CSc 110, Spring 2017
Programming Assignment #7: DNA (20 points)
Due Tuesday, March 7th, 7:00 PM

Thanks to Martin Tompa and Marty Stepp from the University of Washington CSE.

This assignment focuses on lists and file/text processing. Turn in a file named `dna.py`. You will also need the two input files `dna.txt` 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 the two given input files. If you are curious (this is not required), the National Center for Biotechnology Information publishes many other bacteria genome files. The last page tells you how to use your program to process other published genome files.

Background Information About DNA:

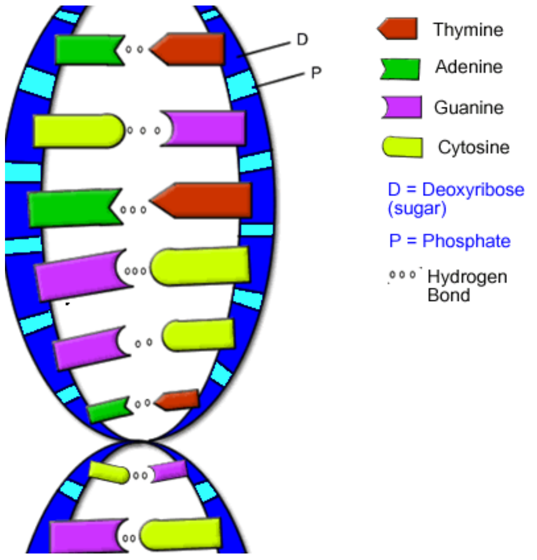
Note: This section explains some information from the field of biology that is related to this assignment. It is for your information only; you do not need to fully understand it to complete the assignment.

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.

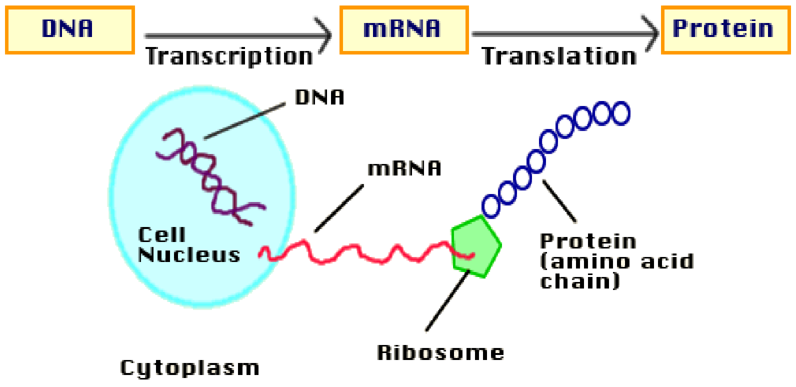
Certain regions of the DNA are called genes. Most genes encode instructions for building proteins (they're called "protein-coding" genes). These proteins are responsible for carrying out most of the life processes of the organism.

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.

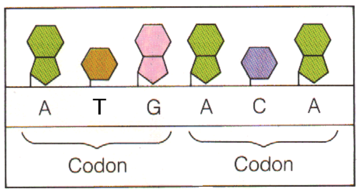
The process of building proteins from DNA has two major phases called *transcription* and *translation*, in which a gene is replicated into an intermediate form called *mRNA*, which is then processed by a structure called a *ribosome* to build the chain of amino acids encoded by the codons of the gene.



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 named sequences of nucleotides and produce information about them. For each nucleotide sequence, your program **counts** the occurrences of each of the four nucleotides (A, C, G, and T). The program also computes the **mass percentage** occupied by each nucleotide type, rounded to one digit past the decimal point. Next the program reports the **codons** (trios of nucleotides) present in each sequence and predicts whether or not the sequence is a **protein**-coding gene. For us, a protein-coding gene is a string 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 data consists of line pairs. The first line has the name of the nucleotide sequence, and the second is the nucleotide sequence itself. Each character in a sequence of nucleotides will be A, C, G, T, or a dash character, `"-"`. The nucleotides in the input can be either upper or lowercase.

Input file `dna.txt` (partial):

```
cure for cancer protein
ATGCCACTATGGTAG
captain picard hair growth protein
ATgCCAACATGgATGCCcGATAtGGATTgA
bogus protein
CCATt-AATgATCa-CAGTt
```
...

The **dash `"-"` characters** represent "junk" or "garbage" regions in the sequence. For most of the program they should be ignored in your computations, though they do contribute to the total mass of the sequence as described later.

Program Behavior:

Your program begins with an introduction and prompts for input and output file names. You may assume the user will type the name of an existing input file that is in the proper format. Your program reads the input file to process its nucleotide sequences and outputs the results into the given output file. Notice the nucleotide sequence is output in uppercase, and 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? dna.txt
Output file name? output.txt
```

Output file `output.txt` after above execution (partial):

```
Region Name: cure for cancer protein
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

Region Name: captain picard hair growth protein
Nucleotides: ATGCCAACATGGATGCCCGATATGGATTGA
Nuc. Counts: [9, 6, 8, 7]
Total Mass%: [30.7, 16.8, 30.5, 22.1] of 3967.5
Codons List: [ATG, CCA, ACA, TGG, ATG, CCC, GAT, ATG, GAT, TGA]
Is Protein?: YES

Region Name: bogus protein
Nucleotides: CCATT-AATGATCA-CAGTT
Nuc. Counts: [6, 4, 2, 6]
Total Mass%: [32.3, 17.7, 12.1, 29.9] of 2508.1
Codons List: [CCA, TTA, ATG, ATC, ACA, GTT]
Is Protein?: NO
```

Implementation Guidelines, Hints, and Development Strategy:

The main purpose of this assignment 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 for cancer" protein data:

	What is computed	Output to file
Nucleotides:	`"ATGCCACTATGGTAG"`	
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`

Recall that you can print any list using `str()`. For example:

```
numbers = [10, 20, 30, 40]
print("my data is " + str(numbers))     # my data is [10, 20, 30, 40]
```

To compute **mass percentages**, use the following as the mass of each nucleotide (grams/mol). The dashes representing "junk" regions are excluded from many parts of your computations, but they *do* contribute mass to the total.

• Adenine	(A):	135.128
• Cytosine	(C):	111.103
• Guanine	(G):	151.128
• Thymine	(T):	125.107
• Junk	(-):	100.000

For example, the mass of the sequence ATGG-AC is (135.128 + 125.107 + 151.128 + 151.128 + 100.000 + 135.128 + 111.103) or 908.722. 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; 125.107 (13.8%) is from the Thymine; and 100.000 (11.0%) is from the "junk" dash.

We suggest that you start this program by writing the code to read the input file. Try writing code to simply read each protein's name and sequence of nucleotides and print them.

Next, write code to pass over a nucleotide sequence and count the number of As, Cs, Gs, and Ts. Put your counts into a list of size 4. To map between nucleotides and list indexes, you may want to write a function that converts a single character (i.e. A, C, T, G) into indices (i.e. 0 to 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. If you've written code to map between nucleotide letters and list indexes, it may also help you to look up mass values in a list such as the following:

```
masses = [135.128, 111.103, 151.128, 125.107]
```

You may store your mass percentages already rounded to one digit past the decimal or you can round when printing the mass percentages list.

Remember that the "junk" dashes do contribute mass to the total. For other parts of your program you may want to remove dashes from the input.

After computing mass percentages, you must break apart the sequence into codons and examine each codon. You may wish to review the `String`s as presented in lecture 11, such as `substring`, `[]`, `upper`, and `lower`.

We also suggest that you first get your program working correctly printing its output to the *console* before you save the output to a file.

You may assume that the input file exists, is readable, and contains valid input. (In other words, you should not re-prompt for input or output file names.) You may assume that each sequence's number of nucleotides (without dashes) will be a multiple of 3, although the nucleotides on a line might be in either uppercase or lowercase or a combination. Your program should overwrite any existing data in the output file.

Style Guidelines:

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.

We will grade your function structure strictly on this assignment. Use at least **four nontrivial functions** besides `main`. These functions should use parameters and returns, including lists, as appropriate. The functions should be well-structured and avoid redundancy. No one function should do too large a share of the overall task. You may not nest these functions inside each other or in main.

In particular, we **require that you have the following particular function** in your program:

- A function to **print all file output** for a given potential protein (nucleotides, counts, %, is it a protein, etc.)

In other words, all output to the file should be done through one function called on each nucleotide sequence from the input. Your other functions should do the computations to gather information to be passed to this output function.

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 character of an input line. Also avoid "chaining," when many functions call each other without ever returning to `main`.

We will also check strictly for redundancy on this assignment. 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.

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.). Also carefully consider how lists should be passed as parameters and/or returned from functions as you are decomposing your program. Recall that lists use *reference semantics* when passed as parameters, meaning that a list passed to a function can be modified by that function and the changes will be seen by the caller.

You are limited to features in lectures 1 - 20. Follow past style guidelines such as indentation, names, variables, line lengths, and comments (at the beginning of your program, on each function, and on complex sections of code). You may no hav3e any global variables.

Additional Input Files (Optional):

If you would like to generate additional input files to test your program, you can create them from actual NCBI genetic data. The following web site has many data files that contain complete genomes for bacterial organisms:

ftp://ftp.ncbi.nih.gov/genomes/Bacteria/

The site contains many directories with names of organisms. After entering a directory, you can find and save a genome file (a file whose name ends with .fna) and a protein table (a file whose name ends with .ptt). On the course web site we will provide you with a program to convert these .fna and .ptt files into input files suitable for your homework.