Hands-on Activity 4.1 | Code Debugging in JavaScript	
Course Code: CPE 026	**Program:** Computer Engineering
Course Title: Emerging Technologies 3 in CpE	**Date Performed: 9/19/2024**
Section: BSCPE41S8	**Date Submitted: 9/19/2024**
Name: Christian Ed B. Efa	**Instructor:** Engr. Roman Richard

1. Discussion

Discuss here the relevant concepts of the activity in your own words.

2. Materials and Equipment

What materials did you use? Explain in detail.

3. Procedure

What are the procedures that you performed?

6.1.1.5 Errors without exceptions

« 6.1.1.5 Errors without exceptions? »

```
console.log(100 / 0); // -> Infinity
console.log(100 * "2"); // -> 200
console.log(100 * "abc"); // -> NaN
console.log(Math.pow("abc", "def")); // -> NaN
```

- Needs to check on how they behave because Some of them will generate exceptions, while others will return some specific values.

6.1.1.6 Limited confidence

« 6.1.1.6 Limited confidence »

```
let sX = prompt("Enter the first number");
let sY = prompt("Enter the second number");
let x = Number(sX);
let y = Number(sY);
if (Number.isFinite(x) && Number.isFinite(y) && y !== 0) {
    console.log(x / y);
} else {
    console.log("incorrect arguments");
}
```

- This code prompts the user to enter two numbers, converting the input strings into numbers while validating that both are finite and that the divisor is not zero before performing the division. This

approach ensures that the program handles potential errors gracefully, preventing issues like division by zero or invalid number formats.

6.2.1.7 Conditional exception handling

« 6.2.1.7 Conditional exception handling »

```
variable instanceof type

```

« 6.2.1.7 Conditional exception handling »

```
let a = -2;
try {
    a = b;
} catch (error) {
    if (error instanceof ReferenceError) {
        console.log("Reference error, reset a to -2"); // -> Reference error, reset a to -2
        a = -2;
    } else {
        console.log("Other error - " + error);
    }
}
console.log(a); // -> -2

```

- its important to know that any variable that gets declared using the let keyword inside a try block is not accessible in the catch block

6.2.1.8 The finally statement

« 6.2.1.8 The finally statement »

```
try {
    // code to try
} finally {
    // this will be always executed
}
```

« 6.2.1.8 The finally statement »

```
let a = 10;
try {
    a = 5;
} finally {
    console.log(a); // -> 5
}
console.log(a); // -> 5
```

« 6.2.1.8 The finally statement »

```
let a = 10;
try {
    a = b;  // ReferenceError
} finally {
    console.log(a); // -> 10
}
console.log(a);
```

- The exception (ReferenceError) interrupts the program in the try block. Because the JavaScript engine cannot find the catch block, it immediately jumps to the finally block, executing its contents and ending its work.

« 6.2.1.8 The finally statement »

```
let a = 10;
try {
    a = b;  // ReferenceError
} catch (error) {
    console.log("An Error!"); // -> An Error!
} finally {
    console.log("Finally!"); // -> Finally!
}
console.log(a); // -> 10
```

- In this case, the exception causes a jump to the catch block, then to the finally block. The program then continues to work outside of the try...catch statement.

6.2.1.9 Why should we use a finally block?

« 6.2.1.9 Why should we use a `finally` block? »

```
let a = 10;
try {
    a = b;  // ReferenceError
} catch (error) {
    console.log("An Error!");
}
console.log("Finally!");
```

6.2.1.9 Why should we use a `finally` block? »

```
let a = 10;
try {
    a = b;  // First ReferenceError
} catch (error) {
    console.log(b); // Second ReferenceError

}
console.log("Finally!");
```

« 6.2.1.9 Why should we use a `finally` block? »

```
let a = 10;
try {
    a = b;  // First ReferenceError
} catch (error) {
    console.log(b); // Second ReferenceError

} finally {
    console.log("Finally!");
}
```

6.2.1.9 Why should we use a `finally` block? »

```
let a = 10;
try {
    a = b; // First ReferenceError
} catch (error) {
    try {
        console.log(b); // Second ReferenceError
    } catch {
        console.log("Second catch!"); // -> Second catch!
    }
} finally {
    console.log("Finally!"); // -> Finally!
}
```

- Using a finally block ensures that specific code will execute regardless of whether an error is thrown, even if that error occurs within the catch block. This is particularly useful for cleanup actions or final statements that need to run, ensuring that they execute no matter what happens in the preceding try or catch blocks.

6.2.1.10 The throw statement and custom errors

« 6.2.1.10 The throw statement and custom errors »

```
console.log("start"); // -> start
throw 100; // -> Uncaught 100
console.log("end");
```

- An unsupported exception (if the number 100 can be called an exception) causes the program to stop. The second console.log instruction is never executed.

« 6.2.1.10 The throw statement and custom errors »

```
console.log("start"); // -> start
try {
    throw 100;
} catch (error) {
    console.log(error); // -> 100
}
console.log("end"); // -> end
```

- his time, the exception is caught and handled in the catch block, and doesn't interrupt further execution.

6.2.1.13 Errors and Exceptions - Tasks 1

« 6.2.1.13 Errors and Exceptions - Tasks »

```
function div(a, b) {
    if (b == 0) {
        throw new RangeError("Can't divide by 0");
    }
    return a / b;
}
console.log(div(4, 2)); // -> 2
console.log(div(4, 0)); // -> Uncaught RangeError: Can't divide by 0
```

- On this task we need to write our own div function that will take two call arguments and return the result of dividing the first argument by the second. In JavaScript, the result of dividing by zero is the value Infinity (or -Infinity, if we try to divide a negative number).

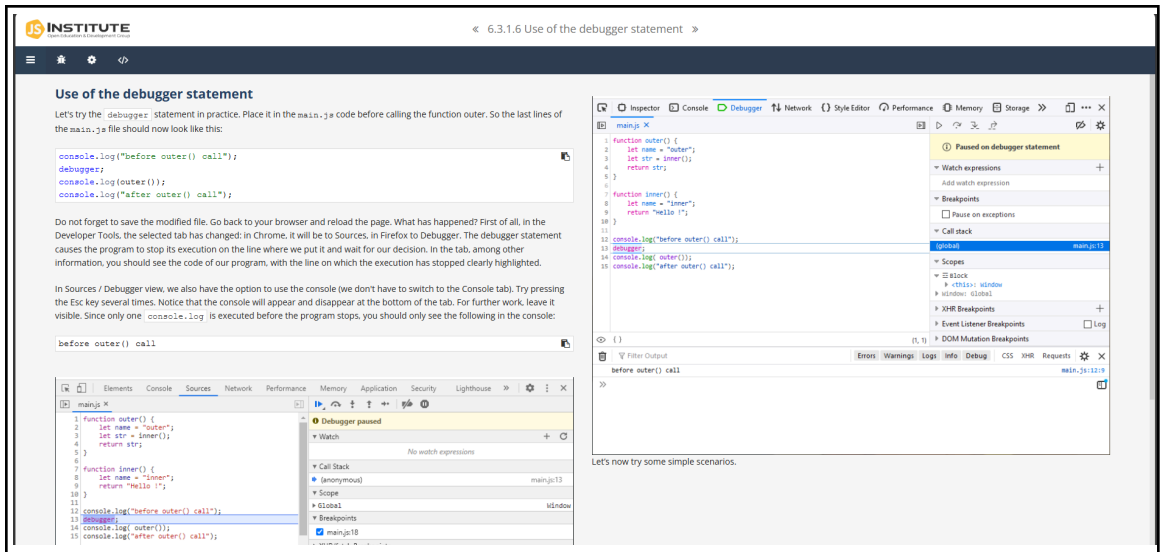
6.2.1.13 Errors and Exceptions - Tasks 2

« 6.2.1.13 Errors and Exceptions - Tasks »

```
for (let i = 0; i < numbers.length; i++) {
    let result;
    try {
        result = div(1000, numbers[i]);
    } catch (e) {
        result = e.message;
    }
    console.log(result);
}
```

- On this task we are needed to write a program that, in a loop, divides the number 1000 by successive elements of the numbers array, displaying the result of each division.

6.3.1.6 Use of the debugger statement



- This module is about the Use of the debugger statement.The debugger statement causes the program to stop its execution on the line where we put it and wait for our decision.

6.3.1.15 Measuring code execution time - continued

« 6.3.1.15 Measuring code execution time - continued »

```
let part = 0;
console.time('Leibniz');
for(let k = 0; k < 10000000; k++) {
    part = part + (k % 2 ? -1 : 1) / (2 * k + 1);
}
console.timeEnd('Leibniz'); // -> Leibniz: 175.5458984375 ms
let pi = part * 4;
console.log(pi);
```

- In each iteration, the number -1 is raised to the power of k. Exponentiation is quite a time-consuming operation, so it strongly affects the speed of the program.

6.3.1.17 Testing your code - Tasks 1

« 6.3.1.17 Testing your code - Tasks »

```
let end = 2;
for(let i=1; i<end; i++) {
    console.log(i);
}
```

- In this task, with the given code snippet that initializes a variable end to 2 and uses a for loop to log numbers from 1 up to, but not including, end. To output the numbers 1, 2, 3, 4, and 5 instead,

you need to use the debugger to change the value of end during execution without modifying the code itself.

6.3.1.17 Testing your code - Tasks 2

« 6.3.1.17 Testing your code - Tasks »

```
let counter = 0;
let maxValue = 10;
let result = 1;

debugger;
for (counter = 0; counter < maxValue; counter++)
    console.log(result);
    result *= maxValue - counter - 1;
}

console.log("Final result: ", result);
```

- This task involves a loop where a variable result is supposed to increase, but the final output logged is zero. By using the debugger and the Watch feature.

6.3.1.17 Testing your code - Tasks 3

« 6.3.1.17 Testing your code - Tasks »

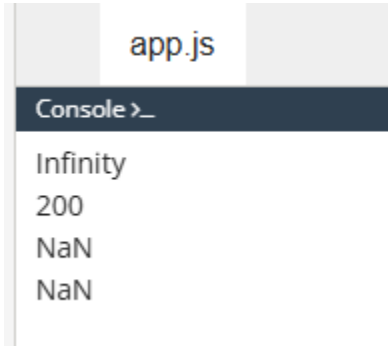
```
function max(array) {
    let maxValue = array[1];
    for(let i=1; i<array.length; i++) {
        if(array[i] > maxValue) {
            maxValue = array[i];
        }
    }
    return maxValue;
}

console.log( max([1, 4, 6, 2])); // -> 6
console.log( max([10, 4, 6, 2])); // -> 6
```

- This task is provided with a function that is supposed to find the maximum value in an array, but it incorrectly returns 6 instead of 10 for the second input. Using the debugger, you will step through the max function to observe the values of the loop variable i and maxValue, allowing you to identify and fix the logic error causing the incorrect output.

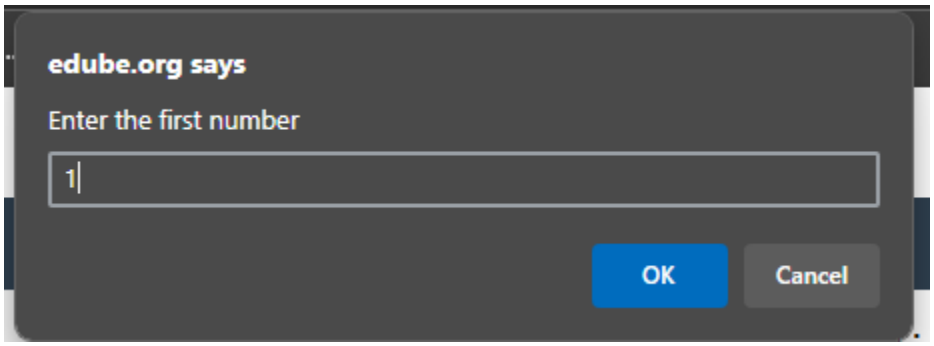
4. Output

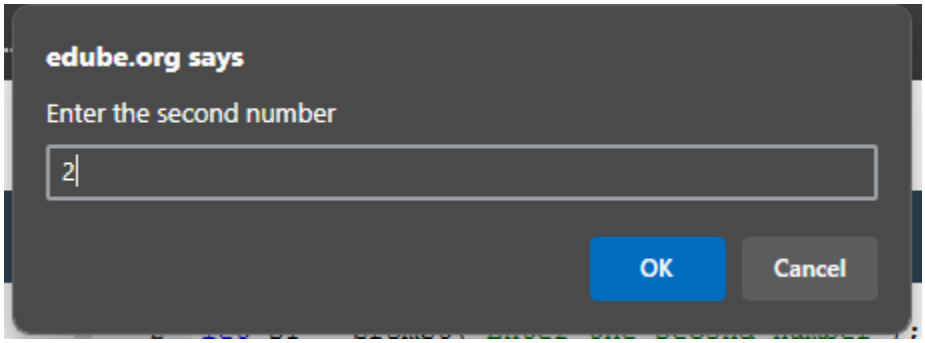
Screenshot of your outputs based on the procedures.

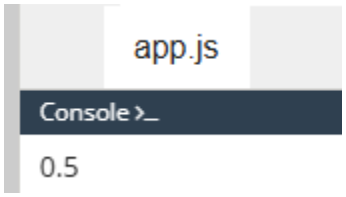
6.1.1.5 Errors without exceptions



6.1.1.6 Limited confidence







6.2.1.7 Conditional exception handling

app.js

Console >_

```
Uncaught ReferenceError: variable is not defined
```

app.js

Console >_

```
Reference error, reset a to -2
-2
```

6.2.1.8 The finally statement

app.js

Console >_

```
5
5
```

app.js

Console >_

```
10
Uncaught ReferenceError: b is not defined
```

app.js

Console >_

```
An Error!
Finally!
10
```

6.2.1.9 Why should we use a finally block?

app.js

Console >_

```
An Error!
Finally!
```

app.js

Console >_

```
Uncaught ReferenceError: b is not defined
```

app.js

Console >_

```
Finally!
Uncaught ReferenceError: b is not defined
```

app.js

Console >_

```
Second catch!
Finally!
```

6.2.1.10 The throw statement and custom errors

app.js

Console >_

```
start
Uncaught 100
```

app.js

Console >_

```
start
100
end
```

6.2.1.13 Errors and Exceptions - Tasks 1

app.js

Console >_

```
2
Uncaught RangeError: Can't divide by 0
```

6.2.1.13 Errors and Exceptions - Tasks 2

app.js

Console >_

```
Uncaught ReferenceError: numbers is not defined
```

6.3.1.15 Measuring code execution time - continued

app.js

Console >_

```
Leibniz
Leibniz: 10.799999999813735 ms
Leibniz
3.1415925535897915
```

6.3.1.17 Testing your code - Tasks 1

app.js

Console >_

```
1
```

6.3.1.17 Testing your code - Tasks 2

app.js

Console >_

```
1
9
72
504
3024
15120
60480
181440
362880
362880
Final result: 0
```

6.3.1.17 Testing your code - Tasks 3

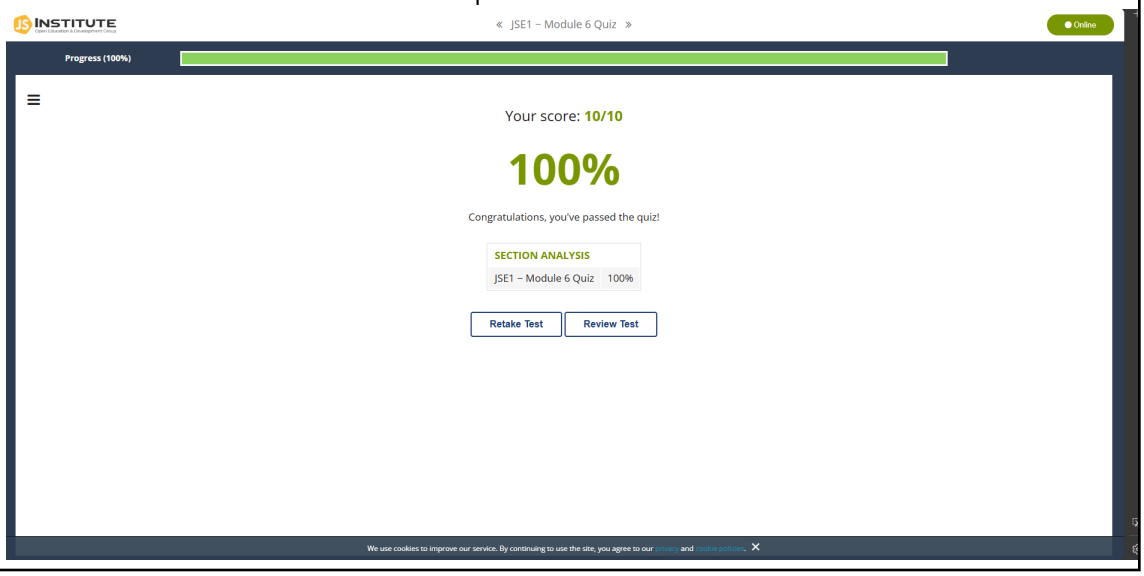
app.js

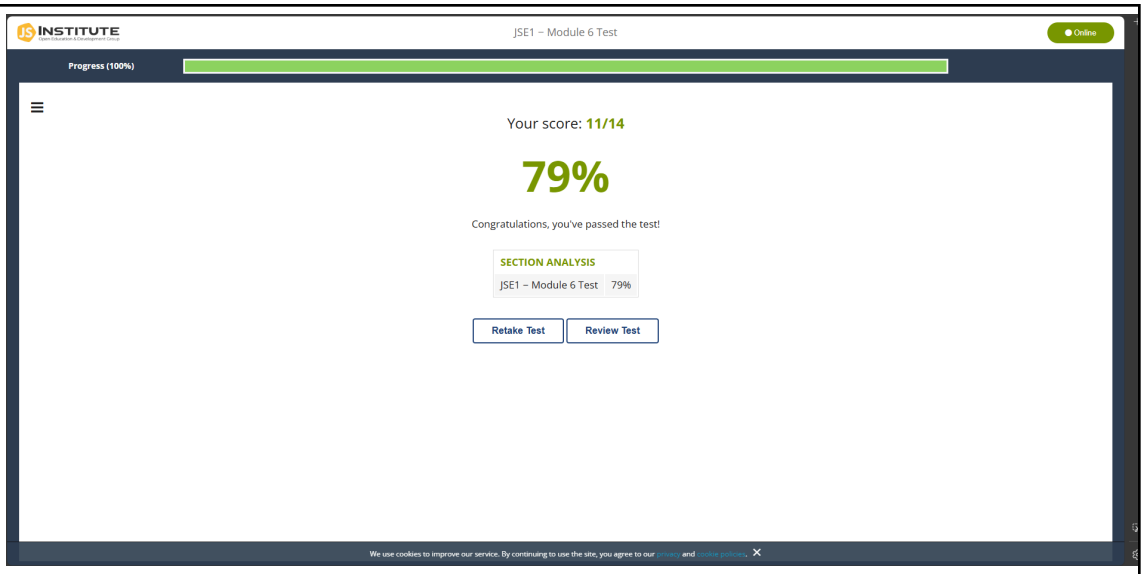
Console >_

```
6
6
```

5. Supplementary Activity

Include here screenshots of the module completion test.





6. Assessment Rubric

Lab Activity Rubric

Criteria	Ratings						Pts
SO 7 PI 1 **Student Outcome 7.1** Acquire and apply new knowledge from outside sources. threshold: 4.8 pts	6 pts Excellent | Educational interests and pursuits exist and flourish outside classroom requirements,knowledge and/or experiences are pursued independently and applies knowledge learned into practice	5 pts Good | Educational interests and pursuits exist and flourish outside classroom requirements,knowledge and/or experiences are pursued independently	4 pts Satisfactory | Look beyond classroom requirements, showing interest in pursuing knowledge independently	3 pts Unsatisfactory | Begins to look beyond classroom requirements, showing interest in pursuing knowledge independently	2 pts Poor | Relies on classroom instruction only	1 pts Very Poor | No initiative or interest in acquiring new knowledge	6 pts
SO 7 PI 2 **Student Outcome 7.2** Learn independently threshold: 4.8 pts	6 pts Excellent | Completes an assigned task independently and practices continuous improvement	5 pts Good | Completes an assigned task without supervision or guidance	4 pts Satisfactory | Requires minimal guidance to complete an assigned task	3 pts Unsatisfactory | Requires detailed or step-by-step instructions to complete a task	2 pts Poor | Shows little interest to complete a task independently	1 pts Very Poor | No interest to complete a task independently	6 pts
SO 7 PI 3 **Student Outcome 7.3** Critical thinking in the broadest context of technological change threshold: 4.8 pts	6 pts Excellent | Synthesizes and integrates information from a variety of sources; formulates a clear and precise perspective; draws appropriate conclusions	5 pts Good | Evaluate information from a variety of sources; formulates a clear and precise perspective.	4 pts Satisfactory | Analyze information from a variety of sources; formulates a clear and precise perspective.	3 pts Unsatisfactory | Apply the gathered information to formulate the problem	2 pts Poor | Gather and summarized the information from a variety of sources but failed to formulate the problem	1 pts Very Poor | Gather information from a variety of sources	6 pts
SO 7 PI 4 **Student Outcome 7.4** Creativity and adaptability to new and emerging technologies threshold: 4.8 pts	6 pts Excellent | Ideas are combined in original and creative ways in line with the new and emerging technology trends to solve a problem or address an issue.	5 pts Good | Ideas are creative and adapt the new knowledge to solve a problem or address an issue	4 pts Satisfactory | Ideas are creative in solving a problem, or address an issue	3 pts Unsatisfactory | Shows some creative ways to solve the problem	2 pts Poor | Shows initiative and attempt to develop creative ideas to solve the problem	1 pts Very Poor | Ideas are copied or restated from the sources consulted	6 pts
							Total Points: 24