Exercises · 03

Fundamentals of Python (Aptamer Stream)

Due May 6th, 2016

Each of the four problems on this set is worth $2 + 2n$ Paul Points, where n is the problem number. These are meant to be challenging, and I expect that you will need to do independent research to supplement what we will go over in lecture on April 29th. Topics to look into include dictionaries and sorting algorithms. Three problems are adapted from Rosalind.

1 Bubble sort ★

Sorting is the act of arranging elements of a sequence into a systematic order. Many programming languages have efficient sorting methods built in; you are not allowed to use them for this problem. Your task is to reimplement the well-known method of sorting known as *bubble sort*, mentioned by Barack Obama to Google CEO Eric Schmidt in this video: `https://www.youtube.com/watch?v=k4RRi_ntQc8`.
Your program should take a list of numbers (separated by spaces) and print out the numbers in increasing order. Your program *must* use bubble sort, even though there are far better sorting algorithms in existence; the purpose of this exercise is for you to get practice with going through the logic of a basic sorting algorithm.
Bubble sort repeatedly goes over a list of items and swaps adjacent items depending on how they compare to each other. For numbers, this involves determining which number is the greater of the two (if a is greater than b, swap a and b). You can imagine that as the algorithm passes over the list repeatedly, the largest number in the list will bubble up to the last position (and thus no longer be swapped with any other number), the second-largest number will bubble up to the penultimate position, and so on. Feel free to check out Wikipedia or other sources for more information. You can use a temp variable for the swap step if you like, but this is not necessary if you do both assignments simultaneously (e.g., `a, b = b, a`). Check out this animation if you want to visualize the algorithm: `http://www.cs.armstrong.edu/liang/animation/web/BubbleSort.html`.

```
Enter numbers: 25 7 6 8 34 2 -1
-1 2 6 7 8 25 34
```

2 Transitions and transversions ★★

Recall that there are two types of nitrogenous bases used in DNA: purines and pyrimidines. Adenine and guanine are purines, whereas cytosine and thymine (and uracil in RNA) are pyrimidines.
A point mutation is a mutation that affects a single nucleotide on a DNA sequence. Types of point mutations include substitutions (such as A to C), deletions (such as TAG to TG), and insertions (such as CGA to CGAG). For this exercise, we will only consider single base substituions, of which there are two types: *transitions* and *transversions*. A transition changes a purine to another purine or a pyrimidine to another pyrimidine. A transversion changes a purine to a pyrimidine or a pyrimidine to a purine. Notice that there are twice as many ways for transversions to occur than transitions. Despite this, it has been observed that transitions are more common than transversions.
Your goal is to write a program that will accept two DNA sequences of equal length; one is a mutated copy of the other. Your program should print out the transition/transversion ratio, which is simply the number of transitions that occured divided by the number of transversions that occured. If zero transversions occurred, do not print out a ratio; instead, print out a message stating that zero transitions were found.
In addition to the above, you must print out the locations where mutations have occured in a comma-separated list (the first nucleotide of a sequence is considered location 1 for this problem). If no transitions or transversions are present, do not print out a string mentioning where mutations occurred. You do not have to print out which types of substitutions occurred at each location.
Sample input and output can be found below.

```
Enter sequence 1: GTCATCGATGCTAGTACGATGTCGATGACTGATAGCTAGTAGCTAGTGATGTAGCTAGTGAC
Enter sequence 2: ATCACTGATGCTAATACGATGTAGATGGCTGATAGTTGGTGGCCCGTGGTGTAACTAGTGAT
The transition/transversion ratio is 6.0.
Mutations occured at locations 1, 5, 6, 14, 23, 28, 36, 38, 41, 44, 45, 49, 54,
62.
```

3 Translating RNA into protein ★★★

There are 20 commonly occurring amino acids. These amino acids are the building blocks of almost every protein in an organism; one can chain together the one-letter identifiers for amino acids to create a protein string. Each amino acid in a protein string can be encoded by a corresponding codon on mRNA, which is transcribed from DNA.
Write a program that will take an mRNA sequence and return the protein that would result if it were to be translated by a ribosome. For simplicity, you may assume that translation will always start at the first codon in the sequence. The codon table you should base your program on can be found here: http://rosalind.info/glossary/rna-codon-table/. It is also reproduced on the next page. Note that you should not print out "Stop"; instead, when a stop codon is encountered, no further amino acids should be added to the sequence that is printed. One challenge you might want to take on is to store the codon

table as one string in your code and possibly use the `dict()`, `zip()`, and `split()` functions to create a dictionary mapping codons to amino acids.

```
UUU F     CUU L     AUU I     GUU V
UUC F     CUC L     AUC I     GUC V
UUA L     CUA L     AUA I     GUA V
UUG L     CUG L     AUG M     GUG V
UCU S     CCU P     ACU T     GCU A
UCC S     CCC P     ACC T     GCC A
UCA S     CCA P     ACA T     GCA A
UCG S     CCG P     ACG T     GCG A
UAU Y     CAU H     AAU N     GAU D
UAC Y     CAC H     AAC N     GAC D
UAA Stop  CAA Q     AAA K     GAA E
UAG Stop  CAG Q     AAG K     GAG E
UGU C     CGU R     AGU S     GGU G
UGC C     CGC R     AGC S     GGC G
UGA Stop  CGA R     AGA R     GGA G
UGG W     CGG R     AGG R     GGG G
```

```
Enter the mRNA sequence: AUGCAUAAGAGUGGAGUGCUCAGCCAUUGA
Protein sequence: MHKSGVLSH
```

4 Generating a consensus sequence ⊕

Write a program that will take in four DNA sequences of identical length and return a valid consensus sequence. A consensus sequence, in some sense, can be thought of as the average of the given sequences. A consensus sequence for DNA consists of the most popular bases at each position. See the sample input and output to see what a potential consensus sequence could look like. When writing your program, you may find it convenient to store the four sequences in a list. One possible way to solve this is to use a matrix to store base counts for each position so that you can select the base that shows up the most for each position in the consensus sequence (the `max()` function may be useful). Note that multiple different valid consensus sequences are often possible.

```
Enter sequence 1: GATATAGGTACGCCTACGGTCAGGAACACTACGAGGGCCTCTCATACTAGATTTGCACTA
Enter sequence 2: ACTTGCAAATCGTAGGGACTAGGTCTAAGCGAATTGTAAAGGCCGAAAAGGCTCTGGTAA
Enter sequence 3: TACACACTGGACGTCCCGTGGCTTGCACGCCCCCAGTGCCTGTTTGTTCTCGTGGTTGTT
Enter sequence 4: TTCACGGTATGATCTTGTCGCACTTGAAAGAAGAACTGAGCTGAAAGCCTTCATATGTCT
Consensus seq is: TACACAGTATCGTCTACGCGCAGTAAAAGCAAGAAGTGAACGCATAATAGACTTGTGTTA
```

Have fun!