Exercise 3

Higher Order Functions in Scala

- Last Week Recap (Recursion)
- List Data Type (an abstract data type that represents a finite number of ordered values, where the same value may occur more than once)
- Functions vs Higher Order Functions
  - *A self-contained block of code that performs a specific task (Function)*
  - *A function that takes another function as an argument or returns a function as its result (HOF)*
- Difference b/w functions and higher-order functions
  - **Functionality:** A regular function is used to perform a specific task, such as adding two numbers or displaying a message on the screen. In contrast, a higher-order function is used to manipulate functions and can be used to create new functions, combine existing functions, or modify the behaviour of existing functions.
  - **Flexibility:** Higher-order functions are more flexible than regular functions because they can take other functions as arguments or return functions as their result. This allows for more dynamic and adaptable code.
  - **Reusability:** Higher-order functions are more reusable than regular functions because they can be used to create generic, reusable code that can be applied to a variety of different scenarios.
  - **Complexity:** Higher-order functions tend to be more complex than regular functions because they involve manipulating other functions. This can make them more difficult to understand and debug.

Classwork

In this lab, you will be exploring the fundamentals of higher order functions in Scala. You will learn about the syntax for HOFs definitions, practice how to pass functions as arguments to functions and return functions as a result.

Homework

1. Write a higher-order function called **operateOnList** that takes a list of integers and a function **oddToEven** (that convert odd integer to even) as an arguments, and applies the function to each element in the list, returning a new list of results with all even integers.
2. Write a function called **filterAndMap** that takes a list of strings, a predicate function **greaterLength** (that filters the string which has length greater than 6), and a mapping function **concat** (that concatenates the length of string with the string i.e. "*string6*") as arguments, and returns a new list of transformed elements that meet the given predicate.

3. Create a higher-order polymorphic function **listTransformation** in Scala that works with any data type and any transformation function.
   It should takes in two arguments:
   - **List** (A list of values of any data type)
   - **transformFunc** (A function that transforms each element in the list into another element of the same or a different type)

   Example 1: a list of strings, a transformation function that transforms a string to string length

   Example 2: a list of integers, a transformation function that transforms an integer to its double value

   After implementing the function, you should test it with at least two different examples to demonstrate its versatility and flexibility.
4. Create a program that demonstrates the use of currying and partially applied functions. I should have the following three components:
   - A **curried function** that takes in three arguments of type Int and returns their product.
   - A **partially applied function** that uses the curried function to multiply two numbers by a fixed constant.
   - A **main function** that demonstrates the use of the curried and partially applied functions with different arguments.

   After implementing the program, you should test it with at least three different examples
   [HINT: Lecture Slides 14-18]

Deliverable

Deliverable: Submit a single scala code file with "*.scala*" extension, and write your Name and Student ID in the code as a comment. The whole deliverable must be well commented and supported with descriptions where required.
Deadline: 07.04.2023 12:00 am [Before Next Friday Session]
Submission: (session respective) Return box on Moodle.
Estimated workload: <= 2 hours
Warning: This is individual work. Strict actions will be taken for plagiarism!

Deliverables for Exercise 3:

1. Implementation of the homework part.

Note:
Make sure to follow the Scala style guide and use appropriate naming conventions for variables, functions, and classes. Your code should be well-organized, well-documented, and easy to read.