Day 1: Variables and Data Types

Tasks/Activities:

Activity 1: Variable Declaration

- **Task 1:** Declare a variable using `var`, assign it a number, and log the value to the console.
- **Task 2:** Declare a variable using `let`, assign it a string, and log the value to the console.

Activity 2: Constant Declaration

- **Task 3:** Declare a variable using `const`, assign it a boolean value, and log the value to the console.

Activity 3: Data Types

- **Task 4:** Create variables of different data types (number, string, boolean, object, array) and log each variable's type using the `typeof` operator.

Activity 4: Reassigning Variables

- **Task 5:** Declare a variable using `let`, assign it an initial value, reassign a new value, and log both values to the console.

Activity 5: Understanding `const`

- **Task 6:** Try reassigning a variable declared with `const` and observe the error.

Feature Request:

1. **Variable Types Console Log:** Write a script that declares variables of different data types and logs both the value and type of each variable to the console.
2. **Reassignment Demo:** Create a script that demonstrates the difference in behavior between `let` and `const` when it comes to reassignment.

Achievement:

By the end of these activities, you will:

- Know how to declare variables using `var`, `let`, and `const`.
- Understand the different data types in JavaScript.
- Be able to use the `typeof` operator to identify the data type of a variable.
- Understand the concept of variable reassignment and the immutability of `const` variables.

Day 2: Operators

Tasks/Activities:

Activity 1: Arithmetic Operations

- **Task 1:** Write a program to add two numbers and log the result to the console.
- **Task 2:** Write a program to subtract two numbers and log the result to the console.
- **Task 3:** Write a program to multiply two numbers and log the result to the console.
- **Task 4:** Write a program to divide two numbers and log the result to the console.
- **Task 5:** Write a program to find the remainder when one number is divided by another and log the result to the console.

Activity 2: Assignment Operators

- **Task 6:** Use the `+=` operator to add a number to a variable and log the result to the console.
- **Task 7:** Use the `-=` operator to subtract a number from a variable and log the result to the console.

Activity 3: Comparison Operators

- **Task 8:** Write a program to compare two numbers using `>` and `<` and log the result to the console.
- **Task 9:** Write a program to compare two numbers using `>=` and `<=` and log the result to the console.
- **Task 10:** Write a program to compare two numbers using `==` and `===` and log the result to the console.

Activity 4: Logical Operators

- **Task 11:** Write a program that uses the `&&` operator to combine two conditions and log the result to the console.
- **Task 12:** Write a program that uses the `||` operator to combine two conditions and log the result to the console.
- **Task 13:** Write a program that uses the `!` operator to negate a condition and log the result to the console.

Activity 5: Ternary Operator

- **Task 14:** Write a program that uses the ternary operator to check if a number is positive or negative and log the result to the console.

Feature Request:

1. **Arithmetic Operations Script:** Write a script that performs basic arithmetic operations (addition, subtraction, multiplication, division, remainder) on two numbers and logs the results.
2. **Comparison and Logical Operators Script:** Create a script that compares two numbers using different comparison operators and combines conditions using logical operators, logging the results.
3. **Ternary Operator Script:** Write a script that uses the ternary operator to determine if a number is positive or negative and logs the result.

Achievement:

By the end of these activities, students will:

- Understand and use arithmetic operators to perform basic calculations.
- Use assignment operators to modify variable values.
- Compare values using comparison operators.
- Combine conditions using logical operators.
- Use the ternary operator for concise conditional expressions.

Day 3: Control Structures

Tasks/Activities:

Activity 1: If-Else Statements

- **Task 1:** Write a program to check if a number is positive, negative, or zero, and log the result to the console.
- **Task 2:** Write a program to check if a person is eligible to vote (age >= 18) and log the result to the console.

Activity 2: Nested If-Else Statements

- **Task 3:** Write a program to find the largest of three numbers using nested if-else statements.

Activity 3: Switch Case

- **Task 4:** Write a program that uses a switch case to determine the day of the week based on a number (1-7) and log the day name to the console.
- **Task 5:** Write a program that uses a switch case to assign a grade ('A', 'B', 'C', 'D', 'F') based on a score and log the grade to the console.

Activity 4: Conditional (Ternary) Operator

- **Task 6:** Write a program that uses the ternary operator to check if a number is even or odd and log the result to the console.

Activity 5: Combining Conditions

- **Task 7:** Write a program to check if a year is a leap year using multiple conditions (divisible by 4, but not 100 unless also divisible by 400) and log the result to the console.

Feature Request:

1. **Number Check Script:** Write a script that checks if a number is positive, negative, or zero using if-else statements and logs the result.
2. **Voting Eligibility Script:** Create a script to check if a person is eligible to vote based on their age and log the result.
3. **Day of the Week Script:** Write a script that uses a switch case to determine the day of the week based on a number (1-7) and logs the day name.
4. **Grade Assignment Script:** Create a script that uses a switch case to assign a grade based on a score and logs the grade.
5. **Leap Year Check Script:** Write a script that checks if a year is a leap year using multiple conditions and logs the result.

Achievement:

By the end of these activities, students will:

- Implement and understand basic if-else control flow.
- Use nested if-else statements to handle multiple conditions.
- Utilize switch cases for control flow based on specific values.
- Apply the ternary operator for concise condition checking.
- Combine multiple conditions to solve more complex problems.

Day 4: Loops

Tasks/Activities:

Activity 1: For Loop

- **Task 1:** Write a program to print numbers from 1 to 10 using a for loop.
- **Task 2:** Write a program to print the multiplication table of 5 using a for loop.

Activity 2: While Loop

- **Task 3:** Write a program to calculate the sum of numbers from 1 to 10 using a while loop.
- **Task 4:** Write a program to print numbers from 10 to 1 using a while loop.

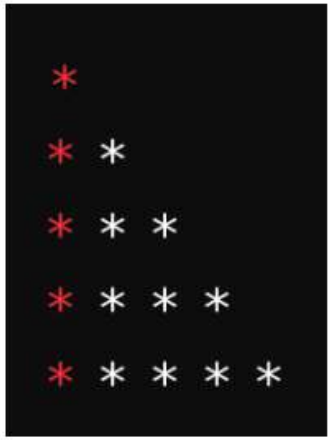
Activity 3: Do...While Loop

- **Task 5:** Write a program to print numbers from 1 to 5 using a do...while loop.
- **Task 6:** Write a program to calculate the factorial of a number using a do...while loop.

Activity 4: Nested Loops

- **Task 7:** Write a program to print a pattern using nested for loops:

(ignore color)



Activity 5: Loop Control Statements

- **Task 8:** Write a program to print numbers from 1 to 10, but skip the number 5 using the `continue` statement.
- **Task 9:** Write a program to print numbers from 1 to 10, but stop the loop when the number is 7 using the `break` statement.

Feature Request:

1. **Number Printing Script:** Write a script that prints numbers from 1 to 10 using a for loop and a while loop.
2. **Multiplication Table Script:** Create a script that prints the multiplication table of 5 using a for loop.
3. **Pattern Printing Script:** Write a script that prints a pattern of stars using nested loops.
4. **Sum Calculation Script:** Write a script that calculates the sum of numbers from 1 to 10 using a while loop.
5. **Factorial Calculation Script:** Create a script that calculates the factorial of a number using a do...while loop.

Achievement:

By the end of these activities, students will:

- Understand and use for loops to iterate over a sequence of numbers.
- Utilize while loops for iteration based on a condition.
- Apply do...while loops to ensure the loop body is executed at least once.
- Implement nested loops to solve more complex problems.
- Use loop control statements (`break` and `continue`) to control the flow of loops.

Day 5: Functions

Tasks/Activities:

Activity 1: Function Declaration

- **Task 1:** Write a function to check if a number is even or odd and log the result to the console.
- **Task 2:** Write a function to calculate the square of a number and return the result.

Activity 2: Function Expression

- **Task 3:** Write a function expression to find the maximum of two numbers and log the result to the console.
- **Task 4:** Write a function expression to concatenate two strings and return the result.

Activity 3: Arrow Functions

- **Task 5:** Write an arrow function to calculate the sum of two numbers and return the result.
- **Task 6:** Write an arrow function to check if a string contains a specific character and return a boolean value.

Activity 4: Function Parameters and Default Values

- **Task 7:** Write a function that takes two parameters and returns their product. Provide a default value for the second parameter.
- **Task 8:** Write a function that takes a person's name and age and returns a greeting message. Provide a default value for the age.

Activity 5: Higher-Order Functions

- **Task 9:** Write a higher-order function that takes a function and a number, and calls the function that many times.
- **Task 10:** Write a higher-order function that takes two functions and a value, applies the first function to the value, and then applies the second function to the result.

Feature Request:

1. **Even or Odd Function Script:** Write a script that includes a function to check if a number is even or odd and logs the result.
2. **Square Calculation Function Script:** Create a script that includes a function to calculate the square of a number and returns the result.
3. **Concatenation Function Script:** Write a script that includes a function expression to concatenate two strings and returns the result.
4. **Sum Calculation Arrow Function Script:** Create a script that includes an arrow function to calculate the sum of two numbers and returns the result.
5. **Higher-Order Function Script:** Write a script that includes a higher-order function to apply a given function multiple times.

Achievement:

By the end of these activities, students will:

- Understand and define functions using function declarations, expressions, and arrow functions.
- Use function parameters and default values effectively.
- Create and utilize higher-order functions.
- Apply functions to solve common problems and perform calculations.
- Enhance code reusability and organization using functions.

Day 6: Arrays

Tasks/Activities:

Activity 1: Array Creation and Access

- Task 1: Create an array of numbers from 1 to 5 and log the array to the console.
- Task 2: Access the first and last elements of the array and log them to the console.

Activity 2: Array Methods (Basic)

- Task 3: Use the `push` method to add a new number to the end of the array and log the updated array.
- Task 4: Use the `pop` method to remove the last element from the array and log the updated array.
- Task 5: Use the `shift` method to remove the first element from the array and log the updated array.
- Task 6: Use the `unshift` method to add a new number to the beginning of the array and log the updated array.

Activity 3: Array Methods (Intermediate)

- Task 7: Use the `map` method to create a new array where each number is doubled and log the new array.
- Task 8: Use the `filter` method to create a new array with only even numbers and log the new array.
- Task 9: Use the `reduce` method to calculate the sum of all numbers in the array and log the result.

Activity 4: Array Iteration

- Task 10: Use a `for` loop to iterate over the array and log each element to the console.
- Task 11: Use the `forEach` method to iterate over the array and log each element to the console.

Activity 5: Multi-dimensional Arrays

- Task 12: Create a two-dimensional array (matrix) and log the entire array to the console.
- Task 13: Access and log a specific element from the two-dimensional array.

Feature Request:

1. **Array Manipulation Script:** Write a script that demonstrates the creation of an array, adding and removing elements using `push`, `pop`, `shift`, and `unshift` methods.
2. **Array Transformation Script:** Create a script that uses `map`, `filter`, and `reduce` methods to transform and aggregate array data.
3. **Array Iteration Script:** Write a script that iterates over an array using both `for` loop and `forEach` method and logs each element.
4. **Two-dimensional Array Script:** Create a script that demonstrates the creation and manipulation of a two-dimensional array.

Achievement:

By the end of these activities, students will:

- Create and manipulate arrays using various methods.
- Transform and aggregate array data using `map`, `filter`, and `reduce`.
- Iterate over arrays using loops and iteration methods.
- Understand and work with multi-dimensional arrays.

Day 7: Objects

Tasks/Activities:

Activity 1: Object Creation and Access

- **Task 1:** Create an object representing a book with properties like title, author, and year, and log the object to the console.
- **Task 2:** Access and log the title and author properties of the book object.

Activity 2: Object Methods

- **Task 3:** Add a method to the book object that returns a string with the book's title and author, and log the result of calling this method.
- **Task 4:** Add a method to the book object that takes a parameter (year) and updates the book's year property, then log the updated object.

Activity 3: Nested Objects

- **Task 5:** Create a nested object representing a library with properties like name and books (an array of book objects), and log the library object to the console.
- **Task 6:** Access and log the name of the library and the titles of all the books in the library.

Activity 4: The `this` Keyword

- **Task 7:** Add a method to the book object that uses the `this` keyword to return a string with the book's title and year, and log the result of calling this method.

Activity 5: Object Iteration

- **Task 8:** Use a `for...in` loop to iterate over the properties of the book object and log each property and its value.
- **Task 9:** Use `Object.keys` and `Object.values` methods to log all the keys and values of the book object.

Feature Request:

1. **Book Object Script:** Write a script that creates a book object, adds methods to it, and logs its properties and method results.
2. **Library Object Script:** Create a script that defines a library object containing an array of book objects and logs the library's details.
3. **Object Iteration Script:** Write a script that demonstrates iterating over an object's properties using `for...in` loop and `Object.keys` / `Object.values`.

Achievement:

By the end of these activities, students will:

- Create and manipulate objects with properties and methods.
- Understand and use the `this` keyword in object methods.
- Work with nested objects and arrays of objects.
- Iterate over an object's properties using loops and built-in methods.

Day 8: ES6+ Features

Tasks/Activities:

Activity 1: Template Literals

- Task 1: Use template literals to create a string that includes variables for a person's name and age, and log the string to the console.
- Task 2: Create a multi-line string using template literals and log it to the console.

Activity 2: Destructuring

- Task 3: Use array destructuring to extract the first and second elements from an array of numbers and log them to the console.
- Task 4: Use object destructuring to extract the title and author from a book object and log them to the console.

Activity 3: Spread and Rest Operators

- Task 5: Use the spread operator to create a new array that includes all elements of an existing array plus additional elements, and log the new array to the console.
- Task 6: Use the rest operator in a function to accept an arbitrary number of arguments, sum them, and return the result.

Activity 4: Default Parameters

- Task 7: Write a function that takes two parameters and returns their product, with the second parameter having a default value of 1. Log the result of calling this function with and without the second parameter.

Activity 5: Enhanced Object Literals

- Task 8: Use enhanced object literals to create an object with methods and properties, and log the object to the console.
- Task 9: Create an object with computed property names based on variables and log the object to the console.

Feature Request:

1. **Template Literals Script**: Write a script that demonstrates the use of template literals to create and log strings with embedded variables and multi-line strings.
2. **Destructuring Script**: Create a script that uses array and object destructuring to extract values and log them.
3. **Spread and Rest Operators Script**: Write a script that demonstrates the use of the spread operator to combine arrays and the rest operator to handle multiple function arguments.
4. **Default Parameters Script**: Create a script that defines a function with default parameters and logs the results of calling it with different arguments.
5. **Enhanced Object Literals Script**: Write a script that uses enhanced object literals to create and log an object with methods and computed property names.

Achievement:

By the end of these activities, students will:

- Understand and use template literals for string interpolation and multi-line strings.
- Apply destructuring to extract values from arrays and objects.
- Utilize spread and rest operators for array manipulation and function arguments.
- Define functions with default parameters.
- Create objects using enhanced object literals, including methods and computed property names.

Day 9: DOM Manipulation

Tasks/Activities:

Activity 1: Selecting and Manipulating Elements

- Task 1: Select an HTML element by its ID and change its text content.
- Task 2: Select an HTML element by its class and change its background color.

Activity 2: Creating and Appending Elements

- Task 3: Create a new `div` element with some text content and append it to the body.
- Task 4: Create a new `li` element and add it to an existing `ul` list.

Activity 3: Removing Elements

- Task 5: Select an HTML element and remove it from the DOM.
- Task 6: Remove the last child of a specific HTML element.

Activity 4: Modifying Attributes and Classes

- Task 7: Select an HTML element and change one of its attributes (e.g., `src` of an `img` tag).
- Task 8: Add and remove a CSS class to/from an HTML element.

Activity 5: Event Handling

- Task 9: Add a click event listener to a button that changes the text content of a paragraph.
- Task 10: Add a mouseover event listener to an element that changes its border color.

Feature Request:

1. **Text Content Manipulation Script**: Write a script that selects an HTML element by its ID and changes its text content.
2. **Element Creation Script**: Create a script that demonstrates creating a new `div` element and appending it to the body.
3. **Element Removal Script**: Write a script that selects an HTML element and removes it from the DOM.
4. **Attribute Modification Script**: Create a script that changes the attributes of an HTML element.
5. **Event Handling Script**: Write a script that adds event listeners to HTML elements to change their content or style based on user interactions.

Achievement:

By the end of these activities, students will:

- Select and manipulate DOM elements using JavaScript.
- Create and append new elements to the DOM.
- Remove elements from the DOM.
- Modify attributes and classes of HTML elements.
- Add and handle events to make web pages interactive.

Day 10: Event Handling

Tasks/Activities:

Activity 1: Basic Event Handling

- **Task 1:** Add a click event listener to a button that changes the text content of a paragraph.
- **Task 2:** Add a double-click event listener to an image that toggles its visibility.

Activity 2: Mouse Events

- **Task 3:** Add a mouseover event listener to an element that changes its background color.
- **Task 4:** Add a mouseout event listener to an element that resets its background color.

Activity 3: Keyboard Events

- **Task 5:** Add a keydown event listener to an input field that logs the key pressed to the console.
- **Task 6:** Add a keyup event listener to an input field that displays the current value in a paragraph.

Activity 4: Form Events

- **Task 7:** Add a submit event listener to a form that prevents the default submission and logs the form data to the console.
- **Task 8:** Add a change event listener to a select dropdown that displays the selected value in a paragraph.

Activity 5: Event Delegation

- **Task 9:** Add a click event listener to a list that logs the text content of the clicked list item using event delegation.
- **Task 10:** Add an event listener to a parent element that listens for events from dynamically added child elements.

Feature Request:

1. **Click Event Script:** Write a script that adds a click event listener to a button to change the text content of a paragraph.
2. **Mouse Events Script:** Create a script that handles mouseover and mouseout events to change the background color of an element.
3. **Keyboard Events Script:** Write a script that logs key presses and displays input field values using keydown and keyup event listeners.
4. **Form Events Script:** Create a script that handles form submission and change events on a select dropdown.
5. **Event Delegation Script:** Write a script that demonstrates event delegation by handling events on dynamically added child elements.

Achievement:

By the end of these activities, students will:

- Add and handle basic events like click, double-click, mouseover, mouseout, keydown, and keyup.
- Understand and handle form events.
- Implement event delegation to manage events on dynamically added elements.
- Make web pages interactive by responding to various user actions.

Day 11: Promises and Async/Await

Tasks/Activities:

Activity 1: Understanding Promises

- Task 1: Create a promise that resolves with a message after a 2-second timeout and log the message to the console.
- Task 2: Create a promise that rejects with an error message after a 2-second timeout and handle the error using `.catch()`.

Activity 2: Chaining Promises

- Task 3: Create a sequence of promises that simulate fetching data from a server. Chain the promises to log messages in a specific order.

Activity 3: Using Async/Await

- Task 4: Write an async function that waits for a promise to resolve and then logs the resolved value.
- Task 5: Write an async function that handles a rejected promise using try-catch and logs the error message.

Activity 4: Fetching Data from an API

- Task 6: Use the `fetch` API to get data from a public API and log the response data to the console using promises.
- Task 7: Use the `fetch` API to get data from a public API and log the response data to the console using async/await.

Activity 5: Concurrent Promises

- Task 8: Use `Promise.all` to wait for multiple promises to resolve and then log all their values.
- Task 9: Use `Promise.race` to log the value of the first promise that resolves among multiple promises.

Feature Request:

1. **Promise Creation Script**: Write a script that demonstrates creating and handling promises, including both resolved and rejected states.
2. **Promise Chaining Script**: Create a script that chains multiple promises and logs messages in a specific sequence.
3. **Async/Await Script**: Write a script that uses async/await to handle promises and includes error handling with try-catch.
4. **API Fetch Script**: Create a script that fetches data from a public API using both promises and async/await, and logs the response data.
5. **Concurrent Promises Script**: Write a script that uses `Promise.all` and `Promise.race` to handle multiple promises concurrently and logs the results.

Achievement:

By the end of these activities, students will:

- Understand and create promises, including handling resolved and rejected states.
- Chain multiple promises to perform sequential asynchronous operations.
- Use async/await to handle asynchronous code more readably.
- Fetch data from public APIs using both promises and async/await.
- Manage multiple concurrent promises using `Promise.all` and `Promise.race`.

Day 12: Error Handling

Tasks/Activities:

Activity 1: Basic Error Handling with Try-Catch

- **Task 1:** Write a function that intentionally throws an error and use a try-catch block to handle the error and log an appropriate message to the console.
- **Task 2:** Create a function that divides two numbers and throws an error if the denominator is zero. Use a try-catch block to handle this error.

Activity 2: Finally Block

- **Task 3:** Write a script that includes a try-catch block and a finally block. Log messages in the try, catch, and finally blocks to observe the execution flow.

Activity 3: Custom Error Objects

- **Task 4:** Create a custom error class that extends the built-in Error class. Throw an instance of this custom error in a function and handle it using a try-catch block.
- **Task 5:** Write a function that validates user input (e.g., checking if a string is not empty) and throws a custom error if the validation fails. Handle the custom error using a try-catch block.

Activity 4: Error Handling in Promises

- **Task 6:** Create a promise that randomly resolves or rejects. Use `.catch()` to handle the rejection and log an appropriate message to the console.
- **Task 7:** Use try-catch within an async function to handle errors from a promise that randomly resolves or rejects, and log the error message.

Activity 5: Graceful Error Handling in Fetch

- **Task 8:** Use the `fetch` API to request data from an invalid URL and handle the error using `.catch()`. Log an appropriate error message to the console.
- **Task 9:** Use the `fetch` API to request data from an invalid URL within an async function and handle the error using try-catch. Log an appropriate error message.

Feature Request:

1. **Basic Error Handling Script:** Write a script that demonstrates basic error handling using try-catch and finally blocks.
2. **Custom Error Script:** Create a script that defines and throws custom errors, handling them with try-catch blocks.
3. **Promise Error Handling Script:** Write a script that handles errors in promises using `.catch()` and try-catch within async functions.
4. **Fetch Error Handling Script:** Create a script that handles errors when using the `fetch` API to request data from invalid URLs.

Achievement:

By the end of these activities, students will:

- Understand and implement basic error handling using try-catch blocks.
- Use finally blocks to execute code regardless of the try-catch outcome.
- Create and use custom error classes.
- Handle errors in promises using `.catch()` and within async functions using try-catch.
- Implement graceful error handling when making network requests with the `fetch` API.

Day 13: Modules

Tasks/Activities:

Activity 1: Creating and Exporting Modules

- **Task 1:** Create a module that exports a function to add two numbers. Import and use this module in another script.
- **Task 2:** Create a module that exports an object representing a person with properties and methods. Import and use this module in another script.

Activity 2: Named and Default Exports

- **Task 3:** Create a module that exports multiple functions using named exports. Import and use these functions in another script.
- **Task 4:** Create a module that exports a single function using default export. Import and use this function in another script.

Activity 3: Importing Entire Modules

- **Task 5:** Create a module that exports multiple constants and functions. Import the entire module as an object in another script and use its properties.

Activity 4: Using Third-Party Modules

- **Task 6:** Install a third-party module (e.g., `lodash`) using npm. Import and use a function from this module in a script.
- **Task 7:** Install a third-party module (e.g., `axios`) using npm. Import and use this module to make a network request in a script.

Activity 5: Module Bundling (Optional)

- **Task 8:** Use a module bundler like Webpack or Parcel to bundle multiple JavaScript files into a single file. Write a script to demonstrate the bundling process.

Feature Request:

1. **Basic Module Script**: Write a script that creates a module exporting a function and imports it in another script.
2. **Named and Default Exports Script**: Create a script demonstrating both named and default exports and their usage.
3. **Third-Party Module Script**: Write a script that installs, imports, and uses functions from third-party modules like `lodash` and `axios`.
4. **Module Bundling Script**: Create a script demonstrating how to bundle JavaScript files using a module bundler (optional).

Achievement:

By the end of these activities, students will:

- Create and export functions, objects, and constants using modules.
- Import modules using named and default imports.
- Use third-party modules installed via npm.
- Understand the basics of module bundling (optional).

Day 14: Classes

Tasks/Activities:

Activity 1: Class Definition

- Task 1: Define a class `Person` with properties `name` and `age`, and a method to return a greeting message. Create an instance of the class and log the greeting message.
- Task 2: Add a method to the `Person` class that updates the age property and logs the updated age.

Activity 2: Class Inheritance

- Task 3: Define a class `Student` that extends the `Person` class. Add a property `studentId` and a method to return the student ID. Create an instance of the `Student` class and log the student ID.
- Task 4: Override the greeting method in the `Student` class to include the student ID in the message. Log the overridden greeting message.

Activity 3: Static Methods and Properties

- Task 5: Add a static method to the `Person` class that returns a generic greeting message. Call this static method without creating an instance of the class and log the message.
- Task 6: Add a static property to the `Student` class to keep track of the number of students created. Increment this property in the constructor and log the total number of students.

Activity 4: Getters and Setters

- Task 7: Add a getter method to the `Person` class to return the full name (assume a `firstName` and `lastName` property). Create an instance and log the full name using the getter.
- Task 8: Add a setter method to the `Person` class to update the name properties (`firstName` and `lastName`). Update the name using the setter and log the updated full name.

Activity 5: Private Fields (Optional)

- Task 9: Define a class `Account` with private fields for `balance` and a method to deposit and withdraw money. Ensure that the balance can only be updated through these methods.
- Task 10: Create an instance of the `Account` class and test the deposit and withdraw methods, logging the balance after each operation.

Feature Request:

1. **Basic Class Script**: Write a script that defines a `Person` class with properties and methods, creates instances, and logs messages.
2. **Class Inheritance Script**: Create a script that defines a `Student` class extending `Person`, overrides methods, and logs the results.
3. **Static Methods and Properties Script**: Write a script that demonstrates static methods and properties in classes.
4. **Getters and Setters Script**: Create a script that uses getters and setters in a class.
5. **Private Fields Script**: Write a script that defines a class with private fields and methods to manipulate these fields (optional).

Achievement:

By the end of these activities, students will:

- Define and use classes with properties and methods.
- Implement inheritance to extend classes.
- Utilize static methods and properties.
- Apply getters and setters for encapsulation.
- Understand and use private fields in classes (optional).

Day 15: Closures

Tasks/Activities:

Activity 1: Understanding Closures

- **Task 1:** Write a function that returns another function, where the inner function accesses a variable from the outer function's scope. Call the inner function and log the result.
- **Task 2:** Create a closure that maintains a private counter. Implement functions to increment and get the current value of the counter.

Activity 2: Practical Closures

- **Task 3:** Write a function that generates unique IDs. Use a closure to keep track of the last generated ID and increment it with each call.
- **Task 4:** Create a closure that captures a user's name and returns a function that greets the user by name.

Activity 3: Closures in Loops

- **Task 5:** Write a loop that creates an array of functions. Each function should log its index when called. Use a closure to ensure each function logs the correct index.

Activity 4: Module Pattern

- **Task 6:** Use closures to create a simple module for managing a collection of items. Implement methods to add, remove, and list items.

Activity 5: Memoization

- **Task 7:** Write a function that memoizes the results of another function. Use a closure to store the results of previous computations.
- **Task 8:** Create a memoized version of a function that calculates the factorial of a number.

Feature Request:

1. **Basic Closure Script**: Write a script that demonstrates a basic closure with a function returning another function that accesses the outer function's variable.
2. **Counter Closure Script**: Create a script that uses a closure to maintain a private counter with increment and get functions.
3. **Unique ID Generator Script**: Write a script that generates unique IDs using a closure to keep track of the last generated ID.
4. **Loop Closure Script**: Create a script that demonstrates closures in loops to ensure functions log the correct index.
5. **Memoization Script**: Write a script that memoizes the results of a function and demonstrates it with a factorial calculation.

Achievement:

By the end of these activities, students will:

- Understand and create closures in JavaScript.
- Use closures to maintain private state and create encapsulated modules.
- Apply closures in practical scenarios like generating unique IDs and memoization.
- Use closures in loops to capture and use variables correctly.

Day 16: Recursion

Tasks/Activities:

Activity 1: Basic Recursion

- **Task 1:** Write a recursive function to calculate the factorial of a number. Log the result for a few test cases.
- **Task 2:** Write a recursive function to calculate the nth Fibonacci number. Log the result for a few test cases.

Activity 2: Recursion with Arrays

- **Task 3:** Write a recursive function to find the sum of all elements in an array. Log the result for a few test cases.
- **Task 4:** Write a recursive function to find the maximum element in an array. Log the result for a few test cases.

Activity 3: String Manipulation with Recursion

- **Task 5:** Write a recursive function to reverse a string. Log the result for a few test cases.
- **Task 6:** Write a recursive function to check if a string is a palindrome. Log the result for a few test cases.

Activity 4: Recursive Search

- **Task 7:** Write a recursive function to perform a binary search on a sorted array. Log the index of the target element for a few test cases.
- **Task 8:** Write a recursive function to count the occurrences of a target element in an array. Log the result for a few test cases.

Activity 5: Tree Traversal (Optional)

- **Task 9:** Write a recursive function to perform an in-order traversal of a binary tree. Log the nodes as they are visited.
- **Task 10:** Write a recursive function to calculate the depth of a binary tree. Log the result for a few test cases.

Feature Request:

1. **Factorial and Fibonacci Script:** Write a script that includes recursive functions to calculate the factorial and Fibonacci numbers.
2. **Array Recursion Script:** Create a script that includes recursive functions to find the sum and maximum element of an array.
3. **String Recursion Script:** Write a script that includes recursive functions to reverse a string and check if a string is a palindrome.
4. **Recursive Search Script:** Create a script that includes recursive functions for binary search and counting occurrences in an array.
5. **Tree Traversal Script:** Write a script that includes recursive functions for in-order traversal and depth calculation of a binary tree (optional).

Achievement:

By the end of these activities, students will:

- Understand and implement basic recursion.
- Apply recursion to solve problems with arrays and strings.
- Use recursion for searching and counting elements in arrays.
- Perform tree traversal and calculate tree depth using recursion (optional).

Day 17: Data Structures

Tasks/Activities:

Activity 1: Linked List

- **Task 1:** Implement a `Node` class to represent an element in a linked list with properties `value` and `next`.
- **Task 2:** Implement a `LinkedList` class with methods to add a node to the end, remove a node from the end, and display all nodes.

Activity 2: Stack

- **Task 3:** Implement a `Stack` class with methods `push` (add element), `pop` (remove element), and `peek` (view the top element).
- **Task 4:** Use the `Stack` class to reverse a string by pushing all characters onto the stack and then popping them off.

Activity 3: Queue

- **Task 5:** Implement a `Queue` class with methods `enqueue` (add element), `dequeue` (remove element), and `front` (view the first element).
- **Task 6:** Use the `Queue` class to simulate a simple printer queue where print jobs are added to the queue and processed in order.

Activity 4: Binary Tree

- **Task 7:** Implement a `TreeNode` class to represent a node in a binary tree with properties `value`, `left`, and `right`.
- **Task 8:** Implement a `BinaryTree` class with methods for inserting values and performing in-order traversal to display nodes.

Activity 5: Graph (Optional)

- **Task 9:** Implement a `Graph` class with methods to add vertices, add edges, and perform a breadth-first search (BFS).
- **Task 10:** Use the `Graph` class to represent a simple network and perform BFS to find the shortest path between two nodes.

Feature Request:

1. **Linked List Script:** Write a script that implements a linked list with methods to add, remove, and display nodes.
2. **Stack Script:** Create a script that implements a stack and uses it to reverse a string.
3. **Queue Script:** Write a script that implements a queue and simulates a printer queue.
4. **Binary Tree Script:** Create a script that implements a binary tree with insertion and in-order traversal methods.
5. **Graph Script:** Write a script that implements a graph and performs breadth-first search (optional).

Achievement:

By the end of these activities, students will:

- Implement and use linked lists for dynamic data storage.
- Use stacks for LIFO (Last-In-First-Out) operations and reverse data.
- Use queues for FIFO (First-In-First-Out) operations and simulate real-world scenarios.
- Implement binary trees for hierarchical data storage and traversal.
- Understand and use graphs for network representations and pathfinding (optional).

Day 18: Algorithms

Tasks/Activities:

Activity 1: Sorting Algorithms

- **Task 1:** Implement the bubble sort algorithm to sort an array of numbers in ascending order. Log the sorted array.
- **Task 2:** Implement the selection sort algorithm to sort an array of numbers in ascending order. Log the sorted array.
- **Task 3:** Implement the quicksort algorithm to sort an array of numbers in ascending order. Log the sorted array.

Activity 2: Searching Algorithms

- **Task 4:** Implement the linear search algorithm to find a target value in an array. Log the index of the target value.
- **Task 5:** Implement the binary search algorithm to find a target value in a sorted array. Log the index of the target value.

Activity 3: String Algorithms

- **Task 6:** Write a function to count the occurrences of each character in a string. Log the character counts.
- **Task 7:** Write a function to find the longest substring without repeating characters in a string. Log the length of the substring.

Activity 4: Array Algorithms

- **Task 8:** Write a function to rotate an array by `k` positions. Log the rotated array.
- **Task 9:** Write a function to merge two sorted arrays into one sorted array. Log the merged array.

Activity 5: Dynamic Programming (Optional)

- **Task 10:** Write a function to solve the Fibonacci sequence using dynamic programming. Log the Fibonacci numbers.
- **Task 11:** Write a function to solve the knapsack problem using dynamic programming. Log the maximum value that can be obtained.

Feature Request:

1. **Sorting Algorithm Script:** Write a script that implements bubble sort, selection sort, and quicksort algorithms to sort arrays.
2. **Searching Algorithm Script:** Create a script that implements linear search and binary search algorithms to find values in arrays.
3. **String Algorithm Script:** Write a script that counts character occurrences and finds the longest substring without repeating characters.
4. **Array Algorithm Script:** Create a script that rotates arrays and merges sorted arrays.
5. **Dynamic Programming Script:** Write a script that solves the Fibonacci sequence and knapsack problem using dynamic programming (optional).

Achievement:

By the end of these activities, students will:

- Implement and understand common sorting algorithms.
- Implement and understand common searching algorithms.
- Solve string manipulation problems using algorithms.
- Perform array operations using algorithms.
- Apply dynamic programming to solve complex problems (optional).

Day 19: Regular Expressions

Tasks/Activities:

Activity 1: Basic Regular Expressions

- **Task 1:** Write a regular expression to match a simple pattern (e.g., match all occurrences of the word "JavaScript" in a string). Log the matches.
- **Task 2:** Write a regular expression to match all digits in a string. Log the matches.

Activity 2: Character Classes and Quantifiers

- **Task 3:** Write a regular expression to match all words in a string that start with a capital letter. Log the matches.
- **Task 4:** Write a regular expression to match all sequences of one or more digits in a string. Log the matches.

Activity 3: Grouping and Capturing

- **Task 5:** Write a regular expression to capture the area code, central office code, and line number from a US phone number format (e.g., (123) 456-7890). Log the captures.
- **Task 6:** Write a regular expression to capture the username and domain from an email address. Log the captures.

Activity 4: Assertions and Boundaries

- **Task 7:** Write a regular expression to match a word only if it is at the beginning of a string. Log the matches.
- **Task 8:** Write a regular expression to match a word only if it is at the end of a string. Log the matches.

Activity 5: Practical Applications

- **Task 9:** Write a regular expression to validate a simple password (must include at least one uppercase letter, one lowercase letter, one digit, and one special character). Log whether the password is valid.
- **Task 10:** Write a regular expression to validate a URL. Log whether the URL is valid.

Feature Request:

1. **Basic Regex Script:** Write a script that uses regular expressions to match simple patterns and log the matches.
2. **Character Classes and Quantifiers Script:** Create a script that uses regular expressions to match words with specific characteristics and log the matches.
3. **Grouping and Capturing Script:** Write a script that uses regular expressions to capture parts of a string, such as phone numbers and email addresses, and log the captures.
4. **Assertions and Boundaries Script:** Create a script that uses regular expressions to match words at specific positions in a string and log the matches.
5. **Validation Script:** Write a script that uses regular expressions to validate passwords and URLs and log whether they are valid.

Achievement:

By the end of these activities, students will:

- Understand and create basic regular expressions.
- Use character classes and quantifiers in regular expressions.
- Implement grouping and capturing in regular expressions.
- Apply assertions and boundaries in regular expressions.
- Use regular expressions for practical applications like validating passwords and URLs.

Day 20: LocalStorage and SessionStorage

Tasks/Activities:

Activity 1: Understanding LocalStorage

- **Task 1:** Write a script to save a string value to `localStorage` and retrieve it. Log the retrieved value.
- **Task 2:** Write a script to save an object to `localStorage` by converting it to a JSON string. Retrieve and parse the object, then log it.

Activity 2: Using LocalStorage

- **Task 3:** Create a simple form that saves user input (e.g., name and email) to `localStorage` when submitted. Retrieve and display the saved data on page load.
- **Task 4:** Write a script to remove an item from `localStorage`. Log the `localStorage` content before and after removal.

Activity 3: Understanding SessionStorage

- **Task 5:** Write a script to save a string value to `sessionStorage` and retrieve it. Log the retrieved value.
- **Task 6:** Write a script to save an object to `sessionStorage` by converting it to a JSON string. Retrieve and parse the object, then log it.

Activity 4: Using SessionStorage

- **Task 7:** Create a simple form that saves user input (e.g., name and email) to `sessionStorage` when submitted. Retrieve and display the saved data on page load.
- **Task 8:** Write a script to remove an item from `sessionStorage`. Log the `sessionStorage` content before and after removal.

Activity 5: Comparing LocalStorage and SessionStorage

- **Task 9:** Write a function that accepts a key and a value, and saves the value to both `localStorage` and `sessionStorage`. Retrieve and log the values from both storage mechanisms.
- **Task 10:** Write a function that clears all data from both `localStorage` and `sessionStorage`. Verify that both storages are empty.

Feature Request:

1. **LocalStorage Script:** Write a script that saves, retrieves, and removes items from `localStorage`, and displays the saved data on page load.
2. **SessionStorage Script:** Create a script that saves, retrieves, and removes items from `sessionStorage`, and displays the saved data on page load.
3. **Storage Comparison Script:** Write a script that saves data to both `localStorage` and `sessionStorage`, retrieves the data, and compares the results.
4. **Clear Storage Script:** Create a script that clears all data from both `localStorage` and `sessionStorage`, and verifies the operation.

Achievement:

By the end of these activities, students will:

- Understand how to use `localStorage` and `sessionStorage` for persistent and session-specific data storage.
- Save, retrieve, and remove data from both `localStorage` and `sessionStorage`.
- Implement form data storage using `localStorage` and `sessionStorage`.
- Compare and contrast the use cases for `localStorage` and `sessionStorage`.

Day 21: LeetCode Easy

Tasks/Activities:

Activity 1: Two Sum

- **Task 1**: Solve the "Two Sum" problem on LeetCode.
  - Write a function that takes an array of numbers and a target number, and returns the indices of the two numbers that add up to the target.
  - Log the indices for a few test cases.

Activity 2: Reverse Integer

- **Task 2**: Solve the "Reverse Integer" problem on LeetCode.
  - Write a function that takes an integer and returns it with its digits reversed.
  - Handle edge cases like negative numbers and numbers ending in zero.
  - Log the reversed integers for a few test cases.

Activity 3: Palindrome Number

- **Task 3**: Solve the "Palindrome Number" problem on LeetCode.
  - Write a function that takes an integer and returns true if it is a palindrome, and false otherwise.
  - Log the result for a few test cases, including edge cases like negative numbers.

Activity 4: Merge Two Sorted Lists

- **Task 4**: Solve the "Merge Two Sorted Lists" problem on LeetCode.
  - Write a function that takes two sorted linked lists and returns a new sorted list by merging them.
  - Create a few test cases with linked lists and log the merged list.

Activity 5: Valid Parentheses

- **Task 5**: Solve the "Valid Parentheses" problem on LeetCode.
  - Write a function that takes a string containing just the characters '(', ')', '{', '}', '[' and ']', and determines if the input string is valid.
  - A string is valid if open brackets are closed in the correct order.
  - Log the result for a few test cases.

Feature Request:

1. **Two Sum Script**: Write a script that includes a function to solve the "Two Sum" problem and logs the indices of the two numbers.
2. **Reverse Integer Script**: Create a script that includes a function to reverse an integer and handles edge cases.
3. **Palindrome Number Script**: Write a script that includes a function to check if an integer is a palindrome and logs the result.
4. **Merge Two Sorted Lists Script**: Create a script that includes a function to merge two sorted linked lists and logs the merged list.
5. **Valid Parentheses Script**: Write a script that includes a function to check if a string of parentheses is valid and logs the result.

Achievement:

By the end of these activities, students will:

- Solve common LeetCode problems.
- Apply problem-solving skills to implement algorithms.
- Understand and handle edge cases in algorithmic solutions.
- Gain confidence in solving easy-level coding challenges on LeetCode.

Day 22: LeetCode Medium

Tasks/Activities:

Activity 1: Add Two Numbers

- **Task 1:** Solve the "Add Two Numbers" problem on LeetCode.
  - Write a function that takes two non-empty linked lists representing two non-negative integers. The digits are stored in reverse order, and each node contains a single digit. Add the two numbers and return the sum as a linked list.
  - Create a few test cases with linked lists and log the sum as a linked list.

Activity 2: Longest Substring Without Repeating Characters

- **Task 2:** Solve the "Longest Substring Without Repeating Characters" problem on LeetCode.
  - Write a function that takes a string and returns the length of the longest substring without repeating characters.
  - Log the length for a few test cases, including edge cases.

Activity 3: Container With Most Water

- **Task 3:** Solve the "Container With Most Water" problem on LeetCode.
  - Write a function that takes an array of non-negative integers where each integer represents the height of a line drawn at each point. Find two lines that together with the x-axis form a container, such that the container holds the most water.
  - Log the maximum amount of water for a few test cases.

Activity 4: 3Sum

- **Task 4:** Solve the "3Sum" problem on LeetCode.
  - Write a function that takes an array of integers and finds all unique triplets in the array which give the sum of zero.
  - Log the triplets for a few test cases, including edge cases.

Activity 5: Group Anagrams

- **Task 5:** Solve the "Group Anagrams" problem on LeetCode.
  - Write a function that takes an array of strings and groups anagrams together.
  - Log the grouped anagrams for a few test cases.

Feature Request:

1. **Add Two Numbers Script**: Write a script that includes a function to solve the "Add Two Numbers" problem and logs the sum as a linked list.
2. **Longest Substring Script**: Create a script that includes a function to find the longest substring without repeating characters and logs the length.
3. **Container With Most Water Script**: Write a script that includes a function to find the container with the most water and logs the maximum amount of water.
4. **3Sum Script**: Create a script that includes a function to find all unique triplets in an array that sum to zero and logs the triplets.
5. **Group Anagrams Script**: Write a script that includes a function to group anagrams and logs the grouped anagrams.

Achievement:

By the end of these activities, students will:

- Solve common medium-level LeetCode problems.
- Apply advanced problem-solving skills to implement algorithms.
- Understand and handle edge cases in more complex algorithmic solutions.
- Gain confidence in solving medium-level coding challenges on LeetCode.

Day 23: LeetCode Hard

Tasks/Activities:

Activity 1: Median of Two Sorted Arrays

- **Task 1**: Solve the "Median of Two Sorted Arrays" problem on LeetCode.
  - Write a function that takes two sorted arrays of integers and returns the median of the two sorted arrays.
  - Log the median for a few test cases, including edge cases.

Activity 2: Merge k Sorted Lists

- **Task 2**: Solve the "Merge k Sorted Lists" problem on LeetCode.
  - Write a function that takes an array of k linked lists, each linked list is sorted in ascending order, and merges all the linked lists into one sorted linked list.
  - Create a few test cases with linked lists and log the merged list.

Activity 3: Trapping Rain Water

- **Task 3**: Solve the "Trapping Rain Water" problem on LeetCode.
  - Write a function that takes an array of non-negative integers representing an elevation map where the width of each bar is 1, and computes how much water it can trap after raining.
  - Log the amount of trapped water for a few test cases.

Activity 4: N-Queens

- **Task 4**: Solve the "N-Queens" problem on LeetCode.
  - Write a function that places n queens on an n×n chessboard such that no two queens attack each other, and returns all distinct solutions to the n-queens puzzle.
  - Log the distinct solutions for a few test cases.

Activity 5: Word Ladder

- **Task 5**: Solve the "Word Ladder" problem on LeetCode.
  - Write a function that takes a begin word, an end word, and a dictionary of words, and finds the length of the shortest transformation sequence from the begin word to the end word, such that only one letter can be changed at a time and each transformed word must exist in the word list.
  - Log the length of the shortest transformation sequence for a few test cases.

Feature Request:

1. **Median of Two Sorted Arrays Script**: Write a script that includes a function to find the median of two sorted arrays and logs the median.
2. **Merge k Sorted Lists Script**: Create a script that includes a function to merge k sorted linked lists and logs the merged list.
3. **Trapping Rain Water Script**: Write a script that includes a function to calculate the amount of trapped rainwater and logs the result.
4. **N-Queens Script**: Create a script that includes a function to solve the N-Queens problem and logs the distinct solutions.
5. **Word Ladder Script**: Write a script that includes a function to find the shortest transformation sequence in a word ladder and logs the sequence length.

Achievement:

By the end of these activities, students will:

- Solve complex LeetCode problems.
- Apply advanced problem-solving skills to implement efficient algorithms.
- Understand and handle edge cases in hard algorithmic solutions.
- Gain confidence in solving hard-level coding challenges on LeetCode.

Day 24: Project 1 - Weather App

Tasks/Activities:

Activity 1: Setting Up the Project

- **Task 1:** Initialize a new project directory and set up the basic HTML structure for the weather app.
- **Task 2:** Add a basic CSS file to style the weather app, including a container for displaying weather information.

Activity 2: Fetching Weather Data

- **Task 3:** Use the `fetch` API to get current weather data from a public weather API (e.g., OpenWeatherMap). Log the response data to the console.
- **Task 4:** Parse the weather data and display the current temperature, weather condition, and city name on the web page.

Activity 3: Adding Search Functionality

- **Task 5:** Add an input field and a search button to the HTML structure. Style the input and button using CSS.
- **Task 6:** Write a function to fetch and display weather data for a city entered in the search input field. Log any errors to the console.

Activity 4: Displaying a 5-Day Forecast

- **Task 7:** Use the `fetch` API to get a 5-day weather forecast from the public weather API. Log the response data to the console.
- **Task 8:** Parse the forecast data and display the temperature and weather condition for each day in the forecast on the web page.

Activity 5: Enhancing the UI

- **Task 9:** Add icons or images to represent different weather conditions (e.g., sunny, rainy, cloudy) based on the weather data.
- **Task 10:** Add CSS animations or transitions to make the weather app more interactive and visually appealing.

Feature Request:

1. **Weather Data Fetching Script**: Write a script that fetches current weather data from a public API and displays the temperature, weather condition, and city name on the web page.
2. **Search Functionality Script**: Create a script that allows users to search for weather information by city name and displays the results.
3. **5-Day Forecast Script**: Write a script that fetches and displays a 5-day weather forecast on the web page.
4. **UI Enhancement Script**: Create a script that adds icons for different weather conditions and includes CSS animations or transitions for a better user experience.

Achievement:

By the end of these activities, students will:

- Set up a basic project structure with HTML and CSS.
- Use the `fetch` API to retrieve and display weather data from a public API.
- Implement search functionality to fetch weather data for different cities.
- Display a 5-day weather forecast using data from a public API.
- Enhance the user interface with icons and animations to make the weather app more interactive and visually appealing.

Day 25: Project 2 - Movie Search App

Tasks/Activities:

Activity 1: Setting Up the Project

- **Task 1:** Initialize a new project directory and set up the basic HTML structure for the movie search app.
- **Task 2:** Add a basic CSS file to style the movie search app, including a container for displaying movie search results.

Activity 2: Fetching Movie Data

- **Task 3:** Use the `fetch` API to get movie data from a public movie API (e.g., OMDB API or The Movie Database API). Log the response data to the console.
- **Task 4:** Parse the movie data and display the movie title, poster, and release year on the web page.

Activity 3: Adding Search Functionality

- **Task 5:** Add an input field and a search button to the HTML structure. Style the input and button using CSS.
- **Task 6:** Write a function to fetch and display movie data based on a search query entered in the input field. Log any errors to the console.

Activity 4: Displaying Detailed Movie Information

- **Task 7:** Modify the search results to include a "More Info" button for each movie. When clicked, fetch and display additional details about the movie, such as the plot, director, and actors.
- **Task 8:** Create a modal or a new section on the page to display the detailed movie information.

Activity 5: Enhancing the UI

- **Task 9:** Add CSS styles to improve the layout and design of the search results and detailed movie information.
- **Task 10:** Add CSS animations or transitions to make the movie search app more interactive and visually appealing.

Feature Request:

1. **Movie Data Fetching Script**: Write a script that fetches movie data from a public API and displays the title, poster, and release year on the web page.
2. **Search Functionality Script**: Create a script that allows users to search for movies by title and displays the search results.
3. **Detailed Information Script**: Write a script that fetches and displays additional details about a selected movie, such as the plot, director, and actors.
4. **UI Enhancement Script**: Create a script that improves the layout and design of the movie search app with CSS styles and animations.

Achievement:

By the end of these activities, students will:

- Set up a basic project structure with HTML and CSS.
- Use the `fetch` API to retrieve and display movie data from a public API.
- Implement search functionality to fetch and display movie data based on user input.
- Fetch and display detailed information about selected movies.
- Enhance the user interface with CSS styles and animations to make the movie search app more interactive and visually appealing.

Day 26: Project 3 - Chat Application

Tasks/Activities:

Activity 1: Setting Up the Project

- **Task 1:** Initialize a new project directory and set up the basic HTML structure for the chat application.
- **Task 2:** Add a basic CSS file to style the chat application, including a chat window and input area.

Activity 2: Setting Up WebSocket Server

- **Task 3:** Set up a simple WebSocket server using Node.js and the `ws` library. Write a script to create the server and handle connections.
- **Task 4:** Test the WebSocket server by logging messages when clients connect and disconnect.

Activity 3: Establishing WebSocket Connection

- **Task 5:** Add a script to the HTML file to establish a WebSocket connection to the server.
- **Task 6:** Write functions to handle sending and receiving messages through the WebSocket connection. Log received messages to the console.

Activity 4: Building the Chat Interface

- **Task 7:** Modify the HTML structure to include a chat window and an input area for typing messages. Style the chat window and input area using CSS.
- **Task 8:** Write a function to display received messages in the chat window. Add event listeners to send messages when the user presses Enter or clicks a send button.

Activity 5: Enhancing the Chat Application

- **Task 9:** Add user authentication to the chat application. Allow users to enter a username before joining the chat. Display usernames alongside their messages in the chat window.
- **Task 10:** Add CSS styles to differentiate messages sent by different users. Add CSS animations or transitions to make the chat application more interactive and visually appealing.

Feature Request:

1. **WebSocket Server Script:** Write a script to set up a WebSocket server using Node.js and the `ws` library, handling connections and logging messages.
2. **WebSocket Connection Script:** Create a script to establish a WebSocket connection from the client side and handle sending and receiving messages.
3. **Chat Interface Script:** Write a script to build the chat interface, displaying received messages and sending messages from the input area.
4. **User Authentication Script:** Create a script to add user authentication, allowing users to enter usernames and display usernames alongside their messages.
5. **UI Enhancement Script:** Write a script to enhance the chat application's UI with CSS styles and animations, differentiating messages from different users.

Achievement:

By the end of these activities, students will:

- Set up a basic project structure with HTML and CSS.
- Create a WebSocket server using Node.js and the `ws` library.
- Establish a WebSocket connection from the client side to send and receive messages.
- Build a chat interface to display and send messages.
- Add user authentication and display usernames in the chat.
- Enhance the user interface with CSS styles and animations to make the chat application more interactive and visually appealing.

Day 27: Project 4 - Task Management App

Tasks/Activities:

Activity 1: Setting Up the Project

- **Task 1:** Initialize a new project directory and set up the basic HTML structure for the task management app.
- **Task 2:** Add a basic CSS file to style the task management app, including a container for displaying tasks and a form for adding new tasks.

Activity 2: Creating Tasks

- **Task 3:** Add a form to the HTML structure with fields for entering task details (e.g., title, description, due date). Style the form using CSS.
- **Task 4:** Write a script to handle form submission, creating a new task object and adding it to an array of tasks. Display the new task in the task list.

Activity 3: Reading and Displaying Tasks

- **Task 5:** Write a function to iterate over the array of tasks and display each task in the task list. Include task details like title, description, and due date.
- **Task 6:** Style the task list using CSS to make it visually appealing.

Activity 4: Updating Tasks

- **Task 7:** Add an "Edit" button to each task item in the task list. Write a function to populate the form with the task details when the "Edit" button is clicked.
- **Task 8:** Write a function to update the task object in the array and refresh the task list display after editing a task.

Activity 5: Deleting Tasks

- **Task 9:** Add a "Delete" button to each task item in the task list. Write a function to remove the task from the array and refresh the task list display when the "Delete" button is clicked.
- **Task 10:** Add a confirmation dialog before deleting a task to prevent accidental deletions.

Feature Request:

1. **Task Creation Script:** Write a script to handle form submission, creating new tasks and displaying them in the task list.
2. **Task Display Script:** Create a script to display tasks from the array in the task list, including task details and styling.
3. **Task Update Script:** Write a script to handle task editing, updating the task in the array and refreshing the display.
4. **Task Deletion Script:** Create a script to handle task deletion, removing the task from the array and refreshing the display with a confirmation dialog.

Achievement:

By the end of these activities, students will:

- Set up a basic project structure with HTML and CSS.
- Implement task creation, reading, updating, and deletion functionalities.
- Handle form submission to create new tasks and display them in the task list.
- Update existing tasks and refresh the display with edited task details.
- Delete tasks from the list with a confirmation dialog to prevent accidental deletions.
- Style the task management app to make it visually appealing and user-friendly.

Day 28: Project 5 - E-commerce Website

Tasks/Activities: (No need to use database, you can use json files or just an array to simulate database)

Activity 1: Setting Up the Project

- **Task 1:** Initialize a new project directory and set up the basic HTML structure for the e-commerce website.
- **Task 2:** Add a basic CSS file to style the e-commerce website, including a product listing grid and a shopping cart section.

Activity 2: Product Listing

- **Task 3:** Create a JSON file or an array of product objects with details like name, price, description, and image URL.
- **Task 4:** Write a script to dynamically generate the product listing from the product data and display it on the web page. Style the product cards using CSS.

Activity 3: Shopping Cart

- **Task 5:** Add an "Add to Cart" button to each product card. Write a function to handle adding products to the shopping cart.
- **Task 6:** Create a shopping cart section that displays the products added to the cart, including the name, price, and quantity. Update the cart display whenever a product is added.

Activity 4: Cart Management

- **Task 7:** Add functionality to update the quantity of products in the cart. Write a function to handle increasing and decreasing the quantity of items.
- **Task 8:** Add a "Remove" button to each item in the cart. Write a function to handle removing products from the cart and updating the display.

Activity 5: Checkout Process

- **Task 9:** Create a checkout form that collects user information (e.g., name, address, payment details). Style the form using CSS.
- **Task 10:** Write a function to handle form submission, simulating the checkout process. Display a confirmation message with the order details.

Feature Request:

1. **Product Listing Script**: Write a script to generate and display a product listing from an array of product objects or a JSON file.
2. **Shopping Cart Script**: Create a script to handle adding products to the shopping cart and updating the cart display.
3. **Cart Management Script**: Write a script to handle updating the quantity of products in the cart and removing products from the cart.
4. **Checkout Process Script**: Create a script to handle the checkout process, including collecting user information and displaying a confirmation message.

Achievement:

By the end of these activities, students will:

- Set up a basic project structure with HTML and CSS.
- Dynamically generate and display a product listing from product data.
- Implement a shopping cart that allows users to add products, update quantities, and remove items.
- Create a checkout form to collect user information and simulate the checkout process.
- Enhance the user interface with CSS styles to make the e-commerce website visually appealing and user-friendly.

Day 29: Project 6 - Social Media Dashboard

Tasks/Activities:

Activity 1: Setting Up the Project

- Task 1: Initialize a new project directory and set up the basic HTML structure for the social media dashboard.
- Task 2: Add a basic CSS file to style the social media dashboard, including a container for posts and a form for creating new posts.

Activity 2: User Authentication

- Task 3: Create a simple login form that collects a username and password. Style the form using CSS.
- Task 4: Write a script to handle user login and store the logged-in user's information in localStorage or sessionStorage.

Activity 3: Creating Posts

- Task 5: Add a form to the HTML structure with fields for entering post details (e.g., text, image). Style the form using CSS.
- Task 6: Write a script to handle form submission, creating a new post object and adding it to an array of posts. Display the new post in the feed.

Activity 4: Displaying Posts

- Task 7: Write a function to iterate over the array of posts and display each post in the feed. Include post details like text, image, username, and timestamp.
- Task 8: Style the post feed using CSS to make it visually appealing.

Activity 5: Post Interactions

- Task 9: Add "Like" and "Comment" buttons to each post. Write functions to handle liking a post and adding comments to a post.
- Task 10: Display the number of likes and comments for each post. Update the display when users interact with the posts.

Activity 6: Enhancing the UI

- Task 11: Add CSS styles to differentiate posts by different users. Display the logged-in user's posts with a distinct style.
- Task 12: Add CSS animations or transitions to make the social media dashboard more interactive and visually appealing.

Feature Request:

1. **User Authentication Script**: Write a script to handle user login and store the logged-in user's information.
2. **Post Creation Script**: Create a script to handle form submission, creating new posts and displaying them in the feed.
3. **Post Display Script**: Write a script to display posts from an array of posts, including post details and styling.
4. **Post Interaction Script**: Create a script to handle liking and commenting on posts, updating the display with the number of likes and comments.
5. **UI Enhancement Script**: Write a script to enhance the UI with CSS styles and animations, differentiating posts by different users and adding interactivity.

Achievement:

By the end of these activities, students will:

- Set up a basic project structure with HTML and CSS.
- Implement user authentication and store user information.
- Create and display posts with details like text, image, username, and timestamp.
- Handle post interactions like liking and commenting, and update the display accordingly.
- Enhance the user interface with CSS styles and animations to make the social media dashboard visually appealing and user-friendly.

Day 30: Final Project - Social Media Dashboard with Full Features

Tasks/Activities:

Activity 1: Setting Up the Project

- **Task 1:** Initialize a new project directory and set up the basic HTML structure for the final project.
- **Task 2:** Add a basic CSS file to style the social media dashboard, including containers for posts, a sidebar for user information, and a form for creating new posts.

Activity 2: User Authentication

- **Task 3:** Create a login and registration form that collects a username, email, and password. Style the forms using CSS.
- **Task 4:** Write scripts to handle user registration and login, storing user information in localStorage or sessionStorage. Include basic validation for input fields.

Activity 3: User Profiles

- **Task 5:** Create a user profile page that displays the logged-in user's information, including their username, email, and a profile picture. Allow users to update their profile information.
- **Task 6:** Write a script to handle updating the user profile information and saving the changes to localStorage or sessionStorage.

Activity 4: Creating and Displaying Posts

- **Task 7:** Add a form to the HTML structure with fields for entering post details (e.g., text, image). Style the form using CSS.
- **Task 8:** Write a script to handle form submission, creating a new post object and adding it to an array of posts. Display the new post in the feed, including the username of the logged-in user.

Activity 5: Post Interactions

- **Task 9:** Add "Like" and "Comment" buttons to each post. Write functions to handle liking a post and adding comments to a post.
- **Task 10:** Display the number of likes and comments for each post. Update the display when users interact with the posts.

Activity 6: Notifications

- **Task 11:** Implement a simple notification system that alerts users when they receive a new like or comment on their posts. Display notifications in a sidebar.
- **Task 12:** Write a script to handle generating and displaying notifications based on user interactions.

Activity 7: Enhancing the UI

- **Task 13:** Add CSS styles to differentiate posts by different users. Display the logged-in user's posts with a distinct style.
- **Task 14:** Add CSS animations or transitions to make the social media dashboard more interactive and visually appealing.

Feature Request:

1. **User Authentication Script**: Write scripts to handle user registration and login, with basic validation and storing user information.
2. **User Profile Script**: Create a script to display and update user profile information, saving changes to localStorage or sessionStorage.
3. **Post Creation and Display Script**: Write a script to handle form submission for creating posts and displaying them in the feed.
4. **Post Interaction Script**: Create a script to handle liking and commenting on posts, updating the display with the number of likes and comments.
5. **Notification System Script**: Write a script to generate and display notifications for user interactions on posts.
6. **UI Enhancement Script**: Create a script to enhance the UI with CSS styles and animations, differentiating posts by different users and adding interactivity.

Achievement:

By the end of these activities, students will:

- Set up a comprehensive project structure with HTML and CSS.
- Implement user authentication, including registration and login, with input validation.
- Create and update user profiles, displaying user information and allowing updates.
- Handle creating and displaying posts with user-specific details.
- Implement post interactions like liking and commenting, with real-time updates.
- Create a notification system to alert users of new interactions on their posts.
- Enhance the user interface with CSS styles and animations to create a visually appealing and interactive social media dashboard.