Summer Research School in Computational Biology and Bioinformatics - coding challenge.

This test is to assess student applicants for the Summer Research School in Computational Biology and Bioinformatics. The test will be conducted in the Julia language (https://julialang.org/).

Please submit your work through the github classroom system. If you found this document, it means you already have a git repository with these questions and a few other files. We can access what you upload to this repository, which is how we will assess your solutions.

Note: by submitting, your solutions (including your name and email) will be stored on GitHub, and you consent to us accessing and processing your submission for the purpose of selecting applicants for the Summer Research School If you object to uploading your solutions, name, or email address to GitHub, please contact the organizers and we can discuss this, and possibly make an alternative arrangement for your submission.

General rules:

- Answer these questions without help.
- Answer as many questions as you can. You do not have to answer them all to submit. Some of these questions are *extremely* difficult, and we do not necessarily expect anyone to solve them all, so please submit whatever you do manage to solve!
- You are allowed to use or adapt code that you find on the internet, but please do not ask for help online, or post these questions online in any form.

Steps:

1. Install the latest version of Julia (https://julialang.org/), along with IJulia and Jupyter Notebooks (instructions here: https://github.com/JuliaLang/IJulia.jl). Check that you can open a Jupyter notebook and run some Julia commands. If you get stuck on any of these installation steps, Google will prove very helpful.
2. Clone or Download the initial challenge repository from GitHub *to your computer.*
3. Open the "FunctionTesting.ipynb" Jupyter notebook *on your computer*, and run the code that will test your solutions. This tests each of the functions in the "functions.jl" file. Initially, you should get messages that all of them fail.
4. For each question, fill in the code for the corresponding function in "functions.jl". You can re-run the tests in "FunctionTesting.ipynb" to see how they perform on a single test case.
5. **Critical:** When you're content with your solutions, or feel like you won't be able to get any further, please update the "functions.jl" in your ***online*** GitHub repository so that it contains your solutions (don't just change the file locally on your computer!). If you are familiar with Git, you can use a "push" command, but you can also simply upload your modified "functions.jl" file using the "Upload files" button through the web browser interface while viewing your repository on GitHub. Note, your repository URL should look like this: https://github.com/MurrellGroup/ki-comp-bio-coding-challenge-2020-*YourGitHubUsername*
Please check that "functions.jl" at this address (but modified for your username) contains the version with your solutions.

Other considerations:

- Avoid putting any print/println statements in the final functions you submit, as these will make the test output more difficult to read.
- For questions you do not solve, just leave the relevant function as is - no need to delete it.
- When we assess you, we will download your repository and run your “functions.jl” against a more comprehensive set of tests, on many different cases, sometimes including timing of the solutions.
- Do not modify "test.jl".
- For the trickier algorithmic problems, optimal solutions are nice, but don’t let that stop you from submitting. We aren’t quite sure how hard these are, nor whether anyone will provide optimal solutions to all of the problems, so please just submit the best solution you can find for each of the problems, or leave that problem blank.

Questions:

Q0) Literally just your name and email (rating: starter):

Make the Q0 function in “functions.jl” return your name and email address, so we can contact you if you are selected.

Q1) GC content (rating: starter):

Count the proportion (as a percent) of bases in a DNA sequence that are either G or C.
Example input:
`“GCATGCACATAGCAGCGAGCTACTACATCGCGGCTAGACTACTGAGCGA”`

Q2) Translation (rating: starter):

In DNA, 3 successive nucleotides forms a “codon” (https://en.wikipedia.org/wiki/Genetic_code), where each codon codes for an Amino Acid. Write a function that takes a DNA string, and returns the corresponding amino acid string.

Q3) A magically annoying coin (rating: intermediate):

You are given a coin that can be flipped up to K times. On the nth flip, the probability of landing heads (H) is `n/K`, and tails (T) is `1-(n/K)`. Write a function that computes the logarithm of the probability of observing a sequence of flips, such as:
`TTTTTTTTTTTTTTTTTHTTTTTHHTTTTTTHTTTHTTHHTTTTHHTHTTHHTHHHHHHHHHHHHTTTHHHTHTHHHTHHTTTHHHHHHTTHHHHHHTHHH`
(What we might ask if you get called for an interview: Do you think that this example sequence above was generated by the magical coin? Why?)

[Continued on next page]

Q4) The biggest product (rating: tricky):
Consider an array of strictly positive numbers, each greater than zero. For any contiguous subsequence of such numbers, you can compute their product. Write a function that, in the shortest compute time, calculates the starting point and ending point of a contiguous subsequence of a strictly positive array that has the largest product.
For example:
`A = [1.6, 0.56, `**`1.3, 1.5, 0.9, 1.5, 2.5, 1.1,`**` 0.46, 0.65]`
The contiguous subsequence with the greatest product is from index 3 to index 8, with a product of 7.239375. On this example, the function should return: (3, 8)
Note: The speed of your implementation, and how it scales to large input arrays, will be considered in the assessment. Will your function be able to run on arrays with 10,000 elements in a reasonable amount of time? How about 1 million or 1 billion elements?

Q5) Data dumpster diving (rating: wat):
The following .csv file contains a symmetric 2D array of pairwise distances between 1722 elements:
`https://drive.google.com/open?id=1MDv1UViwPUoeLJfQ1XH1jvAr-nAzDTTX`
What sort of thing are these elements? What does the element at position 430 (ie. the 430th row or column of the distance matrix) correspond to? How did you figure this out?
Note: This should be answered by returning the answer as text in the relevant function. This will not be included in the automatic assessment, but will be checked by a human.
Hint: Find a way to go from a distance matrix to an embedding of the elements themselves, where you can look at their layout. This might require some Googling.

Q6) Array yeet (rating: difficult):
You have two arrays of numeric values, not necessarily of the same length. Consider a set of deletions in each array, resulting in the arrays being the same length. For every possible set of deletions, you can compute a score, which is the sum of the element-wise squared differences, plus the number of elements you had to delete (across both arrays). Write a function that, in the shortest possible compute time, decides which elements to delete in both arrays to minimize this score. For example, given:
`A = [-5, `***`9`***`, -3, -2, -9, -2, 2, 11, -1, -4, -10, `***`21`***`]`
`B = [-4, -4, -2, -9, -2, `***`8`***`, 3, 10, -2, -3, -9]`
Deleting the three elements in bold gives you:
`A = [-5, -3, -2, -9, -2, 2, 11, -1, -4, -10]`
`B = [-4, -4, -2, -9, -2, 3, 10, -2, -3, -9]`
And taking the total element-wise squared difference (in Julia syntax): `sum((A .- B).^2)`
Gives you 7, plus 3 for the 3 deleted elements, gives you a total score of 10. Your task is to find the deletions, if any, that would minimize this score, for any two numeric arrays. Return a tuple of the optimal score, the position of the deletions in the first input array (an array of integers), and the position of the deletions in the second array (also an array of integers). Return value for the above example:
`(10.0, [2,12], [5])`
Note: The speed of your implementation, and how it scales to large input arrays, will be considered in the assessment.

Q7) Barcodes (rating: unknown):

You are asked to design a set of DNA barcodes. A DNA barcode is made up of a sequence of the letters A,C,G and T, of a particular length. For example, a DNA barcode of length 7 could be `"TAGCTAG"`. *All barcodes in a set must have the same length.* You will have to generate a set of barcodes with the following constraints:

1) In a single barcode, no letter can be immediately repeated (ie. "TA**GG**TAG" is not allowed).
2) The minimum hamming distance (https://en.wikipedia.org/wiki/Hamming_distance) between any pair of barcodes must be greater than or equal to some threshold, D. For example, here is a set of 6 barcodes of length 4, where D = 3:

   ```
   CGTA
   TCTC
   TACG
   GCAT
   AGAG
   ATGC
   ```

Your task is to write a function that takes, as input, the number of barcodes required, and the minimum distance threshold D, and returns a set of barcodes, *as short as you can find while satisfying the requirements*, using a reasonable amount of compute time. As many as 100 barcodes might be requested, with typical values of D ranging from 1 to 6. *Barcodes should be returned as an array of strings.*

Note: The speed of your implementation, and how it scales to large input arrays, will be considered in the assessment.

Hint: Especially here, do not let perfect be the enemy of good.