Exercises · 01

Fundamentals of Python (Aptamer Stream)

Due April 8th, 2016

For each problem, example input is given as well as the expected output for that input (shown in bold). The estimated difficulty levels range from easier (⋆) to harder (⋆ ⋆ ⋆). Advanced problems (⊕) and problem parts are optional and may require you to read ahead on your own. You may work with others to complete these exercises, but you should try tackling the problems alone first. Additionally, you should type up your programs independently (please don't copy and paste from anyone/anywhere) and be prepared to answer questions about how your program works in case I ask you to explain the logic. This is for your own benefit! For Problem 3, I am providing you with my solution so that this might help you come up with your own solutions for the other problems. Questions 3 and 4 are based on Rosalind exercises.

1 Miles to kilometers ★

Write a program that will allow the user to input a number of miles; the program will then convert that to kilometers. As the description implies, you should use the `input()` and `print()` functions. For convenience, there are about 1.60934 km in 1 mi. Note: to output your result successfully with units, you will probably need to convert your numeric result from `float` to `str` (string) by using the `str()` function with the result as the argument.

```
How many miles? 5.4
That is 8.690436 km.
```

2 Combining DNA sequences ★

Imagine that you receive two DNA sequence parts that should be considered as one sequence. Your job is to *concatenate* the parts into one sequence. Your program should allow the user to input two strings (entered one at a time); the program should then print out the combined string after the user submits his/her input.

```
Enter the first sequence part: ATCGTTAACGTT
Enter the second sequence part: GATCGTACTAGC
ATCGTTAACGTTGATCGTACTAGC
```

Advanced: figure out how to let the user enter both sequence parts in one go (i.e., one part, followed by a space, followed by the other part). You might want to read up on the split() and join() methods. Alternatively, you can read about the replace() method instead for a different way to solve this.

3 Counting DNA nucleotides ★★

For this program, take a DNA sequence and print out the counts of A, G, C, and T (in that order).

```
Enter a DNA sequence: ATCGATGCTTGAGC
3 4 3 4
```

My solution:

```python
seq = input("Enter a DNA sequence: ") # get user input and save it to seq
count_A = 0 # four variables to store the base counts
count_G = 0
count_C = 0
count_T = 0

for base in seq:     # loop through all the bases in the sequence
    if base == "A":  # if we find an A
        count_A += 1 # then update our count of A's
    if base == "G":  # if we find a G
        count_G += 1 # then update our count of G's
    if base == "C":  # etc...
        count_C += 1
    if base == "T":
        count_T += 1

print(count_A, count_G, count_C, count_T) # print all the counts on one line
```

Advanced: write a program that does this using only 2 lines (read up on the count() method for strings). Yes, this program can be written in just two lines in Python!

4 Calculating GC content ★★

Write a program that will allow the user to input four integers; these are the counts of adenine, guanine, cytosine, and thymine found on a DNA molecule (the counts will be input in that order). *GC content* is the percentage of nucleobases on a DNA molecule that are guanine or cytosine. Knowing this, you should be able to have your program print out the GC content of this molecule. Hint: perhaps a for loop and checks for equality (with ==) would be useful.

```
Enter counts of A, G, C, T: 2 3 4 5
The GC content is 50.0%.
```

5 A guessing game ⊕

This is a common introductory programming exercise since it ties together many basic concepts, such as `if` statements and `while` loops. I hope that we will get to `while` loops in class; if not, please read about them on your own. Your program should select a number between 1 and 100 inclusive (you can hardcode this number in your program) that is secret to the user. The user should then input a number between 1 and 100. The program will tell the user if his/her guess was too low or too high (less than or greater than the secret number). The user can then guess again. If the user guesses the number correctly, then the program will output a message indicating that the user has won the game.

```
Guess the secret number.
Input your guess: 75
Too high.
Input your guess: 50
Too low.
Input your guess: 61
You win!
```

Bonus: To make this more interesting on subsequent playthroughs, randomly generate the secret number by `import`ing the `random` module and using the `random.randint()` function.