# Comp125 - Homework 3

## Author: Ayca Tuzmen, Emirhan Erel, Ozgun Ozan Nacitarhan

In the spirit of learning how to code, in this assignment we're going to use code to help others learn! Specifically, we'll practice using control flow, variables, and functions to look at patterns in biology and math. We'll also use the **input()** built-in to get user input through the console for each problem. Each of the math and biology sections will include both a problem that we can solve as humans and also a problem that is made much easier when we have programming on our side.

**Important Note:**
For each problem, we give you specific guidelines on how to begin decomposing your solution. While you can (and should) add additional functions for decomposition, **you should not change any of the function names or parameter requirements that we already provide to you in the starter code.** Since we include doctests for these pre-decomposed functions, editing the function headers can cause existing tests to fail.

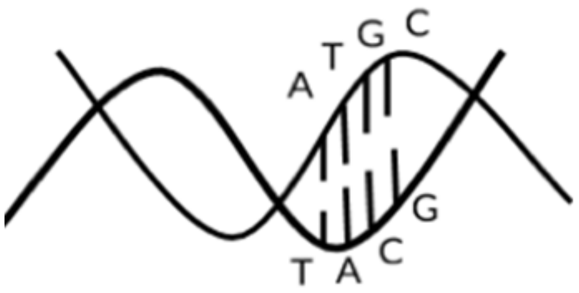
As requirement for this assignment, you must adhere to the following testing guidelines:
• For each pre-decomposed function that we provide you, you must write **at least 3 additional doctests per function**.
- Any new functions that you write should have **at least 3 doctests**.

All tests for a given function should cover completely separate cases, and we encourage you to consider both common use cases and edge cases as you write your doctests. Using good testing practices and thinking through possible inputs/outputs for your code will help increase your likelihood of getting full functionality credit for your work!

---

## Question 1 - Patterns in biology

In biology, you might have learned that the fundamental unit of a nucleic acid is a nucleotide, or base. The four possible bases for DNA are Guanine (G), Cytosine (C), Adenine (A), and Thymine (T) whereas those for RNA are Guanine (G), Cytosine (C), Adenine (A), and Uracil(U). These nucleotides form "base pairs" that make up complementary strands of DNA and RNA. As shown in below figures, in both types of nucleic acids G pairs with C. However, in DNA, A pairs with T and in RNA (during translation), A pairs with U.



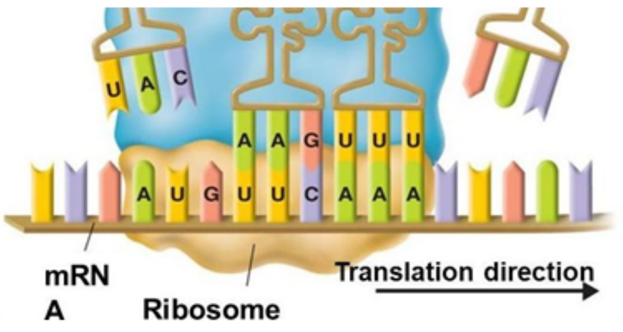


**Figure 3**: Different pairs of nucleotides in two types of nucleic acids. (DNA on the left and RNA on the right).

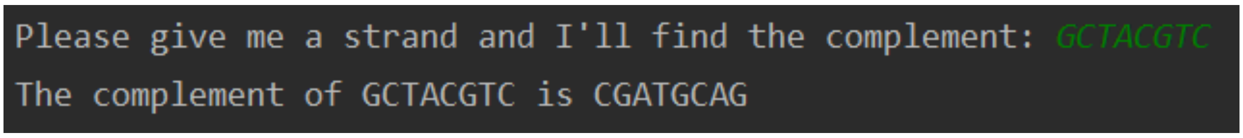
In **complement.py**, write a program that

1) Prompts the user for one strand and returns the complement of the given strand

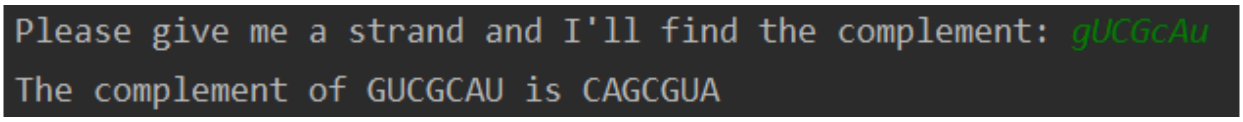
2) Determines whether it is an RNA or a DNA strand by the nucleotides present in the given strand.

3) If the user enters a strand that contains both U and T bases, your program must ask for an appropriate strand until a valid sequence is given.

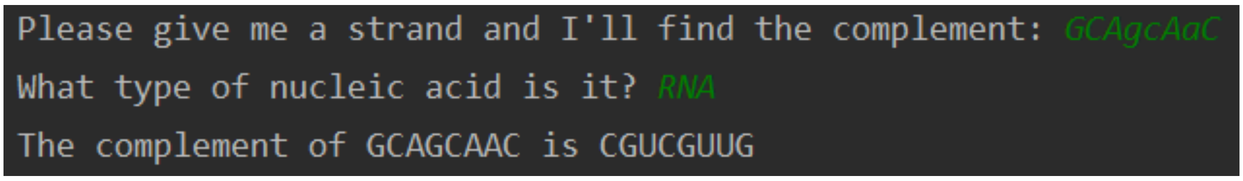
4) If the user enters a strand that DOES NOT contain U or T, the program must ask the user which type of nucleic acid this is and act accordingly in the next step.



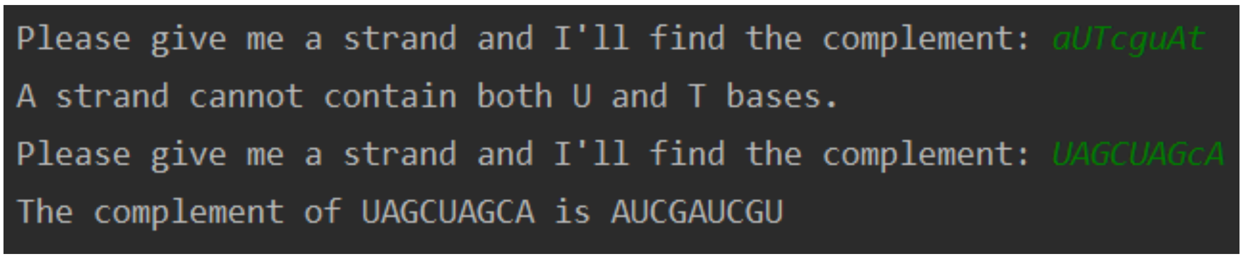
```
Please give me a strand and I'll find the complement: GCTACGTC
The complement of GCTACGTC is CGATGCAG
```



```
Please give me a strand and I'll find the complement: gUCGcAu
The complement of GUCGCAU is CAGCGUA
```



```
Please give me a strand and I'll find the complement: GCAgcAaC
What type of nucleic acid is it? RNA
The complement of GCAGCAAC is CGUCGUUG
```



```
Please give me a strand and I'll find the complement: aUTcguAt
A strand cannot contain both U and T bases.
Please give me a strand and I'll find the complement: UAGCUAGcA
The complement of UAGCUAGCA is AUCGAUCGU
```

**Figure 4** : Several sample outputs of your program in **complement.py** .

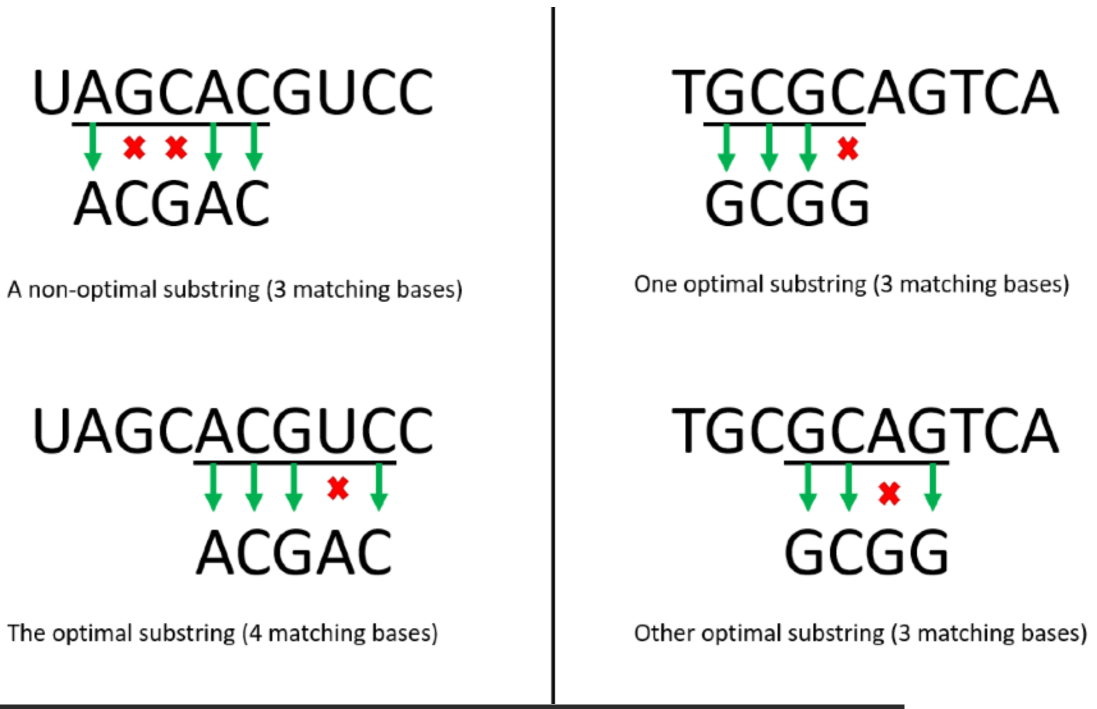
Your code should be decomposed so that it includes a function called **build_complement()** , which takes in a strand as a string and returns its complement as a string. You should call your function after using **input()** to get the strand from the user and before printing out the complement on the screen. Your function's output should be case-insensitive; that is, **build_complement('ATG')** and **build_complement('aTg')** should return the same result. Your function can assume that all of the base pairs of the input string are valid base pairs – that is, the string consists only of the following characters: **'a', 'A', 'g', 'G', 't', 'T', 'c', 'C', 'u', 'U'**

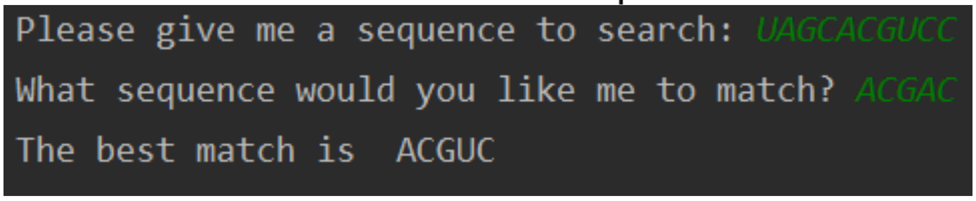
---

## Question 2

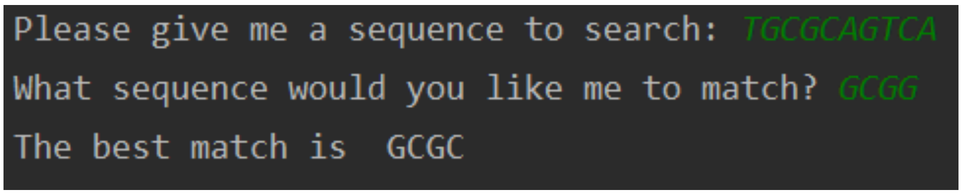
As humans, we can easily look at short segments of a strand and generate the complementary bases. But what about identifying the similarity between two different strands of a nucleic acid? A common use of programming in biology is to determine segments of nucleic acids that have the highest similarity when aligning nucleotides. In particular, this problem will look at how we might measure **homology** or the alignment of matching bases.

In **similarity.py** , write a program that asks the user for a strand to search (we'll refer to this as the "search strand") and a strand to look for within the search strand (we'll refer to this as the "match strand"). Both strands are represented as strings. Your program should return the sub-strand of the search strand that has the most matching nucleotides with the match strand. We define "matching nucleotides" as when the two strands have the same nucleotide at the same index (i.e., if the second nucleotide in both strands is A, then that counts as one matching nucleotide). See Figure 5 for diagrams of how we measure homology and an example run off the program.


UAGCACGUCC
ACGAC
A non-optimal substring (3 matching bases)
TGCGCAGTCA
GCGG
One optimal substring (3 matching bases)
UAGCACGUCC
ACGAC
The optimal substring (4 matching bases)
TGCGCAGTCA
GCGG
Other optimal substring (3 matching bases)

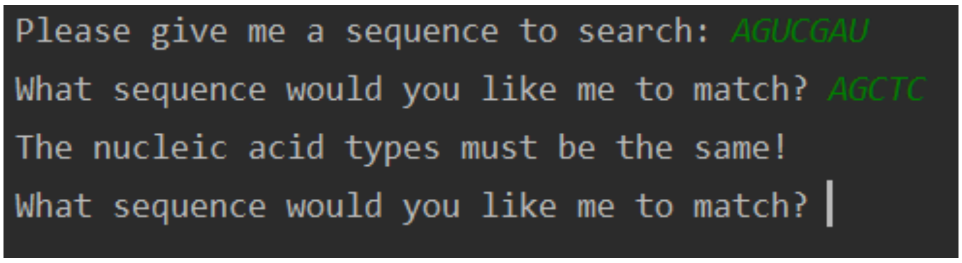


```
Please give me a sequence to search: UAGCACGUCC
What sequence would you like me to match? ACGAC
The best match is  ACGUC
```



```
Please give me a sequence to search: TGCGCAGTCA
What sequence would you like me to match? GCGG
The best match is  GCGC
```

If there are multiple "best matches" possible, your program can return any of them.



```
Please give me a sequence to search: AGUCGAU
What sequence would you like me to match? AGCTC
The nucleic acid types must be the same!
What sequence would you like me to match? |
```

If the types of nucleic acids are different which means one of them contains T and the other contains U, the program must ask for another "sequence to match!".

**Figure 5** : We measure homology by aligning the match strand with a substring of the search strand. For the search strand "UAGCACGUCC" and the match strand "ACGAC," the best match is "ACGUC" since it has four matching bases and running **similarity.py** would produce the output shown (previous page).

You should decompose your program in the following way:

- Write one function called **search_sequence()** that takes in as parameters the match strand and an equal length substring of the search strand and returns the number of nucleotides that match between the two.
- Write a second function called **find_best_substrand()** that loops over the search strand, generates substrings of equal length to the match strand, and

calls your first function to find the sub-strand with the most matching nucleotides.

You can assume that the user will only input strands that include A, T, G, C or U and that the strand being searched will be longer than the strand you are trying to match. Note that your program should be **case insensitive** (it should work regardless of whether the user types in capital or lowercase letters for each strand).

Measuring homology has applications in medicine for detecting diseases based on genetic mutations, like sickle cell disease or certain forms of cancer. As the strands that you are comparing become longer, the problem becomes increasingly more difficult. Computers are an essential tool in helping us find solutions!

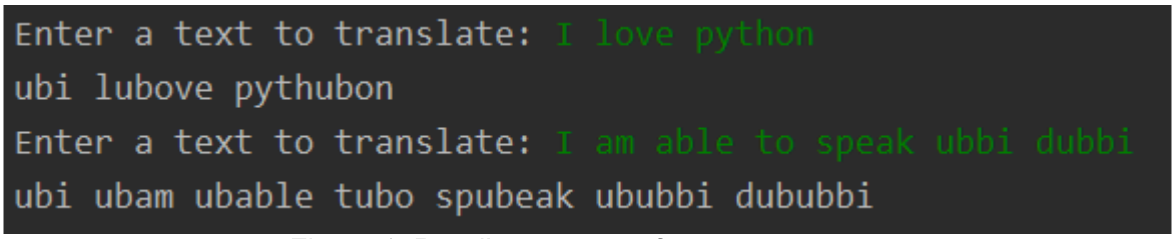
---

## Question 3 - Ubbi Dubbi

Write a Python console program asks the user for a sentence, convert all the words to lowercase, and translate the user's sentence into Ubbi Dubbi words. A Ubbi Dubbi words can be generated using the following rules:

- If a word begins with a vowel letter, add an "ub" sound at the beginning of the word (e.g. "alter" becomes "ubalter")
- If the word begins with a consonant letter, add an "ub" sound before first vowel. (e.g. "cat" becomes "cubat")

Here are example runs:



```
Enter a text to translate: I love python
ubi lubove pythubon
Enter a text to translate: I am able to speak ubbi dubbi
ubi ubam ubable tubo spubeak ububbi dububbi
```

Figure 1: Possible outputs of your program

Important

- You are asked to implement one function in the starter code. 2 doctests are already provided to you, however you are expected to write 3 more doctests to test this function.

## Question 4

Write a Python console program that converts Celsius degrees into Fahrenheit degrees and Fahrenheit degrees into Celsius degrees. The program should randomly choose which conversion will be made. The requirements for this program are:

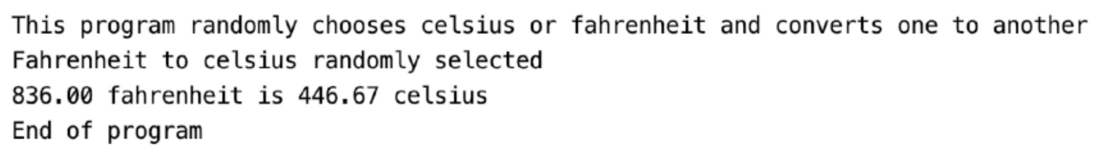
- The program shall randomly select conversion from Celsius to Fahrenheit or from Fahrenheit to Celsius.
- In case of conversion from Celsius to Fahrenheit, the program shall randomly generate a Celsius degree between -273, 1000 (including the numbers).
- In case of conversion from Fahrenheit to Celsius, the program shall randomly generate a Fahrenheit degree between -459, 1832 (including the numbers).
- The program shall convert Celsius into Fahrenheit or convert Fahrenheit to Celsius according to the *celsius_or_fahrenheit()* method using the below formulas:

**Fahrenheit to Celsius:** $^{\circ}C = \frac{5}{9}\left(^{\circ}F - 32\right)$

**Celsius to Fahrenheit:** $^{\circ}F = \frac{9}{5}{}^{\circ}C + 32$

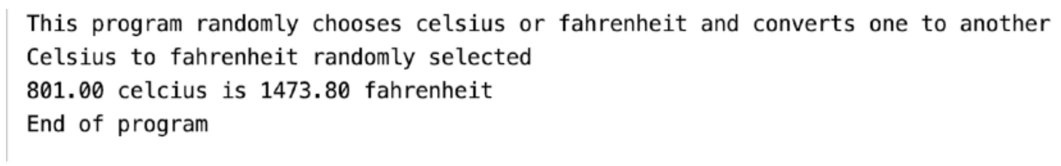
- The program displays a message as shown in sample runs.

Here are example runs:



```
This program randomly chooses celsius or fahrenheit and converts one to another
Fahrenheit to celsius randomly selected
836.00 fahrenheit is 446.67 celsius
End of program
```

Figure 6: First possible output of your program



```
This program randomly chooses celsius or fahrenheit and converts one to another
Celsius to fahrenheit randomly selected
801.00 celcius is 1473.80 fahrenheit
End of program
```

Figure 7: Second possible output of your program

**Important:**

It is important your prompt to the user and display message look exactly the same as the sample output screen. Make sure you display the messages in the same format