Scala - Exercises 1 - Recursion

Notes

Intensive Track

- If you are completing the intensive track then it's expected you'll complete these exercises *before* the next session (i.e. in 2 days), with the exception of the challenge question which is recommended but optional.
- When you have solutions, please send a link to a GitHub repo with them to Nunana and I so we can take a look through them, provide comments etc. It is *strongly* encouraged to write tests and show these pass too. MUnit is a simple scala testing framework that should be simple to set up and use however you can also use ScalaTest, JUnit or any other framework you prefer

Non-Intensive Track

- If you are on the non-intensive track, the exercises are optional as time will be difficult with projects. For some practice however I would recommend:
  - if you are already familiar with Scala and recursion I would suggest the following questions:
    - Warm up - Question 1
    - A Little More Fun - Questions 2 and 3
    - Challenge Question
  - If you are unfamiliar with Scala or recursion I would suggest:
    - Warm up - Question 1
    - Warm up - Questions 1, 4,5
    - A Little More Fun - Question 3

Benefiting from Exercises

As with almost all exercises, it is possible to find answers to these problems online. Nothing will stop you doing that however *please* avoid looking answers up and instead speak to me or Nunana for assistance if you get stuck. You *will* learn significantly more, completing the exercises yourself-even if that means doing less of them-than looking the answers up on line

Warm Up Questions

In each of these cases, for now do not concern yourself with unexpected input. The purpose here is on practicing recursion, not edge cases and error handling

1. Set up a Scala environment within intelliJ and ensure that you can run a hello world program

Write recursive functions for the following functions:

2. Returns the sum of the numbers between two numbers a and b
3. Concatenate a given string n times
4. Find the length of a string
5. Calculate fibonacci number for a provided value n
6. Returns whether a given number n is prime (i.e. it is divisible by itself and 1)

A Little More Fun :-)

1. If you haven't already, re-write each of the above functions to be tail recursive, what do you notice in difference in performance between the two methods?

Write recursive functions for the following functions:

2. counts the characters in a string and returns a map from each character to the number of times it occurs. The signature of this function is:

```
def countCharacters(s: String): Map[Char, Int]
```

3. Checks whether a string containing only parenthesis, e.g. () characters is balanced. A string is balanced, for example:
    1. () - true
    2. ((())) - true
    3. (())()()(())((())) - true
    4. (() - false
    5.)(- false

    The signature of the function is:

```
def hasValidParenthesis(s:String): Boolean
```

4. If you haven't already, write each of the above functions using tail recursion

Challenge Question

1. Write a recursive function that generates all possible combinations of valid parenthesis strings of a given length, provided some input length n. Your function should be tail recursive and the signature of the function is:

```
def generateValidParenthesis(n: Int): List[String]
```

Hints: For this problem you may want to look up for/yield comprehensions