Overview

In this homework, we will write a program that will print the peptide (a string of amino acids) from four pieces of information:

- A DNA sequence (a string).
- The *strand* the gene appears on (a string). If `'+'`, then the DNA is from the positive strand. If `'-'`, then the DNA is from the reverse complement (negative) strand.
- A list of exon start coordinates (a list of integers).
- A list of intron start coordinates (a list of integers).

To check that your function is correct, you will have the mRNA sequence and the peptide sequence to compare to. You are given four different examples of input data (in the `datadir/` directory).

- `testPos` is a short example on the positive strand.
- `SRC` is the SRC gene we've seen in HW2. This is a gene on the positive strand.
- `testNeg` is a short example on the negative strand.
- `TMPRSS2` is a gene on the negative strand.

When HW3 is completed, an example output to the Terminal will look something like this:

```
Dataset: testPos
DNA: CCCATGGTCGGGGGGGGGAGTCCATAACCC
Num exons: 2
strand: +
RNA (from file): AUGGUCAGUCCAUAA
peptide (from file): MVSP*

Computed mRNA: AUGGUCAGUCCAUAA
mRNA matches!

Computed Peptide: MVSP*
Peptide matches!
```

The evaluation criteria will include the following components.

- **Correctness.** Your program should compare computed values with the sequences provided in the `datafiles/` directory. You are encouraged to included other "sanity checks:"
- **Documentation.** You should explain almost every piece of your code with comments (`#`). Additional `print` statements can also help explain the contents of variables.
- **Organization.** Functions make your program "easier" to navigate, since they are blocks of code with specific goals. Follow the organization provided in the starter code.

Internal Deadlines: This is a multi-part homework; you should be finishing up HW3.1 by Friday, HW3.2 by Monday, and wrapping up the entire homework by the end of next week. Ask for help if you have trouble meeting the internal deadlines. You can meet with Rose, Mina, Elaine, or me.

Challenge: If you have written programs before, follow the instructions denoted by this tag.

When you're done, write your name and the estimated amount of time you spent at the top of the file. You can divide the time by parts (e.g., HW3.1, HW3.2, HW3.3).

HW3.1: Write a `transcribe()` Function

1. Become familiar with the starter code in `HW3.py`. You should be able to run it and have some print statements appear. ***Challenge:*** Use the starter code in `HW3_challenge.py`, which is missing the function definitions.

   (a) At the very bottom of the file, there are two lines that will call the `main()` method. Thus, the `main()` method gets called *after* all the functions have been loaded into memory.

   ```
   if __name__ == '__main__':
       main()
   ```

   (b) I have written some functions that convert files into Python variables. There is an `import` line at the top of the file that "reads" all the function definitions in the file `anna_functions.py`.

   ```
   import anna_functions
   ```

   (c) In the `main()` method there is a variable named `dataset` that contains one of the dataset names from the `datafiles/` directory (e.g., `'testPos'`).

   (d) Finally, there is a line that loads six variables into memory by calling the getData() function in `anna_functions.py` (or `anna_challenge_functions.py`)

   ```
   dna,exonStarts,intronStarts,strand,rnaFromFile,peptideFromFile =
    ↪  anna_functions.getData(dataset)
   ```

   Note the line wrap - this is all be written on one line in the `.py` code.

2. About half of the output described on the first page simply prints statistics about the inputs. Write a function to print the information to the screen.

   `printInfo()`: *Print all the information to the screen.*

   Inputs: Six variables containing the dataset information `dna`, `exonStarts`, `intronStarts`, `strand`, `rnaFromFile`, and `peptideFromFile`.

   Returns: Nothing.

   Your program should now output the following:

   ```
   Dataset: testPos
   DNA: CCCATGGTCGGGGGGGGGGAGTCCATAACCC
   Num exons: 2
   strand: +
   RNA (from file): AUGGUCAGUCCAUAA
   peptide (from file): MVSP*
   ```

3. Write a `transcribe()` function.

   `transcribe()`: *Transcribe DNA to mRNA.*

   Inputs: A DNA string (5' to 3' direction), a list of exonStarts, and a list of intronStarts.

   Returns: A string representing the mRNA.

   Within your `transcribe()` function, you should have a `pre_mRNA` variable that represents the pre_mRNA before splicing. You can ignore the poly-A tail and guanine capping steps of RNA processing.

   You may end up borrowing some code from `HW2.3` and `Lab3.py`.

4. Print the computed mRNA to Terminal. Write a test to compare your computed mRNA sequence with the mRNA sequence provided in the data files. Use an `IF` statement to print a message if they are the same or if they are different. Your program should now output the following:

```
Dataset: testPos
DNA: CCCATGGTCGGGGGGGGGAGTCCATAACCC
Num exons: 2
strand: +
RNA (from file): AUGGUCAGUCCAUAA
peptide (from file): MVSP*

Computed mRNA: AUGGUCAGUCCAUAA
mRNA matches!
```

5. Assign the string `'SRC'` to the `dataset` variable. Verify that your program works with this dataset as well.

   - To reduce the amount of text for large sequences, use an `IF` statement to print the DNA/mRNA sequence if it smaller than 50 letters long, and print the *length* of the DNA/mRNA sequence if it is larger than 50 letters long.

HW3.2: Write a `translate()` Function

1. Become familiar with the function `getAminoAcid()` that Anna has written. Enter these lines in your program one by one, then comment them out or delete them. (***Challenge:*** see next step)

```
print anna_functions.getAminoAcid('AAA')
print anna_functions.getAminoAcid('AUG')
print anna_functions.getAminoAcid('UAA')
print anna_functions.getAminoAcid('aug')
print anna_functions.getAminoAcid('AU')
print anna_functions.getAminoAcid('XYZ')
```

2. ***Challenge:*** The `getAminoAcid()` function does not exist in `anna_challenge_functions.py`. Write your own function that converts a codon to an amino acid.

   > `getAminoAcid()`: *Converts a codon to an amino acid.*
   >
   > **Inputs:** A string representing a codon.
   >
   > **Returns:** A string representing a single letter amino acid, or `'*'` for a stop codon.

   - Consider using Python dictionaries, which keep track of key/value pairs: `https://docs.python.org/3/tutorial/datastructures.html#dictionaries`. We will discuss these later in the semester.
   - Catch incorrect inputs report this to the terminal. Test your function on the inputs listed above.
   - If you'd like to skip this challenge, you can copy the function from `anna_functions.py`.

3. Write a `translate()` function. Again, start with the `'testPos'` dataset.

   > `translate()`: *Translate mRNA to a peptide sequence.*
   >
   > **Inputs:** An mRNA string (5' to 3' direction)
   >
   > **Returns:** A string representing the peptide.

   You will need to figure out how to take 3-letter slices of the `mRNA` string. You will also need to "jump" to every third position in the string to output non-overlapping codons. There are a couple ways to do this:

   - The `range()` function provides a way to return every 3rd integer. Look up its usage in the Python documentation:

     `https://docs.python.org/3/library/functions.html#func-range`
   - The remainder (or modulo,%) of a number divided by 3 will either be 0, 1, or 2. Use the remainder of the index to print codons.

4. Print the computed peptide to terminal. Write a test to compare your computed peptide with the peptide provided in the data files. Use an `IF` statement to print a message if they are the same or if they are different. **Your program output should look like the text on the first page.**

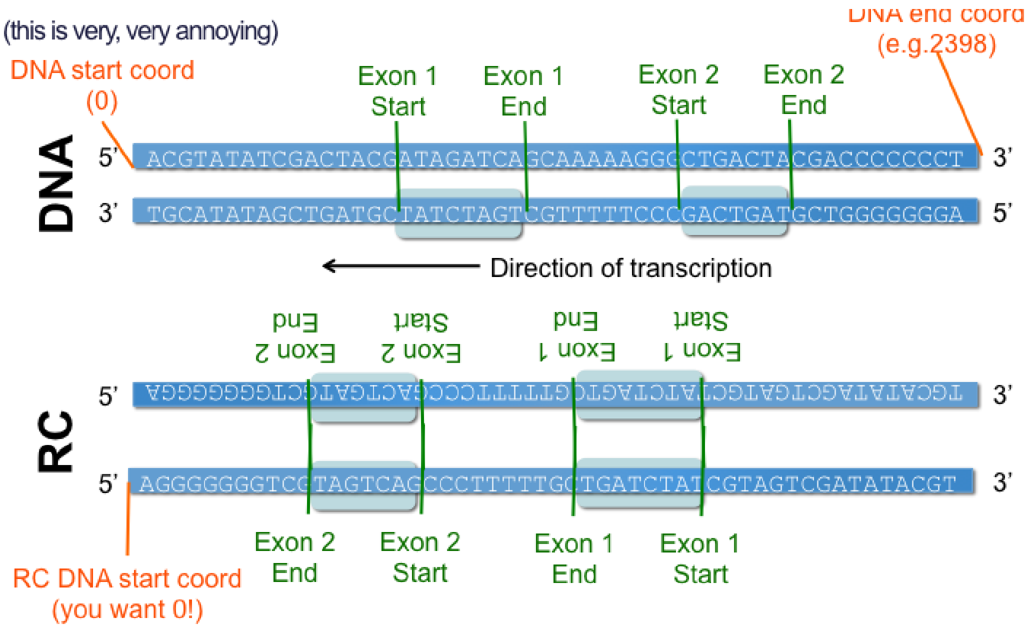
5. Assign the string `'SRC'` to the `dataset` variable. Verify that your program works with this dataset as well.

HW3.3: Allow Genes from the Negative Strand

The UCSC Genome Browser uses standardized file formats for representing genes and sequences.

- *All sequences appear in the 5' to 3' direction.* When the strand is negative (`'-'`), it means that the sequence in the file is the *reverse complement* of the positive strand.
- *All coordinates are reported with respect to the positive strand.* You need to adjust the coordinates for the exon and intron starts.

The first exon on the negative strand is described by the *last* exon listed in the `exonStarts` and `intronStarts` lists.


(this is very, very annoying)
DNA start coord (0)
DNA end coord (e.g.2398)
Exon 1 Start
Exon 1 End
Exon 2 Start
Exon 2 End
DNA
5'
3'
ACGTATATCGACTACGATAGATCAGCAAAAAGGGCTGACTACGACCCCCCCT
TGCATATAGCTGATGCTATCTAGTCGTTTTTCCCGACTGATGCTGGGGGGGA
3'
5'
Direction of transcription
Exon 2 End
Exon 2 Start
Exon 1 End
Exon 1 Start
RC
5'
TGCATATAGCTGATGCTATCTAGTCGTTTTTCCCGACTGATGCTGGGGGGGA
3'
5'
AGGGGGGGTCGTAGTCAGCCCTTTTTGCTGATCTATCGTAGTCGATATACGT
3'
Exon 2 End
Exon 2 Start
Exon 1 End
Exon 1 Start
RC DNA start coord (you want 0!)

1. Open the files `datafiles/testPos-sequences.fasta` and `datafiles/testNeg-sequences.fasta` in a plain text editor. Determine the relationship between these two test datasets.
2. You need to modify some code so the `mRNA` computed is consistent with genes on the negative strand. Write an `IF` statement that checks to see whether the strand is `'+'` or `'-'`.
   - If the strand is positive, you simply need to call the `transcribe()` function you have written for HW3.1.
   - If the strand is negative, print `'Not Yet Implemented!'`. This is a placeholder until you figure out what to write here.
3. Work out the conversion for the `testNeg` dataset by hand.
4. Assign the string `'testNeg'` to the `dataset` variable. Replace the line `'Not Yet Implemented!'` with code so your program works on `'testNeg'`. *Hint:* You don't need to modify `transcribe()` or the `dna` variable; instead, modify the `exonStarts` and `intronStarts` variables.
5. Assign the string `'TMPRSS2'` to the `dataset` variable. Verify that your program works with this dataset as well.
6. Write your name and the estimated amount of time you spent at the top of the file. You can divide the time by parts (e.g., HW3.1, HW3.2, HW3.3).