Lab 5

Object-oriented programming

To introduce object-oriented programming, we will look at a computational problem from modern molecular biology. In DNA sequence analysis, we sometimes want to know how much overlap there is between two sequences (simplified: how much DNA they have in common). You will write simple object-oriented code to support such an analysis.

5.1 Learning goals

- You should be able to write your own class, and methods belonging to that class, in order to solve programming problems.
- You should be able to use functions as arguments to functions.
- You should be able to use exceptions to handle errors.

5.2 Submitting your code

A code file with your solutions is to be uploaded to PeerGrade. Submit *one* Python file that solves the tasks.

5.3 Given code

There is a code stub `dna.py` in the lab code directory for the course. This file contains a starting point for your code (quite minimal) plus some test functions. The test functions have two purposes. (1) They make your testing easier and faster. It is often convenient to have some basic testing available. Later, we will look at "real" systematic testing. (2) The test is a codified specification of how your functions should work. You are probably done when the call `test_all()` prints `Yay, all good`—but it is always a good idea to do some testing of your own!

5.3.1 Tasks

5.3.2 Define a class for DNA sequences (2 points)

Define the class DnaSeq to hold two string attributes: an accession (which is supposed to be a unique identifier of a DNA sequence) and a DNA sequence. Name the attributes `accession` and `seq`. It should be possible to define a DnaSeq object like this:

```
my_seq = DnaSeq('abc123', 'ACGTACGT')
```

Requirements

- Your class should have a constructor (`__init__`, as given in the stub) and two methods: `__len__` and `__str__`. Methods with these names have special meaning in Python: if a class has these two methods defined, then one can apply the functions `len` and `str` to them.
- In this exercise, `__len__` should return the length of the DNA sequence (so ignore the accession), and `__str__` should return a "label" for convenient printing looking like

  ```
  <DnaSeq accession='abc123'>
  ```

  when applied to the example above. That is, you should use a template like `<DnaSeq accession='XXX'>` and insert the accession of the object instead of `XXX`.
- Instantiation with empty accession or sequence strings should raise a ValueError exception. That is, a call like `DnaSeq('', None)` should raise the exception.
- The methods should not print anything, just return values.
- When you are done, the call `test_class_DnaSeq()` should print `DnaSeq passed`, and no errors should occur.

5.3.3 Reading DNA sequences from file (3 points)

Implement the function `read_dna()`, which given a filename as parameter returns a list of `DnaSeq` objects representing the sequences in the file. The file format is as follows:

- A file is a sequence of records.
- Each record consists of an *accession line* and a *sequence line.*
- The accession is a unique identifier, represented by a string, and the accession is preceeded by a greater-than sign (>) which is not part of the accession.
- The sequence line is a string of A, C, G, and T.
- There may be blank lines in between the records. They should be ignored. Note that blank lines do not have length 0, because the newline character is part of the input.

Here is a simple example file:

```
>s123
ACGGACGT
>abc
GATTACA

>X20
AAAAAAGAATTACCCACACACAC
```

The data folder on the course web site contains files (suffix ".fa", these are regular textfiles, have a look!) which you can use to test your code on. The files `ex1.fa`, `ex2.fa`, and `sars_cov_2.fa` are used by the tests, so you should download them.

Requirements

- The call `test_reading()` should print `read_dna passed`, and no errors should occur.
- Newline characters need to be removed from accessions and sequences.

- The function should have a docstring.

5.3.4 A function to check overlaps (2 points)

We want to be able to detect *overlaps* between sequence using a function called `check_exact_overlap`. We say that sequence *a* overlaps with sequence *b* if there is a suffix in *a* of some length *L* that is identical to a prefix of length *L* in *b*. Note that this definition is not commutative: *a* can overlap with *b*, without *b* overlapping with *a*.

Example 1: Below we show all the ways the DNA sequence `AAACCC` can overlap with `CCCCGATT`.

```
AAACCC                   AAACCC                   AAACCC
     CCCCGATT                CCCCGATT                CCCCGATT

length 1 overlap         length 2 overlap         length 3 overlap
```

These sequences have no overlap of length 4 or longer. For example, if we try overlaps of length 4 or 5, the two sequences do not agree completely:

```
AAACCC                   AAACCC
  CCCCGATT                CCCCGATT

not an overlap           not an overlap
```

Example 2: Comparing the sequences from Example 1 in the opposite order, `CCCCGATT` has no overlaps with `AAACCC`, because there is no suffix of `CCCCGATT` that is identical to a prefix of `AAACCC`.

```
CCCCGATT                 CCCCGATT                 CCCCGATT
       AAACCC                  AAACCC                  AAACCC       ...

not an overlap           not an overlap           not an overlap
```

Example 3: `GAGATCAT` and `ATCATTT` have two overlaps, one of length 2 and one of length 5.

```
GAGATCAT                 GAGATCAT
      ATCATTT               ATCATTT

length 2 overlap         length 5 overlap
```

Example 4: `AAACCC` overlaps with `ACCC`. In fact, it overlaps completely, with length 4, and this is the only overlap between the sequences.

```
AAACCC                 AAACCC                 AAACCC                 AAACCC
     ACCC                  ACCC                   ACCC                  ACCC

not an overlap         not an overlap         not an overlap         length 4 overlap
```

Implement `check_exact_overlap` to take three parameters: two DnaSeq objects (corresponding to *a* and *b* above) and a minimum length. Overlaps shorter than the minimum length are to be ignored.

Requirements:

- The minimum overlap length parameter should have a default value of 10.

- The length of the longest detectable overlap larger than the minimum length should be returned.
- Return 0 if no overlap is detected.
- The function should have a docstring.

5.3.5 A higher-order function for checking all overlaps (3 points)

Implement the function `overlaps` which takes two parameters:

- A list of DnaSeq objects.
- An overlap detection function.

An example of an overlap detection function is `check_exact_overlap`, but any replacement would do. In the testing code, alternative overlap functions are constructed from `check_exact_overlap` using other minimum overlap lengths. If `lst` is a list of DnaSeq objects, then `overlaps(lst, check_exact_overlap)` is a valid call to the function.

All detectable overlaps among pairs of sequences in the input list are to be returned.

Requirements:

- Return a dictionary of dictionaries containing the lengths of overlaps between sequences. If `d` is the result of a call to `overlaps`, and the sequences with accessions `s1` and `s2` overlaps with length 10, then `d['s1']['s2'] == 10` should be true. If those two sequences were the only input, the nested dictionary could be written as `{'s1': {'s2': 10}}`.
- There should be no empty entries in the returned dictionary. If `s1` does not overlap with any of the input sequences, then there should not be an entry for `s1` in the returned dictionary.
- Nothing should be printed, only returned, by the function.
- The function should have a docstring.
- The call `test_overlap()` should print `overlap code passed` and no errors should occur.

5.4 Notes for the interested

- The format we have worked with in this lab is a simplified version of the Fasta format, hence the file suffix ".fa".
- We look at DNA data as if we know the direction. That is in practice often not the case. One has to consider both forward and backward direction of each sequence, and with respect to the so-called reverse complements of the sequences.
- The type of overlaps we are looking at is the core of *genome assembly*, but one cannot work with perfect overlaps (real sequences have errors) as we do, and one has to use faster algorithms than we are considering since the typical data size is so much larger.