MERN Stack Training

Tasks

Task 1:

1: JavaScript Language - An Introduction to JavaScript, Code structure

1. **An Introduction to JavaScript**:

    - Task 1: Write a simple script that displays "Hello, World!" on the web page using an alert box.
    - Task 2: Experiment with different data types in JavaScript (e.g., string, number, boolean) by declaring and logging them in the console.
    - Task 3: Use the console to perform basic math operations like addition, subtraction, multiplication, and division.
    - Task 4: Declare two strings and concatenate them using the + operator.
    - Task 5: Use the typeof operator to check the data type of various variables.

2. **Code structure**:

    - Task 6: Write a multi-line JavaScript comment and a single-line comment. Explain the difference.
    - Task 7: Create a script with both semicolon-separated and not separated lines. Note any differences in behavior.
    - Task 8: Use proper indentation to format a nested loop.
    - Task 9: Declare multiple variables in a single line.
    - Task 10: Place a script tag at the top and bottom of an HTML document. Note any differences in behavior.

2: The modern mode, "use strict", Variables

1. **The modern mode, "use strict"**:

    - Task 11: Write a script without using "use strict" and try to assign a value to an undeclared variable. Note the result.

- Task 12: Enable “use strict” mode and repeat the above action, noting the difference.
- Task 13: In “use strict” mode, try to delete a variable, function, or function parameter.
- Task 14: Assign a value to an undeclared variable without “use strict” and then with “use strict”.
- Task 15: Declare a variable with a reserved keyword in “use strict” mode.

2. **Variables**:

    - Task 16: Declare variables using let, const, and var. Discuss when each should be used.
    - Task 17: Attempt to reassign a const variable and observe the result.
    - Task 18: Declare a variable without initializing it and print its value.
    - Task 19: Assign a number, string, and boolean value to a variable and print its type using typeof.
    - Task 20: Rename a variable and observe the outcome.

3: Data types, Basic operators, maths

1. **Data types**:

    - Task 21: Create variables of different data types (e.g., string, number, boolean, null, undefined, object).
    - Task 22: Use the typeof operator to determine the type of various variables.
    - Task 23: Declare a symbol and print its type.
    - Task 24: Assign the value null to a variable and check its type using typeof.
    - Task 25: Differentiate between declaring a variable using var and let in terms of scope.

2. **Basic operators, maths**:

    - Task 26: Convert a string to a number using both implicit and explicit conversion.
    - Task 27: Convert a boolean to a string and vice versa.
    - Task 28: Practice basic arithmetic operators (+, -, *, /, %).
    - Task 29: Use the ++ and -- operators on a numeric variable.

- Task 30: Explore the precedence of operators by combining multiple operators in a single expression.

4: Comparisons, Conditional branching: if, '?'

1. **Comparisons**:

   - Task 31: Compare two numbers using relational operators (>, <, >=, <=).
   - Task 32: Use equality () and strict equality (=) operators to compare different data types and note the differences.
   - Task 33: Compare two strings lexicographically.
   - Task 34: Use the inequality (!=) and strict inequality (!==) operators to compare values.
   - Task 35: Compare null and undefined using both == and ===.

2. **Conditional branching: if, '?'**:

   - Task 36: Write an if statement that checks if a number is even or odd.
   - Task 37: Use nested if statements to classify a number as negative, positive, or zero.
   - Task 38: Use the conditional (ternary) operator '?' to rewrite a simple if...else statement.
   - Task 39: Check the validity of a variable using the ? operator.
   - Task 40: Use the conditional operator to assign a value to a variable based on a condition.

5: Logical operators, Functions

1. **Logical operators**:

   - Task 41: Evaluate various combinations of logical operators (&&, ||, !).
   - Task 42: Use logical operators to write a condition that checks if a number is in a given range.
   - Task 43: Use the NOT (!) operator to invert a boolean value.
   - Task 44: Evaluate the short-circuiting nature of logical operators.

- Task 45: Compare two non-boolean values using logical operators and observe the result.

2. **Functions**:

   - Task 46: Write a function that takes two numbers as arguments and returns their sum.
   - Task 47: Create a function that calculates the area of a rectangle.
   - Task 48: Declare a function without parameters and call it.
   - Task 49: Write a function that returns nothing and observe the default return value.
   - Task 50: Declare a function with default parameters and call it with different arguments.

3. **Arrow Functions**:

   - Task 51: Declare a simple arrow function named greet that takes one parameter name and returns the string "Hello, name!". Test your function with various names.
   - Task 52: Write an arrow function named add that takes two parameters and returns their sum. Validate your function with several pairs of numbers.
   - Task 53: Declare an arrow function named isEven that checks if a number is even. If the number is even, it should return true; otherwise, false. Remember that if the arrow function body has a single statement, you can omit the curly braces.
   - Task 54: Implement an arrow function named maxValue that takes two numbers as parameters and returns the larger number. Here, you'll need to use curly braces for the function body and the return statement.
   - Task 55: Examine the behavior of the *this* keyword inside an arrow function vs a traditional function. Create an object named myObject with a property value set to 10 and two methods: multiplyTraditional using a traditional function and multiplyArrow using an arrow function. Both methods should attempt to multiply the value property by a number passed as a parameter. Check the value of this inside both methods.

Mini Project: Simple JavaScript Quiz App

Objective: Create a web-based quiz application using HTML, CSS, and JavaScript.

Description: Users can take a quiz on basic JavaScript topics. They answer questions, and at the end, they receive a score with feedback.

Requirements and Tasks Implementation:

1. **Setup:**
    - Create an HTML skeleton with a linked JavaScript file.
    - Use script tags appropriately in the HTML, considering **Code Structure**.
2. **User Interface:**
    - A title indicating it's a JavaScript Quiz.
    - Display one question at a time with multiple-choice answers.
    - Buttons to navigate to the next or previous question.
    - A progress bar or indicator to show which question the user is on.
    - At the end, display the user's score and feedback.
3. **Functionality:**

    a. **Question & Answer Data:**
    - Store questions, options, and correct answer in objects (**Data Types** topic).
    - Group all question objects into an array.

    b. **Displaying Questions:**
    - Function to load a question and options (**Functions topic**).
    - Highlight user's choice upon selecting an option using **Conditional Branching**.

    c. **Navigating Questions:**
    - Functions to go to the next or previous question.
    - Use **Basic Operators** to increment or decrement the current question index.

d. **Scoring:**

- Keep track of user's score. Use **Variables** to store the user's current score.
- When a user selects an answer, compare it to the correct answer using **Comparisons**.
- Update the score based on the user's answer.

e. **Feedback:**

- Use **Conditional Branching** to give feedback based on the user's score.
- E.g., if score is above 80%, display "Excellent!"; between 50% and 80% "Good job!"; below 50% "Keep Practicing".

f. **Use "use strict" Mode:**

- Enable strict mode for your JavaScript to catch common errors.

g. **Arrow Functions & Function Expressions:**

- Implement helper functionalities, like updating the progress bar, highlighting answers, etc., using **Arrow Functions**.
- Use **Function Expressions** to define some of your functions, e.g., a function that resets the quiz for another attempt.

h. **Logical Operators:**

- Implement a feature to filter questions based on difficulty or topic. Use **Logical Operators** to achieve this.

4. **Bonus:**

- Add a timer to the quiz. User has to finish the quiz within a set time.
- Provide explanations for answers using **Function Expressions** that display a modal with more info about the question's topic.
- Add animations or effects when the user selects an answer, moves to the next question, or finishes the quiz.
- Use local storage to save the user's progress and score.

5. **Testing & Debugging:**

    - Use the browser's developer tools console to debug any issues that arise.
    - Validate user input and provide feedback for any unexpected behavior using **Logical Operators** and **Comparisons**.