Week 1.2

Lab Activity: Functional Data Transformation Playground

Learning Objectives:

- Master essential JavaScript functions for working with strings, arrays, and objects.
- Understand and apply functional programming principles:
  - Pure functions: Always produce the same output for a given input, no side effects.
  - Higher-order functions: Take or return functions.
  - Immutability: Avoid modifying original data, and create new transformed copies.
- Practice function composition to create complex transformations from simpler ones.

Scenario:

You are given a dataset containing various types of information: strings, arrays of numbers, and objects representing people. Your task is to build a set of pure functions to extract, transform, and functionally analyze this data.

Tasks:

1. String Transformations:
   a. `capitalize(str)`: Capitalizes the first letter of a string.
   b. `reverse(str)`: Reverses a string.
   c. `isPalindrome(str)`: Checks if a string is a palindrome (reads the same backward as forward).
   d. `wordCount(str)`: Counts the number of words in a string.
2. Array Transformations:
   a. `double(arr)`: Doubles every number in an array.
   b. `filterEven(arr)`: Filters out even numbers from an array.
   c. `sum(arr)`: Calculates the sum of all numbers in an array.
   d. `average(arr)`: Calculates the average of all numbers in an array.
3. Object Transformations:
   a. `fullName(person)`: Returns the full name of a person object (given properties `firstName` and `lastName`).

   b. `isAdult(person)`: Checks if a person is 18 or older (given property `age`).
   c. `filterByAge(people, minAge)`: Filters an array of person objects to keep only those at least `minAge` years old.
4. Function Composition:
   a. Use the `compose(...fns)` function (you can find implementations online) to combine your functions in interesting ways. For example, create a function to reverse and capitalize a string, or to double all the even numbers in an array.

Evaluation:

- Code Quality:
  - All functions are pure (no side effects).
  - The code is well-organized, readable, and uses clear variable names.
  - Functions are appropriately decomposed into smaller, reusable pieces.
  - Correct handling of potential errors (e.g., empty arrays, invalid input types).
- Functionality:
  - All tasks are completed, and the functions work correctly with the provided datasets and various inputs.
  - Function composition is used effectively to create more complex transformations.
- Understanding:
  - Demonstrate a solid grasp of functional programming concepts and how they apply to data manipulation.
  - Explain the reasoning behind function implementations and composition choices.