Lab Worksheet: Test-Driven Development (TDD) with a UK Student Record System

Objective: To learn and apply the principles of Test-Driven Development (TDD) while building a function to calculate a student's final grade in a UK-centric grading system.

Total Time: 2 hours (plus time for reflection)

Materials:

- Code editor (e.g., VS Code, Atom)
- Web browser (for running JavaScript code)

Activities:

Activity 1: Setting up the Testing Environment (30 minutes)

1. **Create a new JavaScript file:** Name it `studentGrades.js`.
2. **Implement a basic testing function:** Add the following code to your file. This function will help you run tests and check if they pass or fail.

```
function test(message, assertion) {
    try {
    const result = assertion();
    console.log(result ? `Pass: ${
    message
}
` : `Fail: ${
    message
}
`);
    }
catch (error) {
    console.error(`Error: ${
    message
}
- ${
    error.message
}
`);
    }

}
```

3. **Write your first test:** Let's start with a simple test case for our calculateFinalGrade function (which we haven't created yet). This function

will take an array of module marks as input.

```
test('calculates final grade for all marks above 70', () => {
    const marks = [75, 80, 72];
    const grade = calculateFinalGrade(marks);
    // This will throw an error initially
  return grade === 'First Class';
    }
);
```

4. **Run the test:** Open your studentGrades.js file in a web browser. You should see an error in the console because the calculateFinalGrade function doesn't exist yet.

Reflection (5 minutes):

Why is it important to write the test before the code? What is the purpose of the test function?

Activity 2: Implementing the Core Functionality (45 minutes)

Create the calculateFinalGrade function: Now, implement the function with the minimum amount of code to make the test pass.

```
function calculateFinalGrade(marks) {
    // For now, just return 'First Class' to pass the test
  return 'First Class';
    }
```

Run the test again: You should see a "Pass" message in the console.

Write another test: Add a test for a different scenario (marks between 60 and 69).

```
test('calculates final grade for marks between 60 and 69', () => {
    const marks = [62, 68, 65];
    const grade = calculateFinalGrade(marks);
    return grade === '2:1';
    }
);
```

Run the tests: This new test should fail.

Refactor the function: Improve the calculateFinalGrade function to handle both test cases correctly. You'll need to calculate the average mark and use conditional statements (if/else if) to determine the final grade based on UK grading boundaries (First Class, 2:1, 2:2, Third Class, Fail).

Tip: Use a for loop to sum the marks in the array and then divide by the number of marks to get the average.

Reflection (5 minutes):

How did you approach refactoring the function to make both tests pass? What challenges did you encounter, and how did you overcome them?

Activity 3: Handling Edge Cases and Expanding Tests (45 minutes)

Add more tests: Write more tests to cover various scenarios:

Different grade boundaries (2:2, Third Class, Fail) Edge cases: Empty array input ([]) Non-array input ("not an array") Invalid mark values (below 0 or above 100) Tip: For edge cases, use try catch blocks in your tests to check if the function throws the expected errors. If you're unsure, check the MDN documentation https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/try...catch

Refactor the function: Update the calculateFinalGrade function to handle the edge cases correctly. You'll need to add input validation to check for valid array input and mark values. Use throw new Error("error message") to throw errors for invalid input.

```
if (!Array.isArray(marks) || marks.length === 0) {
throw new Error("Invalid input: marks must be a non-empty array");
}
```

Run all tests: Ensure all tests pass after each change you make to the function. Reflection (5 minutes):

Why is it crucial to test for edge cases? How does TDD help in identifying and handling potential errors? **Bonus:**

Explore using Array.reduce() to simplify the calculation of the average mark. Research and implement more advanced testing techniques (e.g., mocking, stubbing) using a testing framework like Jest. I have videos on the VLE of me developing a unit test framework from scratch and then using Python's built in unittest framework to do the heavy lifting for me.