Homework 5:
Double Helix All the Way

Due date: March 21 by 11:59pm

Overview

In this assignment you will write a program reads a DNA sequence from the user and prints information about a gene expressed in the sequence. You will submit your solution to Zybooks Lab 7.8, and if the test cases on that lab succeed, you can either show your TA during a Zoom lab period / office hour (strongly preferred) or upload it to Dropbox if there are no Zoom sessions at the moment.

DNA Sequences

The cells of all (known) living organisms are built, grow, reproduce, and function based on instructions carried in that organism's **DNA**. A DNA molecule is comprised of two **strands** of **nucleotides** arranged in "double-helix" structure; each nucleotide is one of four different molecules called *cytosine* (C), *guanine* (G), *adenine* (A), or *thymine* (T). In the field of biogenetics, we represent DNA as a string of nucleotide characters: the string `CAG`, for example, would be the DNA sequence of cytosine, adenine, then guanine.

Complementary Sequences:

The two strands of a DNA molecule are **complements** of each other. To find the complement of a particular DNA sequence, we replace each nucleotide with its "opposite": C and G are opposites, and A and T are opposites. For example, the complement of `CAG` is `GTC` – `C` becomes `G`, `A` becomes `T`, and `G` becomes `C`. Each complementary **base pair** is "attached" by a hydrogen bond, giving the "ladder" shape of a DNA double helix. We can approximate that shape in text by connecting two DNA strand sequences with the *vertical pipe* character, `|`, e.g.

```
CAG
|||
GTC
```

which shows each nucleotide in the "upper" strand and its complement in the lower strand. (See Figure 1 below.)

When analyzing DNA sequences, a scientist does not need to know **both** strands of the DNA sequence; it is enough to know the first strand, because the second strand can always be computed by finding the first strand's complement.

Genes:

The genetic information in an organism's DNA can be broken into **genes**, each of which *roughly* describes how to construct a particular protein molecule. That information is encoded by the sequence of nucleotides in the gene's DNA sequence. A single strand of DNA contains numerous invididividual genes depending on the complexity of the organism; the human **genome** (set of all genes in human DNA) consists of *at least* 46,831 genes involving over 6 million base pairs of nucleotides!

The individual genes in a DNA sequence can be easily identified: **every gene always starts** with the nucleotide sequence `ATG`. Thus, given a strand of a DNA sequence, we can identify the beginning of a gene sequence by finding `ATG` in the sequence. All the nucleotides that follow the `ATG`, including `ATG` itself, form a **gene sequence**.

Codons:

Once the start of a gene has been found, we can begin to identify its **codons**. A codon is a group of 3 nucleotides – thus, `ATG` itself is a codon, actually the *first* codon of any gene. From there, the next 3 nucleotides form the second codon, the next 3 form the third codon, etc.

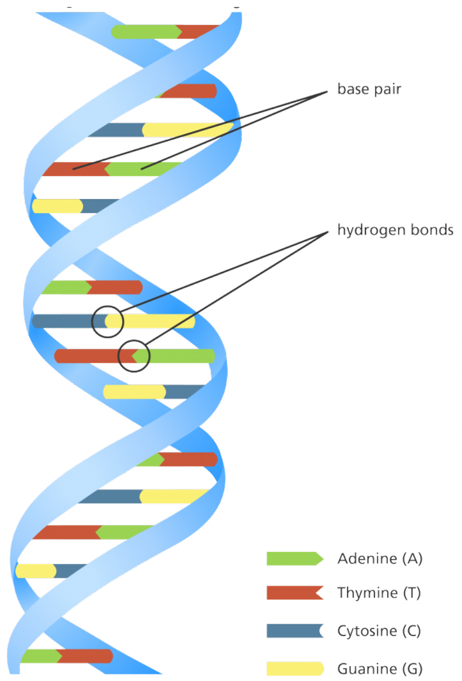

Figure 1: A DNA molecule with two strands of nucleotides bonded into base pairs.

All Together Now:

Given the DNA sequence `CGCCATATAATGCTCGTCCGCGCCCTA`, we identify the gene sequence it contains by finding the first `ATG` starting at the 10th nucleotide/base pair in the sequence. (Count it out!) The gene sequence is the rest of the original sequence starting with `ATG`: `ATGCTCGTCCGCGCCCTA`. The first codon is `ATG`, the second is `CTC`, the third is `GTC`. The **length** of the gene sequence is 18 base pairs (since each nucleotide is secretly paired with its complement in the second strand).

Program Flow

Your Python program will ask the user to input a DNA sequence as a string. You may assume that the string is in all-capital letters, contains (but does not necessarily *start with*) a single gene, and only uses the letters A, T, G, and C. You will use some functions that you write to identify a gene sequence within the whole DNA sequence, and then output information about that sequence, including its **length**, its **complement**, its first three **codons**, and its **upstream** sequence (the portion of the DNA sequence that *preceeds* the first codon).

Functions

You **must** break your program up into the following functions:

1. `find_gene(dna_sequence)`: this function finds the index of the first "`ATG`" that can be found in the given string `dna_sequence`. It then returns a slice of `dna_sequence` starting at the index of the "`ATG`" until the end of the string, and returns that slice.

   Example: `find_gene("CGCCATATAATGCTCGTCCGCGCCCTA")` should return "`ATGCTCGTCCGCGCCCTA`".

2. `find_upstream(dna_sequence)`: this function slices the given string from index 0 up to (but not including) the index of the first "`ATG`". The slice is then returned.

   Example: `find_upstream("CGCCATATAATGCTCGTCCGCGCCCTA")` should return "`CGCCATATA`".

3. `second_codon(gene_sequence)`: given a gene sequence string that starts with "ATG", this function returns the second codon of the gene, e.g., the first group of 3 letters *after* the "ATG" at the beginning of the string.

   Example: `second_codon("ATGCTCGTCCGCGCCCTA")` should return "CTC".

4. `third_codon(gene_sequence)`: similar to `second_codon`, except it returns the next group of 3 letters after the second codon.

   Example: `second_codon("ATGCTCGTCCGCGCCCTA")` should return "GTC".

5. `complementary_nucleotide(nucleotide)`: given a single nucleotide letter, this function returns the complement of that nucleotide.

   Example: `complementary_nucleotide("A")` should return "T".

6. `complementary_sequence(dna_sequence)`: given a DNA sequence, constructs and returns the complement of that sequence, by repeatedly calling `complementary_nucleotide` for each character in the DNA sequence and appending the results to form a single string.

   Example: `complementary_sequence("ATGCTCGTCCGCGCCCTA")` should return "TACGAGCAGGCGCGGGAT".

7. "main" block using an `if` statement: drive the application. Ask the user to input a DNA sequence, then use your functions to compute and output the following, formatted as in the Example Output below:
   (a) the **original DNA sequence** entered by the user.
   (b) the **location** of where the **gene sequence** begins, that is, where the ATG codon occurs in the DNA sequence. To a biologist, the first nucleotide is at location **1**, and that is what we will use for locations. (Not at location 0, as Python would expect.)
   (c) the **second codon** sequence and its location.
   (d) the **third codon** sequence and its location.
   (e) the **upstream sequence** and its **length**. The length is printed as an integer number of "base pairs" (bp).
   (f) the **gene sequence** and its length.
   (g) the gene sequence printed as a **double helix** with its **complementary sequence**: first print the "+ Strand" (the gene sequence itself), then a line of `|` characters, then the "- Strand" (the complement of the gene sequence).

Restrictions:

Your main block is the **only** block of code that can use `print` or `input` statements.

Hints

- Python can multiply a string by an integer; the result is the string repeated that many times, e.g., `"|" * 4` is `"||||"`.
- Your functions will be very short if you read and understand Section 7.3 in Zybooks.

Example Output

User input is in *italics*.

```
Please enter a DNA genetic sequence: CGCCATATAATGCTCGTCCGCGCCCTA
```

```
Original sequence: CGCCATATAATGCTCGTCCGCGCCCTA

ATG codon at bp 10
    followed by CTC at bp 13
    followed by GTC at bp 16

Upstream sequence: CGCCATATA
Upstream length:   9 bp

Gene sequence: ATGCTCGTCCGCGCCCTA
Gene length:   18 bp
[+ Strand]: ATGCTCGTCCGCGCCCTA
            ||||||||||||||||||
[- Strand]: TACGAGCAGGCGCGGGAT
```

Turning in the Assignment

As a reminder, you **must** submit your solution to Zybooks Lab 7.8 and pass all the test cases, **and then** show your accepted solution to the TA or submit it to Dropbox.