DSGT Fall 2021 Bootcamp Drill 1

Welcome to Data Science Club at Georgia Tech bootcamp. It's a great pleasure that you have chosen to embark on the data science journey with us. This is the first drill for bootcamp.

Learning Objectives

1. Getting acclimated to writing Python code
2. Creating your team's Github repository for the bootcamp project

From the workshop, we talked about why Python is heavily used in the field of data science. To recap, these reasons include:

- Easy to read and understand
- Lots of libraries for data science
- One of the most popular languages for data science (besides R)

To get acclimated with Python, the best way is to use it/apply it to a problem of interest. For each of the function headers, fill in the parts with `YOUR CODE HERE` keeping in mind the instructions provided to you in the comments. Test cases are provided to ensure that your functions work properly.

#Todos You will notice two files: `drill1.py` and `tester.py`. The functions you will need to implement are in `drill1.py`. Each function will have some documentation on what needs to be done. `tester.py` contains unit tests to test your work in `drill1.py`. Before starting this drill, make sure to run `pip install pytest` to set up the testing environment.

Problem 1: Cubing a Number Given a number `n`, return the cube of the number. For example, `cube(4) = 64` This text you see here is *actually- written in Markdown! To get a feel for Markdown's syntax, type some text into the left window and watch the results in the right. Put your code in the `cube()` function in `drill1.py`

Problem 2: Taking the Factorial of a Number Given a number `n`, return `n!`. Recall that `n! = n*(n-1)*(n-2)*...*3*2*1` and `0! = 1`. If the number passed in is not an integer, throw a `ValueError` with an informative message explaining why `ValueError` was thrown. You should also throw a `ValueError` if an integer input to factorial is not valid. Put your code in the `factorial()` function in `drill1.py`

Problem 3: Counting the digits in a number Given a number `n` where `n` is a non-negative number, count the number of digits. For example, `count_digits(123) = 3` and `count_digits(99) = 2`. Put your code in the `count_digits()` function in `drill1.py`

Problem 4: Attaining a Target Average Grade Given a set of exam scores as such: `{"Exam 1": 43, "Exam 2": 59, "Exam 3": 60, "Exam 4": 90}`, find how much the student will need to score on the final exam to acheive the target grade. Assume that all exams (including the final exam) are weighted equally for the sake of simplicity.

In the above example, if the student wants target grade of `70`, the student will need to score `98` on the final (`70*5 exams = 43+59+60+90+x`, solve for x)

Put your code in `average_grade()` function in `drill1.py`

Problem 5: Taking Product of Subset of Elements in List Given a list of numbers, a start index, and end index, find the product of the entries in the given slice. Below

is an example:

Suppose that the list is `[1,2,3,4,5]`, `start_pos = 1`, `end_pos = 3`, then slice_product should return `24` (lists are zero-indexed in the sense that the first element in the list is at index `0`, so 234 = 24)

If the `start_pos` is a negative number, raise/throw a `ValueError` with an informative reason. If the `end_pos` is an integer that is at least the length of the list, raise/throw a `ValueError` (eg: in the above array, if `end_pos` >= 5, raise a `ValueError`.)

If the `end_pos` is smaller than the `start_pos`, raise/throw a ValueError as well. The `start_pos` and `end_pos` are guaranteed to be integers. Your implementation should go in the `slice_product()` function under `drill1.py`

Problem 6: Caesar Cipher Encryption This function is responsible for encrypting a message using Caesar Cipher. The Caesar Cipher is basically shifting the alphabet. For example, if the shift is `2`, then `a` becomes `c`, `b` becomes `d` and so on. If the shift is `1`, then `a` becomes `b`, `b` becomes `c` and so on.

Helpful Link to understand Caesar Cipher: https://medium.com/blockgeeks-blog/cryptography-for-dummies-part-2-the-caesar-cipher-665106afac78

If the shift value is negative, you will need to perform a manipulation to find an equivalent shift value (for example if shift was `-27`, the equivalent shift value that's positive is `25`)

(Hint: For the shift value case, think mathematically how the Cipher works)

Caesar Cipher is **case insensitive!!!** ("J" shifted by 3 should be treated the same as "j" shifted by 3 for example). How can you use this property to simplify your solution?

Implement your solution in the `encrypt()` function in `drill1.py`

Problem 7: Caesar Cipher Decryption Given a `message` and `shift`, decrypt the message. Same properties from Problem 6 hold.

Once you pass all the unit tests from `tester.py`, zip the folder for the drill and put it in your folder for the bootcamp workshops that you made in Google Drive (ie: the one under `Participants` folder). Let your bootcamp mentor know that you've completed the drill and provide a screenshot showing that you've passed all the unit tests.