` class, and instantiate an instance of it for each mini-chromosome that the program reads. It could have these methods, but feel free to structure it as you see fit:

- `validate`: determine whether the mini-chromosome contains valid DNA.
- `scan`: identify the indices of START and STOP codons in the mini-chromosome while properly treating ATG as a special case because it encodes for both START and methionine.
- `encode`: using the output of *scan*, encode the proteins in the mini-chromosome into sequences of amino acids.
- `analyze`: using the outputs of *scan* and *encode*, compute the mini-chromosome's properties: the # proteins, the lengths of the shortest and longest proteins, the mean protein length, the # of DNA failures, and the total amount of non-coding DNA.
- `__str__`: to separate data analysis from I/O, create a string representation of a `MiniChromosome` that contains its proteins and properties.

Call variants

Add variant calling to your program. Write code that takes a pair of mini-chromosomes. One mini-chromosomes will represent a *reference* chromosome and the other will represent a *subject* chromosome. Identify changes between the reference and the subject as *variants*. To simplify the problem, assume that all variants are single-nucleotide polymorphisms or SNPs. (Calling insertions and deletions is tricky and far beyond the scope of this assignment.) Therefore, report an error if the reference and subject mini-chromosomes do not have the same number of proteins, and the same length in amino acids for each corresponding protein.

Read the mini-chromosomes from files in the NCBI FASTA file format. Use nucleic acid codes, and represent one mini-chromosome in each sequence that follows a description line which

starts with '>'. Make a class to read FASTA files, and use it to read all FASTA files that are input.

A FASTA file will contain a "genome" of one or more mini-chromosomes. All FASTA files for a particular species will contain the same sequence of mini-chromosomes.

If all variants are SNPs, call variants in each pair of proteins in the reference and subject mini-chromosomes. Report the variants in a protein pair as a sequence of amino acid substitutions, reported in the order in which they appear in the subject protein. If the subject protein contains no variants, return an empty sequence.

Use the HGVS standard nomenclature for protein substitutions to encode variants. By convention, the first amino acid in a protein occupies position 1. For example, if a subject protein replaces a Ser at position 3 with an Arg, and replaces Cys at position 12 with a Trp, then report these variants: p.Ser3Arg, p.Cys12Trp.

In addition, call variants in multiple subjects by comparing genomes in multiple subject FASTA files with a reference genome. Read the reference genome from its FASTA file only once while processing multiple subjects.

I recommend that you implement variant calling by creating a class called `VariantCaller` that's initialized with a reference mini-chromosome and has several methods:

- `validate`: determine whether the only variants in a subject mini-chromosome are SNPs
- `call`: call the variants in a subject mini-chromosome, with respect to the initialized reference, and
- `call_sample`: call the variants in one or more subject mini-chromosomes

To separate data analysis from I/O, have these methods return results in data structures. Create another class that uses `VariantCaller` and prints its results.

Document your code

Write a Python docstring for each class and method.

Unit test the code

Write unit tests for the code in this assignment. Following standard Python practice, if your genetics module is called `my_genetics_code.py`, write a testing module named `test_my_genetics_code.py`, and create a unittest `TestCase` class for each class being tested. Where appropriate, continue this parallelism, and have each individual `test_*` method in the `TestCase` test one method in the class being tested.

Ensure that all lines of code in your classes are tested by the unit tests.

Logistics

I welcome emailed questions about this and any other assignment. But I'm having trouble with my MSSM email, so please cc me at artgoldberg@gmail.com.

Questions about the meaning of the assignment must be received more than 48 hours before the assignment is due. Unless it would be inappropriate, in my response I will remove the name of the student who sent me the question and provide the question and my answer to the full

class. That way, everyone receives equal information at the same time. I will check my email and respond to questions at least once a day, unless I'm traveling or ill.

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

Hand in your assignment by emailing me two Python modules, one containing the functional code and another containing the unit tests. (Since this assignment requires two Python modules, please do not write them as Jupyter notebooks.)

This assignment is due at noon on Thurs., 2024-06-20.

Last revision: 2024-06-13