Project 2:
You Were so Preoccupied with Whether You *Could*, You Didn't Stop to Think if You *Should*[1]

Due date: Milestone 1 submitted to Zybooks by 11:59PM April 3. Milestone 2 submitted to DropBox on BeachBoard by 11:59 PM April 12.

Overview

In this project, you will develop an application that allows a geneticist to compare two strands of DNA for similarity.

DNA Similarity

Scientists measure how closely related two species are by looking at the DNA sequences for key proteins and seeing how similar or dissimilar they are. The process of **sequence alignment** arranges two or more DNA sequences so that corresponding nucleotides in the sequences can be compared; the more nucleotides that match between the sequences, the "closer" we say the two sequences are evolutionarily.

Consider the two strings of DNA below, where matching nucleotides are underlined:

Species 1: `AATAACGAAA`
Species 2: `AAAACGAAAA`

We would say that these two DNA sequences have **6** matching positions and **4** mismatches, perhaps suggesting that the two species are not closely related. Every corresponding position that are equal is a **match**, and all others are **mismatches**.

Of course, DNA changes over time with reproduction and evolution, and so even two very similar species may have mis-matched DNA sequences due to the random insertion, removal, or mutation of one nucleotide. If a scientist supposes that these two species are more closely related than it appears, they can change the **alignment** of the two DNA sequences by inserting an **indel**, which is a *placeholder* that represents some missing nucleotide, and is **always counted as a mismatch**. Playing with alignments can reveal that two species are actually very closely related despite initial appearances.

Reconsider the two species from before. A scientist places an indel (shown as a dash (`-`)) at the end of Species 1, and at character position **3** (using the biology standard of starting at position 1, not the computing standard of starting at 0) in Species 2. Check out the alignment now:

Species 1: `AATAACGAAA-`
Species 2: `AA-AACGAAAA`

Only two mismatches in the sequences! Our scientist could use this as evidence that a single change occurred in each species' DNA sequence for this protein, and that change accounts for the differences in the expression of that protein in each species.

While complex algorithms exist to do sequence alignment, it is also useful to allow a researcher to perform alignments by hand.

Positions vs. indexes:

In this document, the words "position" and "index" both refer to one particular spot in a string; when "index" is used, it is implied to be **0-based** (first character is index 0), whereas when "position" is used, the

[1]This assignment is based on the work of Bill Punch and Richard Enbody at Michigan State University. CC BY-SA.

implication is **1-based** (first character is at position 1).

Overall Goal

The overal goal for this project is to allow a user to enter two DNA sequence strings. The program will then enter a "main menu" loop where the user will be given many options to manipulate the two strings. One option will add an indel to a string, another will remove it. A third option will print a *similarity score* for the two strings, summarizing how many positions were matching. A fourth option will *suggest* where to add an indel to maximize the number of matches.

As in the previous project, you will develop your solution in two milestones. The menu itself will be part of Milestone 2; in Milestone 1 you will work on gathering input and scoring the similarity of those inputs. You will break your solution into functions defined below, and test cases will make sure you follow the requirements of each function, as well as the overall application itself.

Milestone 1

The first milestone **will not be demoed** to your TA; all you have to do is submit to Zybooks **and pass all the test cases** to get credit, which is worth 40% of the overall project grade. The second milestone will be submitted to DropBox on BeachBoard; the remaining 60% of the overall grade will be based on private test cases and a rubric that will be evaluated by a grader.

For the first milestone, write a Python program that performs the following:

1. Prompt the user to input two DNA sequence strings. You can assume the sequences consist only of the letters `A`, `T`, `C`, and `G`, but they **do not** have to be the same length.
2. If one string is shorter than the other, the shorter string needs to be "padded" with indels at the end, so the strings end up the same length. Do so, and include the indels whenever you would print the strings.
3. Score the similarity of the two strings.
   (a) Print each sequence on its own line, **except** that any matching positions in the sequences are printed in **lowercase**, and any mismatching positions are printed in uppercase.
   (b) Then calculate and print the **number of matching positions**, the **number of mismatching positions**, and the overall **match percentage**. Print the percentage rounded to 1 decimal point (`:.1f`).
4. Prompt the user to input a position in the first string, where position 1 is the first character (*index 0*). Insert an indel into the first string at that position, and then pad the second string with one additional indel so their lengths match once again.
5. Score the two modified strings and print as in step 4.
6. End the program.

To implement this program, you **must** write and use the following functions:

- `pad_with_indels(sequence, num)`: takes the given string `sequence` and adds `num` copies of the indel character `-` to the end of the string, then returns the new string.

  Example: `pad_with_indels("ABC", 2)` returns "`ABC--`".

- `insert_indel(sequence, index)`: given an **index** starting from **0**, takes the string `sequence` and returns a new string where an indel has been added to the sequence at the given index, "pushing" all the remaining characters one index to the right. If `index` is equal to the length of the string, an indel is inserted at the end of the sequence.

  Example: `insert_indel("ABC", 2)`returns "`AB-C`". `insert_indel("ABC", 3)` returns "`ABC-`".

  Note: the user will enter a **position** which starts from 1; you must take their input and translate it to an index starting from 0.

- `count_matches(sequence1, sequence2)`: given two strings of the same length, counts and returns the number of indexes in the strings that are equal, **except** that an indel is **never** equal to anything.

  Example: `count_matches("ABC", "ACC")` returns 2. `count_matches("A-C-", "ABC-")` returns 2.

- "main" block: this is where you will drive the application as described in the 7 steps above. All input will be gathered in the main, and all output will be done here as well. (**No input or printing in any other function.**) You may assume that all inputs are valid, both their type and their ranges, e.g., an indel position for step #5 will always be an integer between 1 and the length of the string plus one.

Your program must match the spelling, grammar, and formatting of the Example Output below **exactly**. The test cases will reject submissions that do not.

Milestone 2

Milestone 2 will expand upon the basic program built in Milestone 1 to include a menu for repeatedly inserting and removing indels. You will also write a function to suggest an indel position for maximizing matches. Your program should be programmed to:

1. Prompt the user to input two DNA sequence strings, as in Milestone 1. Pad the shorter string with indels.
2. Print both sequences, as in Milestone 1.
3. Print a menu to the screen (see Example Output).
4. Read the user's menu choice as input. **Validate** this input as being a valid menu option, and repeatedly ask for a menu option until it is valid.
5. Execute the appropriate menu option depending on the user's choice:
   (a) **Insert an indel**:
      i. Ask the user to select Sequence 1 or Sequence 2. Read their integer input and **validate** it as being the value `1` or `2`.
      ii. Ask the user to input a position within their selected sequence. Read their integer input and **validate** it as being between 1 and the length of the selected sequence **plus one** inclusive.
      iii. Insert an indel into the sequence using `insert_indel`. The returned value replaces the original sequence string from now on.

   (b) **Remove an indel**:
      i. Ask the user to select Sequence 1 or Sequence 2. Read their integer input and **validate** it as being the value `1` or `2`.
      ii. Ask the user to input a position within their selected sequence. Read their integer input and **validate** that it is both *between 1 and the length of the sequence inclusive*, **and** *the character at that position is actually an indel.*
      iii. Remove the indel from the identified position. The returned value replaces the original sequence string from now on.
      iv. The lengths of the two strings is now off by 1. **You may assume that the other string has an indel in the last position**; remove it so the lengths match again.

   (c) **Score similarity**:
      i. Compute and print the similarity between the two sequences, exactly as in Milestone 1.
      ii. (Do not print the strings themselves, only the similarity numbers.)

   (d) **Suggest indel**:

   i. Ask the user to select Sequence 1 or Sequence 2. Read their integer input and **validate** it as being the value `1` or `2`.
   ii. Compute the **optimal position** at which an indel should be inserted into the selected sequence in order to **maximize** the number of matches between the resulting sequences.
      A. We will "**brute force**" this computation: in a loop, *try every single position in the given sequence*. For each position, insert an indel into the original sequence at that position, then count the number of matches. **Accumulate** this answer to find the position with the maximum number of matches. If two positions result in the same number of matches, use the **first** of the two.
   iii. Print the optimal position for an indel.
   iv. Compute and print the similarity using your proposed position, exactly as in menu option (c).

(e) **Quit**: terminate the program.

6. **Repeat** steps 2 through 5 until the user chooses to quit.

To implement this program, you **must** write and use the following functions, *in addition to* the Milestone 1 functions which **will not change**:

- `print_menu()`: prints the menu to the screen.
- `get_menu_choice()`: asks the user to input a menu option choice, and validates that input as being a valid menu option. Returns the validated menu option.
- `get_sequence_number()`: asks the user to input a sequence number (1 or 2), and validates that input until it is correct. Returns the validated sequence number. Used by the main any time the user needs to select a sequence number.
- `get_insert_position(sequence)`: given a sequence, asks the user to input a position where an indel can be inserted within that sequence (from 1 up to and including the length of the sequence), and validates that input until it is correct. Returns the validated position number.
- `get_remove_position(sequence)`: given a sequence, asks the user to input a position within that sequence *where an indel can be found*, and validates that input until it is correct. Returns the validated position number. A position where there is no indel is *not a valid position*.
- `remove_indel(sequence, index)`: given a sequence and an index of an indel in that sequence, removes the indel at the given index and returns the new string.
- `find_optimal_indel_position(sequence, other_sequence)`: given two sequences, where the first parameter is the sequence to add an indel to, calculate and return the **position** in the first sequence where an indel should be added to maximize the similarity between the two sequences.

The **only** functions that are allowed to print (besides the main) are `print_menu` and the three `get_X` functions.

Non-functional requirements:

As in Project 1, your Milestone 2 submission will be graded both functionally and non-functionally. You must follow these guidelines to receive full non-functional credit:

1. All the variables in your program must be given meaningful names. A *meaningful name* makes it clear what the variable's purpose is to someone who is familiar with the problem.
2. The only functions that can use the `print` function are the main, any function named `print_X`, and any function named `get_Y`. Only the main and `get_Y` functions can use `input`.
3. You **cannot** index, slice, concatenate, or call methods (`find`, `lower`/`upper`, `replace`, etc.) **any strings** *in the program's main.* Whenever you need to manipulate a string, you *must* call one of *your* functions to accomplish that goal. However, you *can* use the `len` function anywhere you like.

4. When determining optimal indel position, your code must not call (or otherwise use code similar to) the `count_matches` function **more than** $n + 1$ **times**, where n is the length of each sequence.
5. **Every single function** must have a 1-to-3-sentence comment that preceeds the function `def` and describes the purpose of the function. Your comments must be written as full English sentences with proper spelling and grammar. You should identify the *big picture* purpose of each function, **and include** a *big picture* description of the function's parameters and what the function returns (if anything).

   Example of a bad comment:
   ```
   # gets the menu choice
   def get_menu_choice()
   ```

   This comment doesn't describe what the function is doing, and is completely redundant with the function's name.

   Example of a bad comment:
   ```
   # Puts a dash in a string.
   def insert_indel(sequence, index)
   ```

   This comment is far too vague. Where does the dash go? What are the parameters for? What gets returned? 0 out of 10, would not read again.

   Example of a good comment:
   ```
   # Takes a DNA sequence string and adds a dash at the indicated index.  Returns
   # the new string with the dash inserted.
   def insert_indel(sequence, index)
   ```

   We indicate the purpose of the method, describe its parameters, and what to expect as a return value. A programmer can read this brief description and have a general understanding of what the function is for.
6. Inside of the `find_optimal_indel_position` function, you must also write a comment that describes how your algorithm *works*. This comment should be 3-4 sentences and independent of the comment mandated by the previous requirement. You should explain each general step of the algorithm in plain English, using the vocabulary of the problem domain ("indels", "sequences", etc.). It should be clear to a reader of your comment *why* you have written each line of the function, and why the overall approach is correct.

Example Output

User input is in *underlined italics*. These do not cover every possible test case. It is up to you to thoroughly test your program before submitting it.

Milestone 1:

```
Please enter DNA Sequence 1: AAAACGAAAA
Please enter DNA Sequence 2: AATAACGAAA

Sequence 1: aaAaCGAaaa
Sequence 2: aaTaACGaaa

Similarity: 6 matches, 4 mismatches. 60.0% match rate.

Please enter an indel location for Sequence 1: 3
```

```
Sequence 1: aa-aacgaaaA
Sequence 2: aaTaacgaaa-

Similarity: 9 matches, 2 mismatches. 81.8% match rate.
```

Milestone 2:

```
Please enter DNA Sequence 1: ATGCGAAAT
Please enter DNA Sequence 2: AATAACGAAA

Sequence 1: aTGCGAAaT-
Sequence 2: aATAACGaAA

Main Menu
1. Insert an indel
2. Remove an indel
3. Score similarity
4. Suggest indel
5. Quit

Please choose an option: 3

Similarity: 2 matches, 8 mismatches. 20.0% match rate.

Sequence 1: aTGCGAAaT-
Sequence 2: aATAACGaAA

[Main Menu prints again]
Please choose an option: 1

Sequence 1 or 2? -1
Sequence 1 or 2? 1
Please choose a position: -1
Please choose a position: 2

Sequence 1: a-tGCGAaaT-
Sequence 2: aAtAACGaaA-

[Main Menu prints again]
Please choose an option: 2

Sequence 1 or 2? 2
Please choose a position: 2
Please choose a position: 11

Sequence 1: a-tGCGAaaT
Sequence 2: aAtAACGaaA

[Main Menu prints again]
Please choose an option: 3

Similarity: 4 matches, 6 mismatches. 40.0% match rate.

Sequence 1: a-tGCGAaaT
Sequence 2: aAtAACGaaA
```

```
[Main Menu prints again]
Please choose an option: 4

Sequence 1 or 2? 1
Insert an indel into Sequence 1 at position 4.
Similarity: 7 matches, 4 mismatches. 63.6% match rate.

Sequence 1: a-tGCGAaaT
Sequence 2: aAtAACGaaA

[Main Menu prints again]
Please choose an option: 1
Sequence 1 or 2? 1
Please choose a position: 4

Sequence 1: a-t-GcgaaaT
Sequence 2: aAtAAcgaaa-

[Main Menu prints again]
Please choose an option: 3

Similarity: 7 matches, 4 mismatches. 63.6% match rate.

Sequence 1: a-t-GcgaaaT
Sequence 2: aAtAAcgaaa-

[Main Menu prints again]
Please choose an option: 5
```