Laboratory Activity 3: JavaScript Fundamentals 1

Cloud Sedgwick B. Daet

BSCS - 2B

Link of the GitHub repository for the activity.

TABLE OF CONTENTS

OBJECTIVES

- To apply **conditional statements** (if, else if, else) in solving problems.
- To implement **loops** (for, while) for repetitive tasks.
- To create and call **functions** that can be reused.
- To demonstrate problem-solving by coding unique variations of a task.

EXPECTED OUTPUT

A program file (.js) implementing all 4 problems.

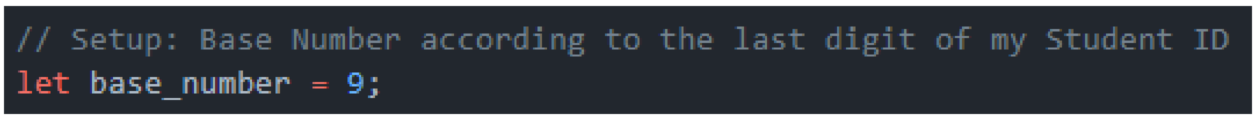
A short report containing:

- Source code screenshots.
- Sample outputs.
- Link of your GitHub repository for the activity

METHODOLOGY

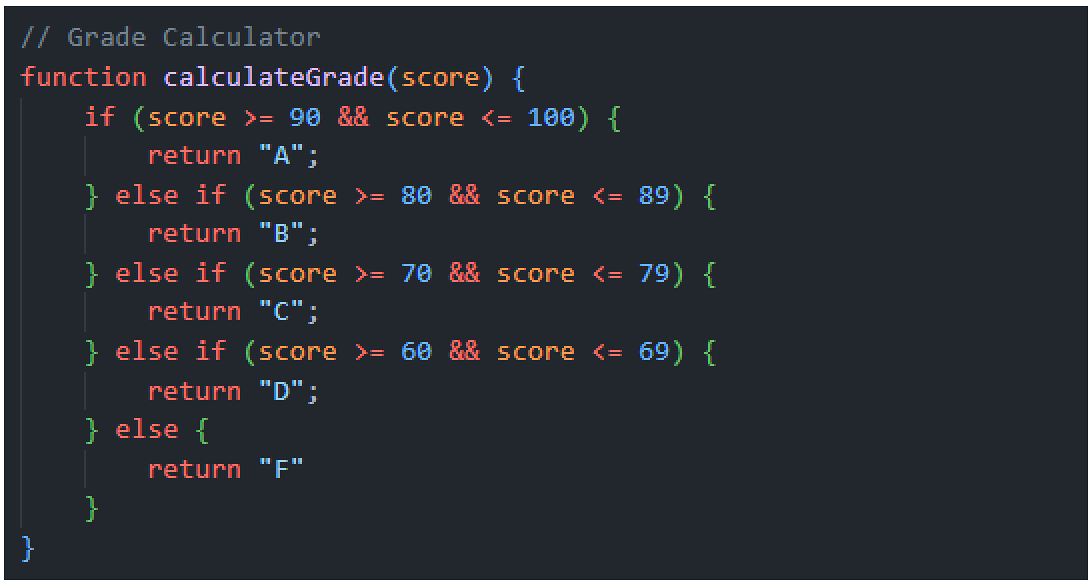
Base Number

The base of everything was simple: my Student ID's last digit, **9**. As per the instructions, all functions in this project uses `base_number` as input.



```
// Setup: Base Number according to the last digit of my Student ID
let base_number = 9;
```

Grade Calculator

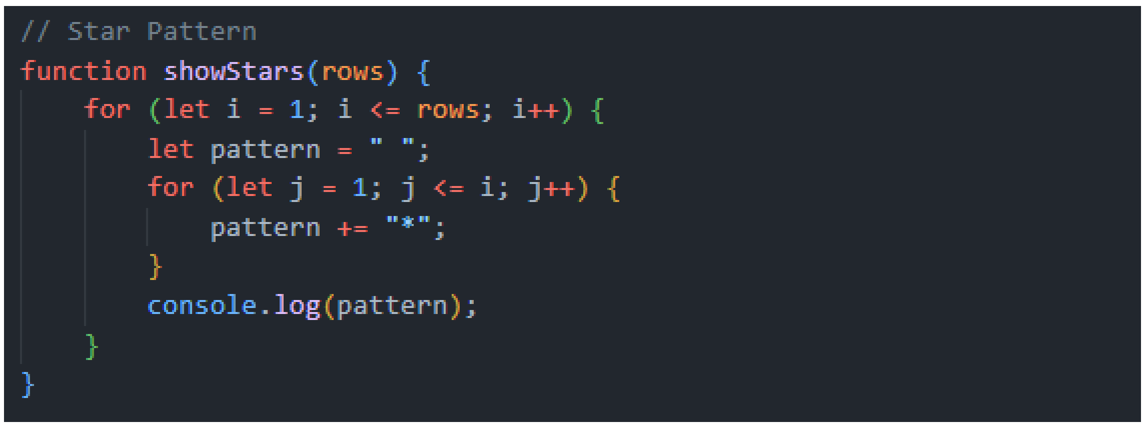


```javascript
// Grade Calculator
function calculateGrade(score) {
    if (score >= 90 && score <= 100) {
        return "A";
    } else if (score >= 80 && score <= 89) {
        return "B";
    } else if (score >= 70 && score <= 79) {
        return "C";
    } else if (score >= 60 && score <= 69) {
        return "D";
    } else {
        return "F"
    }
}
```

A function that checks. Nothing more, nothing less. It looks at the score, compares it against ranges, and decides.

I chose this structure because it's readable, honest, and mirrors how we grade in real life: "If it's above 90, that's an A. If not, check the next range." Simple and straightforward.

Star Pattern

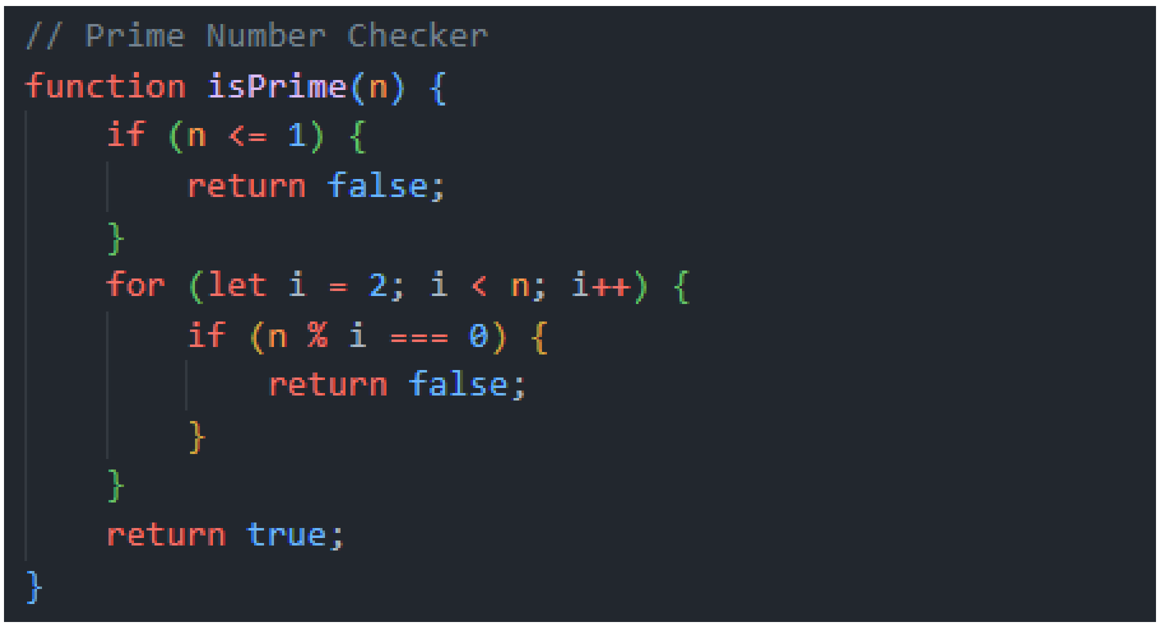


```javascript
// Star Pattern
function showStars(rows) {
    for (let i = 1; i <= rows; i++) {
        let pattern = " ";
        for (let j = 1; j <= i; j++) {
            pattern += "*";
        }
        console.log(pattern);
    }
}
```

A loop inside a loop.

The outer loop controls rows—the height of the triangle. The inner loop builds the pattern, star by star.

Prime Number Checker



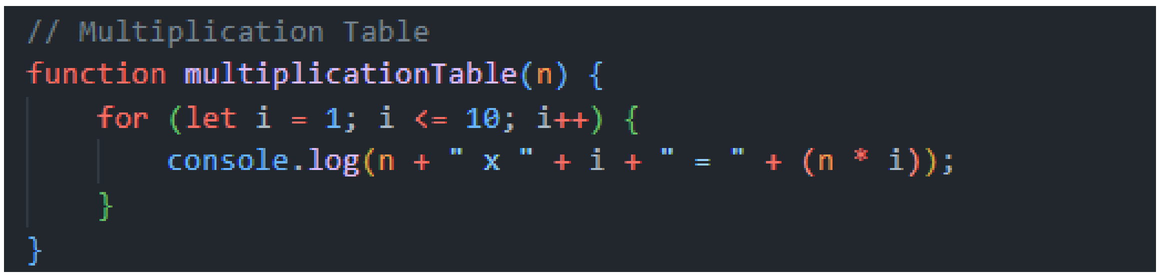
```javascript
// Prime Number Checker
function isPrime(n) {
    if (n <= 1) {
        return false;
    }
    for (let i = 2; i < n; i++) {
        if (n % i === 0) {
            return false;
        }
    }
    return true;
}
```

Primes are strange. They don't divide evenly, except by 1 and themselves.

To test that, I used a loop. From 2 up to `n-1`, checking one divisor at a time. If one divides evenly, the number is not prime. Otherwise, it survives the gauntlet.

Why this? Because it's the most human way to explain it. Try dividing. If it works, it fails. If it never works, it's prime.

Multiplication Table

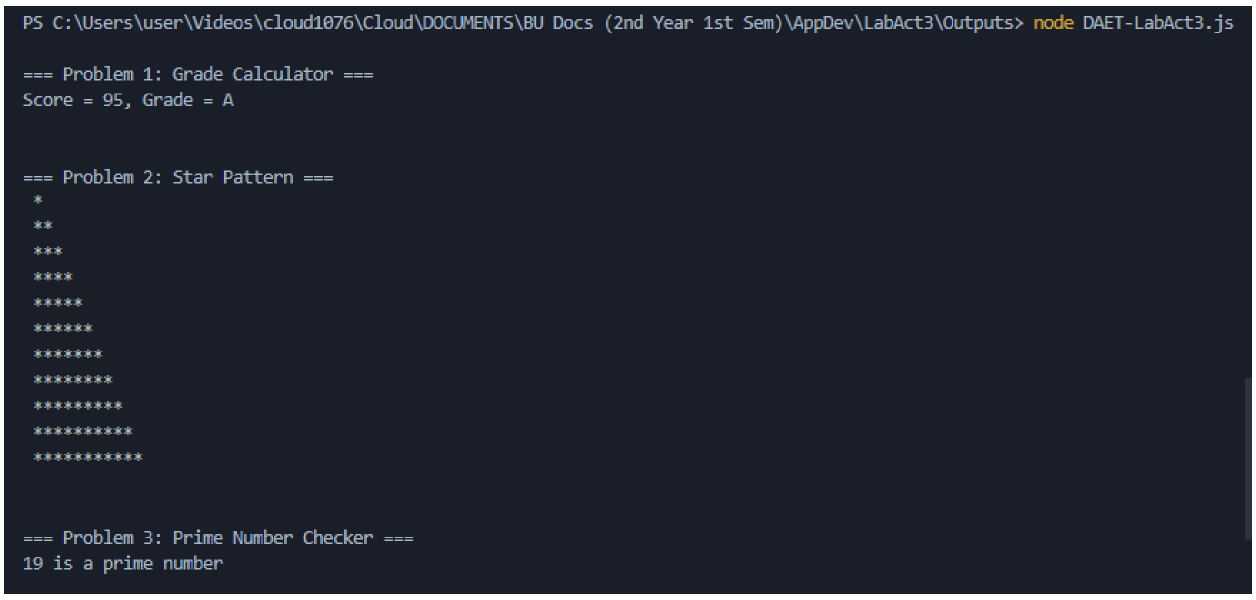


```javascript
// Multiplication Table
function multiplicationTable(n) {
    for (let i = 1; i <= 10; i++) {
        console.log(n + " x " + i + " = " + (n * i));
    }
}
```

Just a loop from 1 to 10. Multiply. Print. Repeat. No tricks. No hidden logic.

I chose a `for` loop here too, because the repetition is fixed. Always ten lines. It fits perfectly.

SAMPLE OUTPUTS

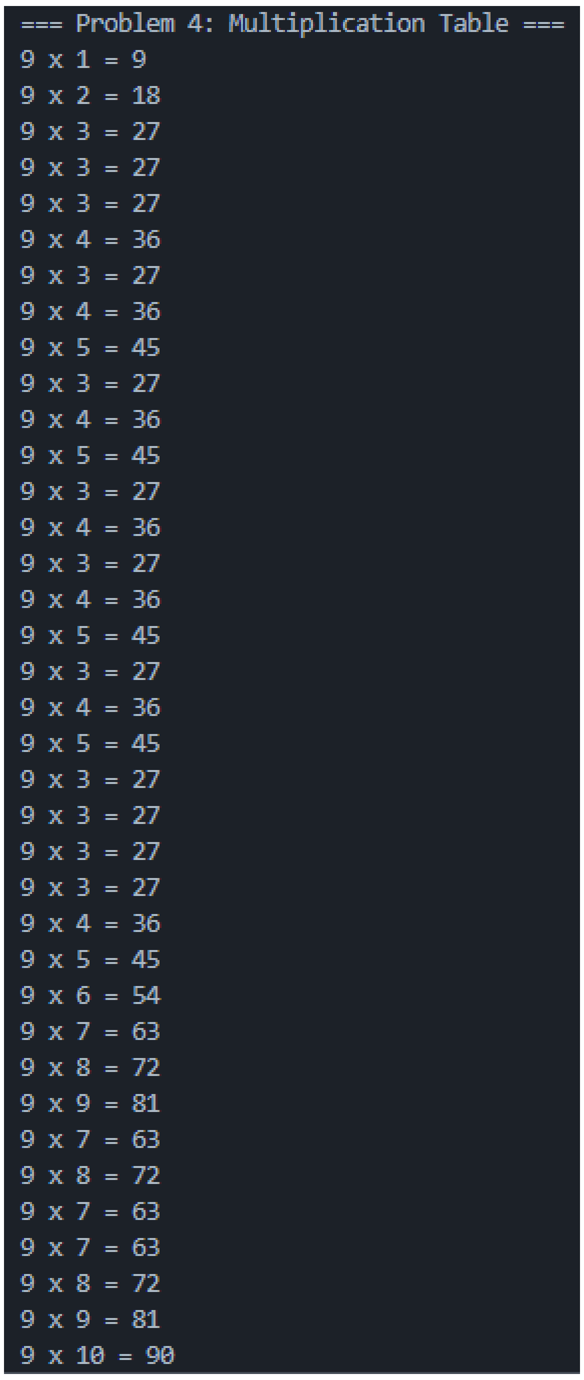


```
PS C:\Users\user\Videos\cloud1076\Cloud\DOCUMENTS\BU Docs (2nd Year 1st Sem)\AppDev\LabAct3\Outputs> node DAET-LabAct3.js

=== Problem 1: Grade Calculator ===
Score = 95, Grade = A


=== Problem 2: Star Pattern ===
 *
 **
 ***
 ****
 *****
 ******
 *******
 ********
 *********
 **********
 ***********


=== Problem 3: Prime Number Checker ===
19 is a prime number
```



```
=== Problem 4: Multiplication Table ===
9 x 1 = 9
9 x 2 = 18
9 x 3 = 27
9 x 3 = 27
9 x 3 = 27
9 x 4 = 36
9 x 3 = 27
9 x 4 = 36
9 x 5 = 45
9 x 3 = 27
9 x 4 = 36
9 x 5 = 45
9 x 3 = 27
9 x 4 = 36
9 x 3 = 27
9 x 4 = 36
9 x 5 = 45
9 x 3 = 27
9 x 4 = 36
9 x 5 = 45
9 x 3 = 27
9 x 3 = 27
9 x 3 = 27
9 x 3 = 27
9 x 4 = 36
9 x 5 = 45
9 x 6 = 54
9 x 7 = 63
9 x 8 = 72
9 x 9 = 81
9 x 7 = 63
9 x 8 = 72
9 x 7 = 63
9 x 7 = 63
9 x 8 = 72
9 x 9 = 81
9 x 10 = 90
```