# Task-1

**Aim:**

Variables and Data Types

Declare a variable using var, let, and const. Assign different data types to each variable and print their values.

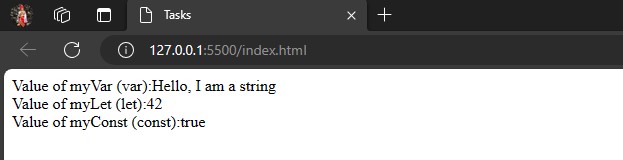
**Description:**

**var** variables can be redeclared and reassigned**, let** variables can be reassigned but not redeclared in the same scope, and **const** variables cannot be redeclared or reassigned.

**Source Code:**

|  |
| --- |
| // Using var var myVar = "Hello,I  am a string";  console.log("Value of myVar (var):", myVar);    // Using let let myLet = 42; console.log("Value of myLet (let):", myLet);    // Using const const myConst = true; console.log("Value of myConst (const):", myConst); |

**Output:**



**Learning Outcome:**

Overall, the provided code examples cover various important aspects of web development, including HTML structure, event handling, error management, and asynchronous JavaScript.

# Task-2

**Aim:**

Operators and Expressions

Write a function that takes two numbers as arguments and returns their sum, difference, product, and quotient using arithmetic operators.

**Description:**

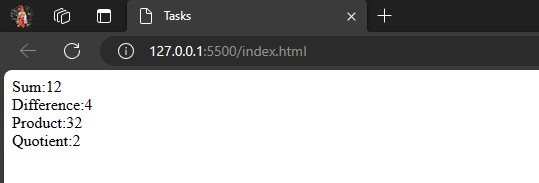
JavaScript operators can be categorized into two main categories

i.e., *Unary* and *Binary* operators. Unary takes only one operand whereas binary takes two. Anything that evaluates to a value is called an *expression.*

**Source Code:**

|  |
| --- |
| function calculateOperations(num1, num2) {  const sum = num1 + num2;  const difference = num1 - num2;  const product = num1 \* num2;  const quotient = num1 / num2;  document.write("Sum:", sum+"<br>");  document.write("Difference:", difference+"<br>");  document.write("Product:", product+"<br>");  document.write("Quotient:", quotient+"<br>");  } calculateOperations(8, 4); |

**Output:**



**Learning Outcome:**

Develop problem-solving skills by applying operators and expressions to solve mathematical problems, manipulate data, and implement algorithms in JavaScript

# Task-3

**Aim:**

Control Flow

Write a program that prompts the user to enter their age. Based on their age, display different messages:

○ If the age is less than 18, display "You are a minor."

○ If the age is between 18 and 65, display "You are an adult."

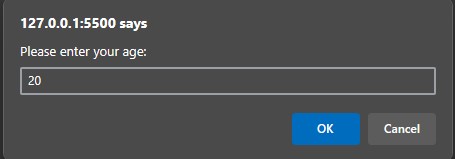
○ If the age is 65 or older, display "You are a senior citizen." **Description:**

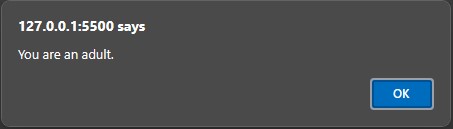
Control flow in JavaScript refers to the order in which statements are executed in a program. It is managed through conditional statements and loops, allowing developers to create dynamic and responsive code.

**Source Code:**

|  |
| --- |
| // Prompt the user to enter their age  const userAge = prompt("Please enter your age:");    // Convert the entered age to a number  const age = parseInt(userAge);    // Check the age and display the appropriate message if (age < 18) {  alert("You are a minor.");  }  else if (age >= 18 && age <= 65) {  alert("You are an adult.");  }  else {  alert("You are a senior citizen.");  } |

**Output:**





**Learning Outcome:**

Conditional Logic Mastery:

Understand how to use if, else if, and else statements to control the flow of execution based on different conditions.

B

# Task-4

**Aim:**

Functions

Write a function that takes an array of salary as an argument and returns the min/max salary in the array.

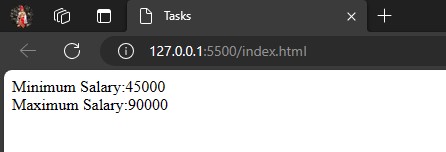
**Description:**

The findMinMaxSalary function uses the Math.min and Math.max functions along with the spread operator (...) to find the minimum and maximum salaries in the given array. It also includes a check for an empty array.

**Source Code:**

|  |
| --- |
| function findMinMaxSalary(salaries) { if (salaries.length === 0) { return "The array is empty.";  } const minSalary = Math.min(...salaries);  const maxSalary = Math.max(...salaries);  return{  min: minSalary,  max: maxSalary,  };  }    // Example usage:  const salariesArray = [50000, 75000, 60000, 90000, 45000];  const result = findMinMaxSalary(salariesArray);    // Output in different lines  document.write("Minimum Salary:", result.min+"<br>"); document.write("Maximum Salary:", result.max+"<br>"); |
|  |

**Output:**



**Learning Outcome:**

Functions are a fundamental concept in JavaScript, facilitating the creation of efficient, modular, and maintainable code.

# Task-5

**Aim:**

Arrays and Objects

Create an array of your favorite books. Write a function that takes the array as an argument and displays each book title on a separate line.

**Description:**

The program uses an array favoriteBooks to store a list of favorite book titles. The displayBooks function takes an array as an argument and uses a for...of loop to iterate through the array, printing each book title on a separate line.

**Source Code:**

// Array of favorite books const favoriteBooks = ["The Great Gatsby", "To Kill a Mockingbird", "1984",

"Pride and Prejudice", "The Catcher in the Rye"];

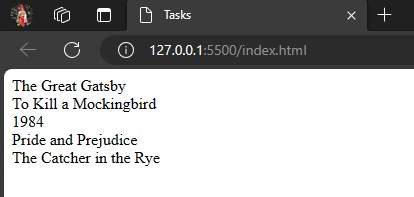
// Function to display each book title on a separate line function displayBooks(books) { for (const book of books) { document.write(book+"<br>");

}

}

// Calling the function with the array of favorite books displayBooks(favoriteBooks);

**Output:**



**Learning Outcome:**

Understanding arrays and objects is foundational for managing and manipulating data in JavaScript, allowing developers to build organized and efficient programs.

# Task-6

**Aim:**

Scope and Hoisting

Declare a variable inside a function and try to access it outside the function.

Observe the

scope behavior and explain the results. [var vs let vs const]

Description:

1.var Variable:

Variables declared with var are function-scoped, meaning they are only accessible within the function where they are declared or globally if declared outside any function. 2.let Variable:

Variables declared with let have block scope, meaning they are only accessible within the block (in this case, the function) where they are declared.

3.const Variable:

Variables declared with const are also block-scoped. They have the added feature of being constants, meaning their values cannot be reassigned after declaration.

**Source Code:**

|  |
| --- |
| function exampleFunction() {  // Using var var variableVar = "I am a var variable.";    // Using let let variableLet = "I am a let variable.";  // Using const const variableConst = "I am a const variable.";  }    // Trying to access variables outside the function  document.write(variableVar); // Will result in an error with "variableVar is not defined"  document.write(variableLet); // Will result in an error with "variableLet is not defined" document.write(variableConst); // Will result in an error with "variableConst is not defined" |

**Output:**

-NOTHING

**Learning Outcome:**

Understanding scope and hoisting is crucial for writing predictable and maintainable code in JavaScript. Using let and const helps developers avoid common pitfalls associated with variable declarations using var.

# Task-7

**Aim:**

DOM Manipulation

Create an HTML page with a button. Write JavaScript code that adds an event listener to the button and changes its text when clicked.

**Description:**

JavaScript DOM Manipulation:

The JavaScript file (script.js) uses document.getElementById to obtain a reference to the button element.

An event listener is added to the button using addEventListener. This listener responds to the "click" event.

**Source Code:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Tasks</title>

</head>

<body>

<button id="myButton">Click me</button>

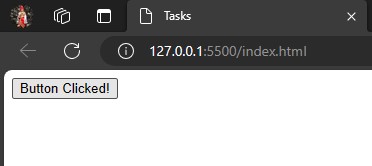
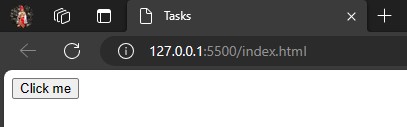
<script src="java7.js"></script>

</body>

</html>

|  |
| --- |
| // Get the button element by its ID const myButton = document.getElementById("myButton");    // Add an event listener to the button  myButton.addEventListener("click", function() {  // Change the text content of the button when clicked myButton.textContent = "Button Clicked!";  }); |

**Output:**



**Learning Outcome:**

DOM manipulation is a fundamental skill for web developers, allowing them to create dynamic and interactive websites that respond to user input and enhance user engagement.

# Task-8

**Aim:**

Error Handling

Write a function that takes a number as an argument and throws an error if the number is negative. Handle the error and display a custom error message.

Description:

The checkPositiveNumber function now takes userInput as a parameter, converts it to a number using parseFloat, and checks if it's negative.

If the input is not a valid number (e.g., if the user enters a non-numeric value), it throws an error with a custom error message.

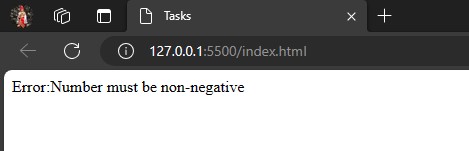
The example usage prompts the user to enter a number and then calls the function, handling any errors with a try...catch block

**Source Code:**

|  |
| --- |
| function checkPositiveNumber(userInput) { const number = parseFloat(userInput);  if (isNaN(number)) { throw new Error("Invalid input. Please enter a valid number.");  } if (number < 0) {  throw new Error("Number must be non-negative");  }    // If the number is not negative, continue with the rest of the code return Math.sqrt(number); // For example, calculating the square root of the non-negative number  }    // Example usage with user input |
| try { const userInput = prompt("Enter a number:"); const result = checkPositiveNumber(userInput); document.write("Result:", result); } catch (error) {  document.write("Error:", error.message);  } |

**Output:**





**Learning Outcome:**

Handling user input and potential errors is crucial in real-world applications to create a robust and user-friendly experience. Proper error handling ensures that the application gracefully handles unexpected scenarios and provides helpful feedback to users.

# Task-9

**Aim:**

Asynchronous JavaScript

Write a function that uses setTimeout to simulate an asynchronous operation. Use a callback function to handle the result.

**Description:**

The simulateAsyncOperation function simulates an asynchronous operation using setTimeout. It logs a message and simulates a 2-second delay.

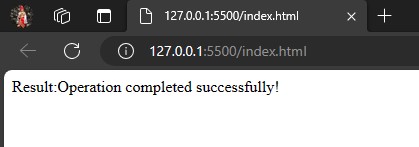
After the delay, it invokes the provided callback function with null as the error (indicating success) and a result message.

The example usage demonstrates calling simulateAsyncOperation with a callback function that handles the result or any potential error.

**Source Code:**

|  |
| --- |
| function simulateAsyncOperation(callback) { console.log("Simulating an asynchronous operation...");  // Simulate a delay of 2 seconds (2000 milliseconds) setTimeout(function() { const result = "Operation completed successfully!"; // Invoke the callback function with the result callback(null, result);  }, 2000);  }    // Example usage with a callback function  simulateAsyncOperation(function(error,result) {  if (error)  {  document.error("Error:", error);  }  else {  document.write("Result:", result);  }  } |
|  |

**Output:**



**Learning Outcome:**

Asynchronous JavaScript is essential for building responsive and efficient web applications. Callback functions are a fundamental aspect of handling asynchronous operations, providing a way to structure code and manage the flow of execution in scenarios where timing is uncertain.