HW 11: DNA, RNA & Peptide Strings

This week you are going to create classes representing a string of DNA, RNA, and peptides. These objects will be called `DNAString`, `RNAString`, and `PeptideString`. This final program will be called `gene-finder.ipynb`.

For those of you who are not familiar with the biology, a gene in DNA can be *transcribed* into RNA. RNA is similar to DNA, except that it has only one strand instead of two, and thymine (T) is replaced by uracil (U). Then, RNA is *translated* into a polypeptide chain. Every codon (three bases) corresponds to one of twenty amino acids, which is determined by the genetic code (contained in the file `genetic-code.tsv`). Three codons (TAA, TAG, and TGA) don't indicate an amino acid, but instead stop the translation. The new polypeptide can then be used to make a protein.

The three classes must contain the methods listed here. The `DNAString` will contain:

- the constructor, which creates a new `DNAString` from an input string. (The method should fail an assertion if it has any inappropriate character.)
- `__str__()`, which returns a string representing this DNAString, with the format `<5'-AAA-3'>`.
- `__len__()`, which returns the length of this DNAString.
- `make_complement()`, which makes a new `DNAString` that is the complement of this `DNAString`.
- `has_gene()`, that returns `True` or `False` depending on whether or not this `DNAString` contains a gene (`ATG` followed by a multiple of three nucleotides, finishing with `TGA`, `TAG`, or `TAA`).
- `transcribe()` which creates a new `RNAString` object from the first gene contained in this `DNAString`. The RNAString will be created using the gene portion of the `DNAString` as a template, with all Ts replaced by Us. If there are multiple genes, start at the first `ATG` and end as early as possible. If there is no gene, it should return `None`.

The `RNAString` will contain:

- the constructor, which creates a new `RNAString` similar to the `DNAString` constructor.
- `__str__()`, which returns a string representing this `RNAString`, with the same format `<5'-AAA-3'>`.
- `__len__()`, which returns the length of this `RNAString`.
- `translate()`, which creates a new `PeptideString` from this `RNAString`, using the genetic code to change every three bases into an amino acid.
- a class variable, holding a dictionary to map codons to the appropriate amino acid. It should also have a class method, that creates this dictionary from the file.

The `PeptideString` will contain:

- the constructor, which creates a new `PeptideString` from a string as above.

- `__str__()`, which returns a string representing this `PeptideString`, with the similar format `<N-MRFV-C>`. (A peptide chain is read from the *N terminus* to the *C terminus*).
- `__len__()`, which returns the length of this `PeptideString`.

Once these classes are created, you must make a program that does the following:

1. Asks the user for a file to open, which will contain DNA information.
2. Loads the file `genetic-code.tsv`, using it to initialize the needed translation information.
3. For every line, it will:
    - Create a DNAString object, printing the length.
    - If a gene exists in the `DNAString`, it will transcribe it to an `RNAString` and then translate that to a `PeptideString` object. It should report the length of each, and print out the final `PeptideString`.
    - If no gene exists, it will state this.
    - It will then do the same with the `DNAString`'s complement.

For example, if the data file contains the following:

```
AAATGCCCCCCTAGCC
ATTACCCCATC
CCCCCCCCCCCCCC
```

...then the output should be similar to this:

```
Please enter a dna file: data.gene

DNA String 1 (16 bases):
contains gene (12 bases), peptide (3 amino acids): <N-MPP-C>
complement contains no gene

DNA String 2 (11 bases):
contains no gene
complement contains gene (9 bases), peptide (2 amino acids): <N-MG-C>

DNA String 3 (14 bases):
contains no gene
complement contains no gene
```