1. Translation

Gene expression involves the processes of **transcription** and **translation**. First DNA transcribes into RNA. DNA is made up of the bases A, G, C, and T. RNA is similar, but contains the base U instead of T. There is a one to one correspondence between DNA and RNA bases.

Next, the messenger form of RNA (mRNA) is translated into a sequence of amino acids. There is a three to one correspondence between mRNA bases and amino acids. In other words, each set of three mRNA bases codes for one amino acid. The RNA codon table shows the translation of mRNA triplets into the appropriate amino acid. Fortunately for you, you don't have to know those details. We have built a library in python to translate an mRNA codon. To see how it works, try this:

```
$ python
>>> from genetics import *
>>> translate('UUU')
'F'
>>> translate('UUA')
'L'
```

Check that the results match the data given in the RNA codon table. If you'd rather translate each triplet into the longer abbreviations of the amino acid names you can do:

```
>>> translateName('UUU')
'Phe'
>>> translateName('UUA')
'Leu'
```

For the first part of this assignment, you will be focusing on the translation process. You will be writing a program, in the file **`translate.py`**, that first generates a random string of mRNA. You will ask the user to enter a desired length, and then use the **`genRNA`** function from the **`gencode`** library to create this mRNA string. For example, to create an mRNA string of length 21:

```
$ python
>>> from gencode import *
>>> genRNA(21)
'UGCUUGGGACGAUCGGAUCUU'
```

Next, you will translate the mRNA string into the appropriate amino acid sequence using the **`translate`** function from the **`genetics`** library. The `translate` function takes in one codon (i.e., three bases) at a time. So, to translate the above sequence, `'UGC'` translates into `'C'`, then `'UUG'` is translated to `'L'`, creating a protein sequence of `'CL'`, and so on. Notice a pattern?

When importing libraries into a program always put the **`import`** statements at the top of the file, prior to the definition of the **`main`**.

Your program should produce the following sort of interaction with the user:

```
Please enter the desired mRNA sequence length: 21
GCACAAACGUUGCCGCUCUAA

The translated protein sequence contains 7 amino acids:
AQTLPL$
```

Once you have that working, you will need to implement a second algorithm. After outputting the translated protein, you should create and output the original mRNA string in **reverse** order. A complete run of the program will appear as follows:

```
This program will translate an mRNA sequence into a protein sequence
and then reverse the mRNA sequence.

Please enter the desired mRNA sequence length: 27
UCCGUCGAUCUGUAGCCACCCACGAGA

The translated protein sequence:
SVDL$PPTR

The mRNA string in reverse is:
AGAGCACCCACCGAUGUCUAGCUGCCU
```

2. Transcription

Biologists often get a piece of DNA sequence and want to know what's in it. One of the most obvious questions to ask is, does it contain a gene? Because genomes of organisms consist of many non-coding regions, it's not clear that a random piece of DNA will always have a gene. And if there is a gene, where does it begin and end? A simple strategy for finding genes is to look for open reading frames. An open reading frame is the section of a sequence between a start codon and a stop codon.

Gene expression involves the processes of transcription and translation. Last week we focused on implementing translation. This week we will implement transcription, and then search for an open reading frame in the resulting mRNA. To simplify the program we will only search for an open reading frame at offset 0. If an open reading frame is found, we will translate it into the appropriate amino acid sequence.

You will write a single program called `transcribeAndTranslate.py` to perform all of the necessary steps. You should create your solution incrementally by following the instructions given below. Test each step of your partial solution. Do not go onto the next step until the previous step is working correctly.

Below is a sample run of the program in which an open reading frame was found. The start and stop codons have been highlighted in red for clarity, but your program need not do this.

```
This program simulates both transcription and translation.

Enter length of random DNA string as a multiple of 3: 300

antisense strand of DNA:
CGTCCGAGGCTGTGGCCAATTACTGTCAACTGCAGAGTGTACGTGATATGAATGATTTAGATG
GGGCCTCCTTGGACGTCGTGCGGTGAGAGAGGCGAGCACAGATAGGTACTGGAAATGTACCGT
TCTAGTTCGTGATTTACGCACGTGGAGATTGCCGCGTCGTCCACAATAGTGGCACGAGAATGT
TCGGAGTTAAGACTTAATATCAAGAACAAGATGTCGCGGAGGACAGCGGGTCTGAAAGATGCG
CTTTATCAGACACCTGCATGACCCTATGATACCTGAAACCTACTGGGA

mRNA:
GCAGGCUCCGACACCGGUUAAUGACAGUUGACGUCUCACAUGCACUAUACUUACUAAAUCUAC
CCCGGAGGAACCUGCAGCACGCCACUCUCUCCGCUCGUGUCUAUCCAUGACCUUUACAUGGCA
AGAUCAAGCACUAAAUGCGUGCACCUCUAACGGCGCAGCAGGUGUUAUCACCGUGCUCUUACA
AGCCUCAAUUCUGAAUUAUAGUUCUUGUUCUACAGCGCCUCCUGUCGCCCAGACUUUCUACGC
GAAAUAGUCUGUGGACGUACUGGGAUACUAUGGACUUUGGAUGACCCU

Found an open reading frame starting at 39 and ending at 54
MetHisTyrThrTyrSTOP
```

Here is a sample run of the program in which an open reading frame was not found:

```
This program simulates both transcription and translation.

Enter length of random DNA string as a multiple of 3: 75

antisense strand of DNA:
GACAAGCCTCCGCTTAGTCTTTTTCCGTGTTGCGTGGAGTTACTTGACTATTATAAAAGGCGT
TATCCGTTACAG

mRNA:
CUGUUCGGAGGCGAAUCAGAAAAAGGCACAACGCACCUCAAUGAACUGAUAAUAUUUUCCGCA
AUAGGCAAUGUC

No open reading frame found
```

1. Generate a random antisense strand of DNA

DNA is composed of two strands, termed sense and antisense. The antisense strand is a complement of the sense strand as shown in the small example here:

```
AGAATGGCCTGGTAAGGC  sense strand of DNA
TCTTACCGGACCATTCCG  antisense strand of DNA
```

Generate a random antisense strand of DNA represented as a string. Use the `choice` function from the `random` library to randomly select from the bases T, A, G, and C. [Please note: The sense strand of DNA is provided above for illustration only. You do not need to generate the sense strand. Your program begins with generating the antisense strand of DNA.]

2. Transcribe the antisense strand of DNA into mRNA

When transcribing the antisense strand of DNA, T becomes A, A becomes U, G becomes C, and C becomes G. The transcription of the antisense strand is almost identical to the original sense strand except that the T's are replaced with U's. For instance, if we continue with the previous example:

```
TCTTACCGGACCATTCCG  antisense strand of DNA
AGAAUGGCCUGGUAAGGC  transcription into mRNA
```

Create an mRNA string representing the transcription of the anitsense strand from the previous step.

3. Search for an open reading frame in the mRNA

Only particular sections of the mRNA can code for proteins and these are termed open reading frames. An open reading frame begins with a particular start codon: AUG. The open reading frame can end with several different stop codons: UAA, UGA, or UAG. For example, the following mRNA contains a short open reading frame that begins at position 3 (starting from 0) and ends at position 12 (the first letter of the stop codon).

```
AGAAUGGCCUGGUAAGGC  open reading frame found in mRNA
```

Write a `for` loop to find the position of the first start codon in the mRNA string, if one exists. Use a `break` statement to exit from the loop as soon as a start codon is found.

Write another `for` loop to look for the first stop codon beginning from the location of a found start codon, if one exists. Use a `break` statement to exit from the loop as soon as a stop codon is found.

If both a start and stop codon were found, report the locations of the open reading frame.

4. When an open reading frame is found, translate it

If an open reading frame was found, use the `translateName` function from the `genetics` library to translate the open reading frame into amino acids. For example the open reading frame found in the previous step translates to:

```
MetAlaTrpSTOP
```

If no open reading frame was found, report this.

Hints and Tips

1. Be sure to attack the problem as 4 subproblems. You should begin with step 1 (generating random DNA) before moving to step 2. Be sure to thoroughly test each subproblem before moving on.
2. Remember to import the `random` library to use `choice`, which takes in a sequence (or list) of items and randomly selections amongst those options.
3. Testing out your solution with a random string can be difficult. You can always test it out with a fixed, small DNA sequence first. For example use a variable `testDNA` that is set to the example above
4. `testDNA = "TCTTACCGGACCATTCCG"`

   and then test the program to see if you get the same output as in the example. Once that is working, you can remove testDNA and use the randomly generated sequence.

Optional extensions

These suggestions are **not** required.

If you're interested in making your program more complete, you can enhance it so that it searches for open reading frames at offset 0, 1, and 2 and reports the first one found.

A further enhancement is to report the longest open reading frame found at any of the possible offsets.