CS 115, Autumn 2021
Programming Project #7: DNA (20 points)
Due Tuesday, November 23rd, 11:30 PM

Special thanks to Marty Stepp and UW CSE professor Martin Tompa.

This assignment focuses on parameters, returns, lists and file processing. Turn in a file named `dna.py`. You will also need the three input files `cancer_cure.txt`, `hair_growth` and `ecoli.txt` from the course web site. Save these files in the same folder as your program.

The assignment involves processing data from genome files. Your program should work with all of the given input files.

Background Information About DNA:

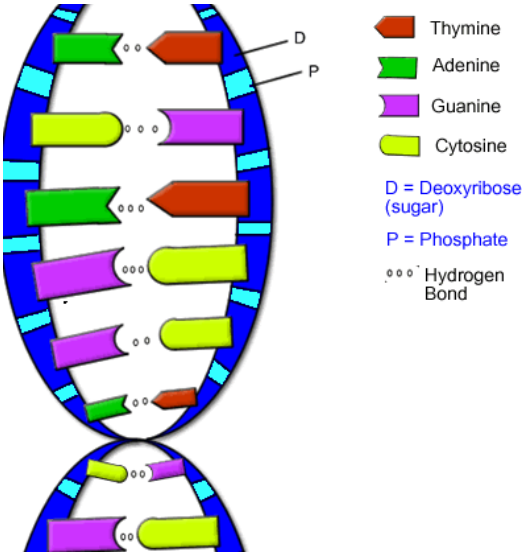
Note: This section explains some information from the field of biology that is related to this assignment.
It is for your information only; you do not need to fully understand it to complete the assignment.

Deoxyribonucleic acid (DNA) is a complex biochemical macromolecule that carries genetic information for cellular life forms and some viruses. DNA is also the mechanism through which genetic information from parents is passed on during reproduction. DNA consists of long chains of chemical compounds called *nucleotides*. Four nucleotides are present in DNA: Adenine (A), Cytosine (C), Guanine (G), and Thymine (T). DNA has a double-helix structure (see diagram below) containing complementary chains of these four nucleotides connected by hydrogen bonds.

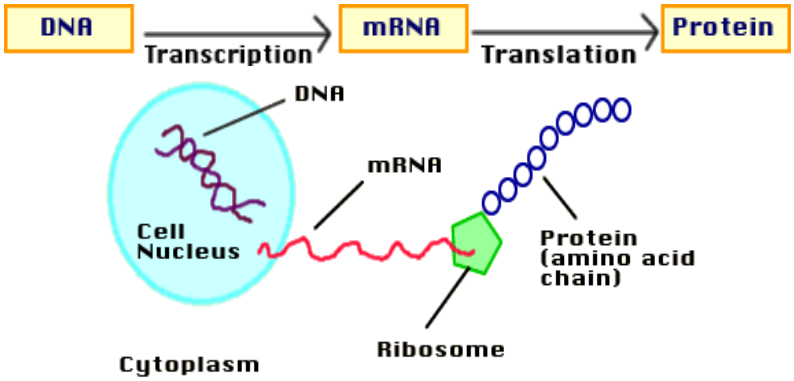
Certain regions of the DNA are called genes. Most genes encode instructions for building proteins (they're called "protein-coding" genes). These proteins are responsible for carrying out most of the life processes of the organism.

Nucleotides in a gene are organized into *codons*. Codons are groups of three nucleotides and are written as the first letters of their nucleotides (e.g., TAC or GGA). Each codon uniquely encodes a single amino acid, a building block of proteins.

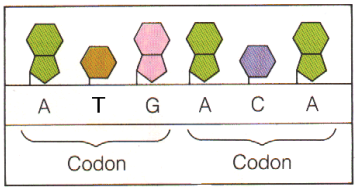
The process of building proteins from DNA has two major phases called *transcription* and *translation*, in which a gene is replicated into an intermediate form called *mRNA*, which is then processed by a structure called a *ribosome* to build the chain of amino acids encoded by the codons of the gene.



The chemical structure of DNA.



DNA translation.



The sequences of DNA that encode proteins occur between a *start codon* (which we will assume to be ATG) and a *stop codon* (which is any of TAA, TAG, or TGA). Not all regions of DNA are genes; large portions that do not lie between a valid start and stop codon are called *intergenic DNA* and have other (possibly unknown) functions. Computational biologists examine large DNA data files to find patterns and important information, such as which regions are genes. Sometimes they are interested in the percentages of mass accounted for by each of the four nucleotide types. Often high percentages of Cytosine (C) and Guanine (G) are indicators of important genetic data.

For more information, visit the Wikipedia page about DNA: http://en.wikipedia.org/wiki/DNA

In this assignment you read an input file containing a sequence of nucleotides and produce information about it. Your program must output the **counts** the occurrences of each of the four nucleotides (A, C, G, and T). The program also should compute the **mass percentage** occupied by each nucleotide type, rounded to one digit past the decimal point. Next the program should report the **codons** (trios of nucleotides) present in each sequence and predict whether or not the sequence is a **protein**-coding gene. For us, a protein-coding gene is a file of nucleotides that matches all of the following constraints*:

- begins with a valid *start codon* (ATG)
- ends with a valid *stop codon* (one of the following: TAA, TAG, or TGA)
- contains at least 5 total codons (including its initial start codon and final stop codon)
- Cytosine (C) and Guanine (G) combined account for at least 30% of its total mass

*(*These are approximations for our assignment, not exact constraints used in computational biology to identify proteins.)*

The DNA input file consists of lines containing nucleotides separated by one or more spaces each. Nucleotides will be A, C, G or T.

Input file `cancer_cure.txt`:

```
A T G C C A C T A T G G T A G
```

Program Behavior:

Your program begins with an introduction and prompts for an input file name. You may assume the user will type the name of an existing input file that is in the proper format. Your program should read the input file to process its nucleotide sequence and output the result. Notice that the nucleotide counts and mass percentages are shown in A, C, G, T order. A given codon such as GAT might occur more than once in the same sequence.

Log of execution (user input underlined):

```
This program reports information about DNA
nucleotide sequences that may encode proteins.
Input file name? cancer_cure.txt

Nucleotides: ATGCCACTATGGTAG
Nuc. Counts: [4, 3, 4, 4]
Total Mass%: [27.3, 16.8, 30.6, 25.3] of 1978.8
Codons List: ['ATG', 'CCA', 'CTA', 'TGG', 'TAG']
Is Protein?: YES
```

Implementation Guidelines, Hints, and Development Strategy:

The main purpose of this project is to demonstrate your understanding of lists and list traversals with for loops. Therefore, you should use lists to store the various data for each sequence. In particular, **your nucleotide counts, mass percentages, and codons should all be stored using lists**. Additionally, you should **use lists and `for` loops** to transform the data from one form to another as follows:

- from the original nucleotide sequence string to nucleotide counts;
- from nucleotide counts to mass percentages; and
- from the original nucleotide sequence string to codon triplets.

These transformations are summarized by the following diagram using the `cure_cancer.txt` input file:

```
Nucleotides:  "ATGCCACTATGGTAG"
    ↓         What is computed                  Output to file
Counts:       {4, 3, 4, 4}                      Nuc. Counts: [4, 3, 4, 4]
    ↓
Mass %:       {27.3, 16.8, 30.6, 25.3}          Total Mass%: [27.3, 16.8, 30.6, 25.3] of 1978.8
    ↓
Codons:       {ATG, CCA, CTA, TGG, TAG}         Codons List: [ATG, CCA, CTA, TGG, TAG]
                                                Is protein?: YES
```

To compute **mass percentages**, use the following as the mass of each nucleotide (grams/mol).

- Adenine (A): 135.128
- Cytosine (C): 111.103
- Guanine (G): 151.128
- Thymine (T): 125.107

For example, the mass of the sequence ATGGAC is (135.128 + 125.107 + 151.128 + 151.128 + 135.128 + 111.103) or 808.722. Of this, 270.256 (29.7%) is from the two Adenines; 111.103 (12.2%) is from the Cytosine; 302.256 (33.3%) is from the two Guanines; and 125.107 (13.8%) is from the Thymine.

We suggest that you start this program by writing the code to read the input file into a list. Print it out to check that you are reading and separating nucleotides correctly.

Next, write code to pass over the list of nucleotides and print out each letter on the same line with no spaces.

Then, write code to pass over the list of nucleotides and count the number of As, Cs, Gs, and Ts. Put your counts into a list of size 4.

Write a function to map between nucleotides and list indexes to help you figure out which count to add to for each nucleotide. Store the A count at index 0, the C count at index 1, the T count at index 2 and the G count at index 3.

Once you have the counts working correctly, you can convert your counts into a new list of percentages of mass for each nucleotide using the preceding nucleotide mass values

You can round percentages to one digit after the decimal by using Python's built-in `round` function. This function takes a real number and a number of digits past the decimal to round to as parameters and returns the real number with that number of digits after the decimal. For example,

```
number = 234.2876800001
result = round(number, 1)
print(result)
```

would print out `234.2`.

It will be easiest to print your mass percentages if you round them and store the rounded result back in your list of masses. However, make sure you do not round them early or you will lose precision and may get a slightly incorrect result. Only round them when you have finished all the computation you need to do with them.

After computing mass percentages, you must break apart the sequence into codons and examine each codon. We suggest that you loop through your nucleotide list 3 indexes at a time, concatenating together the nucleotide characters at the current and two following indexes and adding the result to the end of a new list. Remember, you must use `append` to add a new item to a list instead of [].

You may assume that the input file exists, is readable, and contains valid input. (In other words, you should not re-prompt for the input file name.) You may assume that the sequence's number of nucleotides will be a multiple of 3.

Style Guidelines:

For full credit on this project, **you may only read the data in the input file once**.

For this assignment you are required to have the following **four constants**:

- one for the **minimum number of codons** a valid protein must have, as an integer (default of 5)
- a second for the **percentage** of mass from C and G in order for a protein to be valid, as an integer (default of 30)
- a third for the number of **unique nucleotides** (4, representing A, C, G, and T)
- a fourth for the number of **nucleotides per codon** (3)

For full credit it should be possible to change the first two constant values (minimum codons and minimum mass percentage) and cause your program to change its behavior for evaluating protein validity. The other two constants won't ever be changed but are still useful to make your program more readable. Refer to these constants in your code and do not refer to the bare number such as 4 or 3 directly. You may use additional constants if they make your code clearer.

(Continued on next page)

For this assignment you are required to have the following **four functions**:

- one to **map between nucleotides and list indexes** to help you figure out which count to add to for each nucleotide (described above). This function should take a nucleotide and return the index its count is stored in.
- a second to **print out the nucleotides** (the text from all lines from the original file without the new lines).
- A third to **count the nucleotides**. These counts must be stored in a list. You may either pass this list into your function and fill it up or you may create it in your function and return it.
- A fourth to **compute the mass percentages**. This function will need to know about the counts in order to be able to figure out the masses. You may not recompute the counts.
- A fifth to **generate the codons list**.
- A sixth to **determine if the data is a protein.**

Your `main` function should be a concise summary of the overall program. It is okay for `main` to contain some code such as `print` statements. But `main` should not perform too large a share of the overall work itself, such as examining each index in a list. Also avoid "chaining," when many functions call each other without ever returning to `main`. All of the above functions, except the first, should be called from `main`. Note that this means you will need to return some data computed in your functions.

We will check carefully for redundancy on this project. If you have a very similar piece of code that is repeated several times in your program, eliminate the redundancy such as by creating a function, by using `for` loops over the elements of lists, and/or by factoring `if/else` code as described earlier this quarter.

Since lists are a key component of this assignment, part of your grade comes from using lists properly. For example, you should reduce redundancy as appropriate by using **traversals** over lists (`for` loops over the list's elements). This is preferable to writing out a separate statement for each list element (a statement for element `[0]`, then another for `[1]`, then for `[2]`, etc.).

You are limited to the material we have covered in class this quarter. Follow past style guidelines such as names, variables, line lengths, and comments (at the beginning of your program, on each function, and on complex sections of code).