Learn to Program: The Fundamentals // Assignment 2

Preface

Required Preparation

This handout assumes that you have watched all the week 4 videos and also done the week 4 exercise. If you read this handout before you've done all of that, please come back and re-read it after you've passed the week 4 exercise.

A2 Problem Domain: Deoxyribonucleic Acid (DNA)

The problem domain for A2 is Deoxyribonucleic Acid (DNA), the double-stranded molecule that encodes genetic information for living organisms. DNA is made up of four kinds of *nucleotides*, which are molecules that bond together to form DNA sequences.

The four nucleotides are adenine (A), guanine (G), cytosine (C), and thymine (T). Each strand of DNA is a sequence of nucleotides, for example AGCTAC. In a program, we will use a string representation of this, `"AGCTAC"`.

DNA has 2 strands in a double helix. The nucleotides in one strand are bonded to the nucleotides in the other strand. A and T can be bonded together, and thus *complement* each other; similarly, C and G are complements of each other.

You can see a picture of this on the Wikipedia page for DNA. The two strands in DNA are *complementary* because each nucleotide in one strand is bonded with its complement in the other strand. Thus, given the DNA sequence ACGTACG, its complementary strand is TGCATGC.

Terminology in this handout

A *DNA sequence* is a sequence of nucleotides, such as TCATGT.

What to do

Step 1: Download the starter code

In this assignment, we are providing *starter code* that contains some function headers and docstrings.

Download the starter code: a2.py.

Step 2: Write functions `get_length`, `is_longer`, `count_nucleotides`, and `contains_sequence`. (Be sure to test your code with empty strings where

appropriate.)

For these functions, you may use built-in functions, `str` operations (for example: `in`, `+`, indexing), and `str` methods.

Function name: (Parameter types) -> Return type	Description
`get_length: (str) -> int`	The parameter is a DNA sequence. Return the length of that sequence.
`is_longer: (str, str) -> bool`	The two parameters are DNA sequences. Return `True` if and only if the first DNA sequence is longer than the second DNA sequence. (If they are the same length, return `False`.)
`count_nucleotides: (str, str) -> int`	The first parameter is a DNA sequence and the second parameter is a nucleotide (`'A'`, `'T'`, `'C'` or `'G'`). Return the number of times the nucleotide occurs in the DNA sequence.
`contains_sequence: (str, str) -> bool`	The two parameters are DNA sequences. Return `True` if and only if the first DNA sequence contains the second DNA sequence.

Once you have finished writing these functions, in IDLE, choose Run -> Run Module. In the shell, test each function by running some example function calls. You can also submit your assignment at this point to see whether you're on the right track: remember, you can submit once every hour up until the deadline. If the example calls return the expected results and you pass all the tests when you submit, move on to Step 3. Otherwise, modify your code and repeat the tests.

Step 3: Write functions `is_valid_sequence` and `insert_sequence`.

There is no starter code for these functions. Use the design recipe to complete them. We have given you some suggestions for examples to try, but you should come up with more on your own based on the descriptions.

For these functions, you may use built-in functions and `str` operations (for example: `in`, `+`, indexing).

Do not use `str` methods.

Function name: (Parameter types) -> Return type	Description
`is_valid_sequence: (str) -> bool`	The parameter is a potential DNA sequence. Return `True` if and only if the DNA sequence is valid (that is, it contains no characters other than `'A'`, `'T'`, `'C'` and `'G'`). There are at least 2 ways to approach this. One way is to count the number of characters that are not nucleotides and then at the end check whether there were more than zero. Another way is to use a Boolean variable that represents whether you have found a non-nucleotide character; it would start off as `True` and would be set to `False` if you found something that wasn't an `'A'`, `'T'`, `'C'` or `'G'`.

	You should construct examples that contain only `'A'`s, `'T'`s, `'C'`s and `'G'`s, and you should also create examples that contain other characters. A string is not a valid DNA sequence if it contains lowercase letters.
`insert_sequence: (str, str, int) -> str`	The first two parameters are DNA sequences and the third parameter is an index. Return the DNA sequence obtained by inserting the second DNA sequence into the first DNA sequence at the given index. (You can assume that the index is valid.) For example, If you call this function with arguments `'CCGG'`, `'AT'`, and `2`, then it should return `'CCATGG'`. When coming up with more examples, think about where the second DNA sequence might be inserted: what are the extremes?

Once you have finished writing these functions, in IDLE, choose Run -> Run Module. In the shell, test your function by running some example function calls. You can also submit your assignment to see whether you're on the right track: remember, you can submit once every hour up until the deadline. If the example calls return the expected results and you pass all the tests when you submit, move on to Step 4. Otherwise, modify your code and repeat the tests.

Step 4: Write functions `get_complement` and `get_complementary_sequence`.

There is no starter code for these functions. Use the design recipe to complete them. We have given you some suggestions for examples to try, but you should come up with more on your own based on the descriptions.

For these functions, you may use built-in functions and `str` operations (for example: `in`, `+`, indexing).

Do not use `str` methods.

Function name: (Parameter types) -> Return type	Description
`get_complement: (str) -> str`	The first parameter is a nucleotide (`'A'`, `'T'`, `'C'` or `'G'`). Return the nucleotide's complement. We have intentionally not given you any examples for this function. The Problem Domain section explains what a nucleotide is and what a complement is.
`get_complementary_sequence: (str) -> str`	The parameter is a DNA sequence. Return the DNA sequence that is complementary to the given DNA sequence. For exmaple, if you call this function with `'AT'` as the argument, it should return `'TA'`.

Once you have finished writing these functions, in IDLE, choose Run -> Run Module. In the shell, test your function by running some example function calls. You can also submit your assignment to see whether you're on the right track: remember, you can submit once every hour up until the deadline. If the example calls return the expected results and you pass all the tests when you submit, move on to Step 4. Otherwise, modify your code and repeat the tests.

Step 5: Submit your work

Go to the Assignments page and click the appropriate Submit button. Choose your completed **a2.py** file. It should be marked within a few minutes. You can view your results by clicking on the View button in the Feedback column.

You can submit **a2.py** once every hour. Notice that this means that you can submit a lot of times before the due date if you start early. You can even submit before you've finished all of the functions. We will take the highest score out of all of your submissions.