**Module 6 – Mernstack – Javascript Essential and Advanced**

Question 1: What is JavaScript? Explain the role of JavaScript in web development

JavaScript (often abbreviated as JS) is a **high-level, interpreted programming language** primarily used to make web pages interactive. It is a **client-side scripting language**, meaning it runs directly in the user’s web browser, although it can also be run on the server using environments like Node.js.

JavaScript was initially created to **add dynamic behavior to static HTML pages**, such as updating content, validating forms, and responding to user actions.

**Role of JavaScript in Web Development:**

1. Client-Side Interactivity
2. Dynamic Content Manipulation
3. Asynchronous Communication
4. Web Applications and Frameworks
5. Server-Side Development
6. Client-Side Interactivity
7. Dynamic Content Manipulation
8. Asynchronous Communication
9. Web Applications and Frameworks
10. Server-Side Development

**Use of <script> Tag in HTML**

The <script> tag is used to include **JavaScript code** in an HTML document. JavaScript can be used to add interactivity, manipulate the DOM, handle events, validate forms, and much more.

**Key Points:**

1. **Placement in HTML**:
   * In the <head> section for scripts that should load before the page content.
   * At the end of the <body> section to ensure the HTML content loads first (common practice).
2. **Types of Scripts**:
   * **Inline JavaScript**: The code is written directly inside the <script> tag.
   * **External JavaScript**: The code is stored in a separate .js file and linked using the src attribute.

Lab Assignment:  
Task: • Create a simple HTML page and add a

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>JavaScript Alert Example</title>

<script>

// JavaScript code to display alert when page loads

window.onload = function() {

alert("Welcome to JavaScript!");

};

</script>

</head>

<body>

<h1>My First JavaScript Page</h1>

<p>This page shows an alert when it loads.</p>

</body>

</html>

Question 1: What are variables in JavaScript? How do you declare a variable using var, let, and const?

Variables are **containers used to store data** (like numbers, text, or objects) that can be used and manipulated throughout your program.

**a) var**

* **Scope:** Function-scoped (not block-scoped).
* **Re-declaration:** Allowed.
* **Re-assignment:** Allowed.
* **Old way** of declaring variables (less recommended now).

var city = "Ahmedabad";

city = "Mumbai"; // Re-assigning is allowed

var city = "Delhi"; // Re-declaring is allowed

**let**

* **Scope:** Block-scoped (only available inside { } where declared).
* **Re-declaration:** Not allowed in the same scope.
* **Re-assignment:** Allowed.
* **Recommended** for most variables.

let country = "India";

country = "USA"; // Re-assigning is allowed

// let country = "Canada"; // ❌ Error: can't re-declare in same scope

**c) const**

* **Scope:** Block-scoped.
* **Re-declaration:** Not allowed.
* **Re-assignment:** Not allowed (value is constant).
* Used for **values that shouldn’t change**.

const pi = 3.14;

// pi = 3.1415; // ❌ Error: cannot re-assign a const variable

// const pi = 3.1415; // ❌ Error: cannot re-declare

Question 2: Explain the different data types in JavaScript. Provide examples for each.

Data types define the **kind of value a variable can hold**.  
In JavaScript, data types are broadly divided into:

1. **Primitive (basic) data types**
2. **Non-primitive (reference) data types**

**a) String**

* Represents **text**, enclosed in quotes ("", '', or ``````).

**b) Number**

* Represents **numeric values**, including integers and decimals.

**c) Boolean**

* Represents **true or false** values. Useful in conditions.

**d) Undefined**

* A variable **declared but not assigned a value** is undefined.

**e) Null**

* Represents **no value** or **empty value**. It is assigned **intentionally**.

let result = null;

console.log(result); // Output: null

* Represents a **unique identifier**, often used in objects to avoid name conflicts.

let id = Symbol("id");

let id2 = Symbol("id");

console.log(id === id2); // false, always unique

**3. Non-Primitive / Reference Data Types**

These store **collections of values or objects** and are mutable.

**Object**

* Represents a **collection of key-value pairs**.

let person = {

name: "Prashant",

age: 22

};

console.log(person.name); // Prashant

console.log(person.age); // 22

**Array**

* A type of object that stores an **ordered list of values**.

let fruits = ["Apple", "Banana", "Mango"];

console.log(fruits[0]); // Apple

console.log(fruits[2]); // Mango

**Function**

* A **block of code** that can be called to perform a task.

function greet() {

console.log("Hello World");

}

greet(); // Output: Hello World

Question 3: What is the difference between undefinedand nullin JavaScript?

**Undefined**

* **Definition:** A variable is undefined when it has been **declared but not assigned any value**.
* **Type:** undefined
* **Represents:** Absence of value **by default**.
* **Example:**

let x;

console.log(x); // Output: undefined

console.log(typeof x); // Output: "undefined"

**Null**

* **Definition:** null is an **intentional assignment** that represents “no value” or “empty”.
* **Type:** object (this is a quirk in JavaScript)
* **Represents:** Absence of a value **intentionally set by the programmer**.
* **Example:**

let result = null;

console.log(result); // Output: null

console.log(typeof result); // Output: "object"

Lab Assignment:

Write a JavaScript program to declare variables for different data types (string,number, boolean, null, and undefined). • Log the values of the variables and their types to the console using console.log().

// Declaring variables of different data types

// String

let name = "Prashant";

// Number

let age = 22;

// Boolean

let isStudent = true;

// Null

let result = null;

// Undefined

let score; // declared but not assigned

// Logging values and types to the console

console.log("Value of name:", name, "| Type:", typeof name);

console.log("Value of age:", age, "| Type:", typeof age);

console.log("Value of isStudent:", isStudent, "| Type:", typeof isStudent);

console.log("Value of result:", result, "| Type:", typeof result);

console.log("Value of score:", score, "| Type:", typeof score);

Question 1: What are the different types of operators in JavaScript? Explain with examples. • Arithmetic operators • Assignment operators • Comparison operators • Logical operators

**. Arithmetic Operators**

Used to perform **mathematical operations** like addition, subtraction, multiplication, etc.

| **Operator** | **Description** | **Example** | **Result** |
| --- | --- | --- | --- |
| + | Addition | 5 + 3 | 8 |
| - | Subtraction | 5 - 3 | 2 |
| \* | Multiplication | 5 \* 3 | 15 |
| / | Division | 15 / 3 | 5 |
| % | Modulus (remainder) | 5 % 3 | 2 |
| \*\* | Exponentiation | 2 \*\* 3 | 8 |

**Example in JavaScript:**

let a = 10;

let b = 3;

console.log(a + b); // 13

console.log(a - b); // 7

console.log(a \* b); // 30

console.log(a / b); // 3.3333

console.log(a % b); // 1

console.log(a \*\* b); // 1000

**2. Assignment Operators**

Used to **assign values** to variables.

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| = | Assign | x = 5 |
| += | Add and assign | x += 3 → x = x + 3 |
| -= | Subtract and assign | x -= 2 |
| \*= | Multiply and assign | x \*= 4 |
| /= | Divide and assign | x /= 2 |
| %= | Modulus and assign | x %= 3 |

**Example:**

let x = 10;

x += 5; // x = x + 5

console.log(x); // 15

x \*= 2; // x = x \* 2

console.log(x); // 30

**3. Comparison Operators**

Used to **compare two values**. They return a **boolean (true/false)**.

| **Operator** | **Description** | **Example** | **Result** |
| --- | --- | --- | --- |
| == | Equal to (value only) | 5 == '5' | true |
| === | Equal value and type | 5 === '5' | false |
| != | Not equal to (value only) | 5 != 3 | true |
| !== | Not equal value or type | 5 !== '5' | true |
| > | Greater than | 5 > 3 | true |
| < | Less than | 5 < 3 | false |
| >= | Greater than or equal to | 5 >= 5 | true |
| <= | Less than or equal to | 5 <= 4 | false |

**Example:**

let a = 10;

let b = 20;

console.log(a == b); // false

console.log(a < b); // true

console.log(a != b); // true

console.log(a === 10); // true

**4. Logical Operators**

Used to **combine or invert boolean values**.

| **Operator** | **Description** | **Example** | **Result** |
| --- | --- | --- | --- |
| && | Logical AND | true && false | false |
| ` |  | ` | Logical OR |
| ! | Logical NOT | !true | false |

**Example:**

let x = true;

let y = false;

console.log(x && y); // false

console.log(x || y); // true

console.log(!x); // false

Question 2: What is the difference between ==and ===in JavaScript?

**== (Equality Operator)**

* Checks **if the values are equal**, **ignoring the data type**.
* Performs **type coercion**, meaning it converts one value to match the other’s type before comparison.

**Example:**

let a = 5;

let b = "5";

console.log(a == b); // true, because "5" is converted to number 5

**=== (Strict Equality Operator)**

* Checks **if the values AND data types are equal**.
* **No type coercion** is done. Both **value and type** must match.

**Example:**

let a = 5;

let b = "5";

console.log(a === b); // false, number !== string

Question 1: What is control flow in JavaScript? Explain how if-elsestatements work withan example.

**What is Control Flow in JavaScript?**

* **Control flow** determines **the order in which the code is executed**.
* By default, JavaScript executes code **line by line** (top to bottom).
* **Control flow statements** let you make decisions or repeat code based on conditions, loops, or other rules.

**Common control flow statements:**

* if, if-else, if-else if-else
* switch
* Loops: for, while, do-while

**2. if-else Statements**

if-else is used to **execute code conditionally**.

**Syntax:**

if (condition) {

// code runs if condition is true

} else {

// code runs if condition is false

}

**Example:**

let age = 18;

if (age >= 18) {

console.log("You are eligible to vote.");

} else {

console.log("You are not eligible to vote yet.");

}

**Output:**

You are eligible to vote.

**Explanation:**

1. The if block checks the condition age >= 18.
2. If the condition is **true**, the first block runs.
3. If the condition is **false**, the else block runs instead.

**3. if-else if-else Example**

Used when there are **multiple conditions**:

let marks = 75;

if (marks >= 90) {

console.log("Grade: A+");

} else if (marks >= 75) {

console.log("Grade: A");

} else if (marks >= 50) {

console.log("Grade: B");

} else {

console.log("Grade: C");

}

**Output:**

Grade: A

Question 2: Describe how switchstatements work in JavaScript. When should you use a switchstatement instead of if-else?

A **switch statement** is a **control flow statement** in JavaScript that allows you to **choose among multiple alternatives** based on the value of an expression.

* Instead of writing **many if-else-if conditions**, a switch provides a **cleaner and more readable structure** when a single variable or expression needs to be compared against several possible values.
* It is particularly useful when all comparisons are **equality checks** (i.e., checking if a value equals some constant).

**2. Structure of a Switch Statement**

A switch statement has the following parts:

1. **Expression**: The value or variable you want to evaluate.
2. **Case labels**: Specific values that the expression can match. Each case is followed by code to execute if it matches.
3. **Break statement**: Stops execution from continuing into the next case. Without it, execution will **“fall through”** to subsequent cases.
4. **Default case**: Optional. Executed if none of the case values match the expression.

**Generic Structure:**

switch (expression) {

case value1:

// Code executed if expression === value1

break;

case value2:

// Code executed if expression === value2

break;

...

default:

// Code executed if no case matches

}

**3. How Switch Statements Work**

1. The **expression is evaluated once**.
2. JavaScript **compares the expression to each case value** using **strict equality (===)**.
3. If a match is found, **the code inside that case executes**.
4. If there is no break, execution continues into the next case (fall-through).
5. If no match is found, the **default block executes** (if provided).

**Key Points:**

* Switch is only for **discrete value comparisons**, not ranges or complex conditions.
* break is essential to **prevent unwanted execution of multiple cases**.
* default is optional but ensures a response when no case matches.

**4. When to Use Switch Statements**

**Use switch statements when:**

1. You need to **compare a single variable/expression against multiple discrete values**.
2. You want to **replace long chains of if-else-if** statements for clarity and readability.
3. You want to **group cases** that should perform the same action.

**Do NOT use switch when:**

* You need to check **ranges** (like if(x > 10)), because switch only checks equality.
* Conditions are complex or involve multiple variables.

Lab Assignment:

ask 1: • Write a JavaScript program to check if a number is positive, negative, or zero usingan if-elsestatement.

// Program to check if a number is positive, negative, or zero

let number = 5; // You can change this value to test

if (number > 0) {

console.log(number + " is positive");

} else if (number < 0) {

console.log(number + " is negative");

} else {

console.log(number + " is zero");

}

Task 2: • Create a JavaScript program using a switchstatement to display the day of the week based on the user input (e.g., 1 for Monday, 2 for Tuesday, etc.).

// Program to display day of the week using switch

// Ask the user to enter a number between 1 and 7

let dayNumber = prompt("Enter a number (1-7) to get the day of the week:");

// Convert the input to a number

dayNumber = Number(dayNumber);

let dayName;

switch(dayNumber) {

case 1:

dayName = "Monday";

break;

case 2:

dayName = "Tuesday";

break;

case 3:

dayName = "Wednesday";

break;

case 4:

dayName = "Thursday";

break;

case 5:

dayName = "Friday";

break;

case 6:

dayName = "Saturday";

break;

case 7:

dayName = "Sunday";

break;

default:

dayName = "Invalid input! Please enter a number between 1 and 7.";

}

console.log(dayName);

alert(dayName); // Optional: show the result in an alert box

Question 1: Explain the different types of loops in JavaScript (for, while, do-while). Provide abasic example of each.

Loops are used to **repeat a block of code multiple times** until a specified condition is met.  
They help **avoid writing repetitive code**.

**2. Types of Loops**

**a) for Loop**

* Repeats a block of code a **specific number of times**.
* Syntax:

for (initialization; condition; increment/decrement) {

// code to execute

}

**Example:**

for (let i = 1; i <= 5; i++) {

console.log("for loop iteration:", i);

}

**Output:**

for loop iteration: 1

for loop iteration: 2

for loop iteration: 3

for loop iteration: 4

for loop iteration: 5

**b) while Loop**

* Repeats a block of code **while a condition is true**.
* Syntax:

while (condition) {

// code to execute

}

**Example:**

let i = 1;

while (i <= 5) {

console.log("while loop iteration:", i);

i++; // increment is necessary to avoid infinite loop

}

**Output:**  
Same as for loop example.

**c) do-while Loop**

* Similar to while loop, **but the code executes at least once** even if the condition is false.
* Syntax:

do {

// code to execute

} while (condition);

**Example:**

let i = 1;

do {

console.log("do-while loop iteration:", i);

i++;

} while (i <= 5);

**Output:**  
Same as for and while loops.

**Key Difference:**  
If the initial condition is false:

let i = 10;

do {

console.log("This will run once even if condition is false");

} while (i < 5);

**Output:**

This will run once even if condition is false

• Question 2: What is the difference between a whileloop and a do-whileloop?

**ey Differences**

| **Feature** | **while Loop** | **do-while Loop** |
| --- | --- | --- |
| Condition Checked | Before executing loop body | After executing loop body |
| Execution Guarantee | May not run if false | Runs at least once |
| Use Case | Unknown iterations, run only if condition is true | Run at least once, then repeat if condition is true |

Lab Assignment:

Task 1: • Write a JavaScript program using a forloop to print numbers from 1 to 10.

// Program to print numbers from 1 to 10 using a for loop

for (let i = 1; i <= 10; i++) {

console.log(i);

}

Task 2: • Create a JavaScript program that uses a whileloop to sum all even numbers between 1 and 20.

// Program to sum all even numbers between 1 and 20

let i = 1; // Start from 1

let sum = 0; // Variable to store the sum

while (i <= 20) {

if (i % 2 === 0) { // Check if the number is even

sum += i; // Add even number to sum

}

i++; // Move to the next number

}

console.log("Sum of all even numbers between 1 and 20 is:", sum);

Task 3: • Write a do-whileloop that continues to ask the user for input until they enter a number greater than 10. // Program to keep asking user for a number until it is greater than 10

let number;

do {

number = prompt("Enter a number greater than 10:"); // Ask user input

number = Number(number); // Convert input to number

} while (number <= 10); // Continue loop if number is <= 10

console.log("You entered:", number);

alert("You entered: " + number);

• Question 1: What are functions in JavaScript? Explain the syntax for declaring and

**What Are Functions in JavaScript?**

A **function** is a **reusable block of code** that performs a **specific task**.

* You can **call** the function whenever needed instead of writing the same code again.
* Functions can **take input**, perform operations, and **return output**.

Think of a function like a **machine**:

* You **give it input** → it **does something** → it **gives output**.

**2. Benefits of Using Functions**

1. **Code Reusability** – Write once, use multiple times.
2. **Modularity** – Break your program into smaller, manageable parts.
3. **Maintainability** – Easier to debug and update code.
4. **Readability** – Makes code organized and cleaner.

Question 2: What is the difference between a function declaration and a functionexpression?

**. Main Differences**

| **Feature** | **Function Declaration** | **Function Expression** |
| --- | --- | --- |
| Hoisting | Yes, can be called before declaration | No, must be defined before calling |
| Syntax | function name(params) {} | const name = function(params) {} |
| Name | Must have a name | Can be anonymous |
| Use Case | General reusable functions | Callbacks, event handlers, assigned to variables |

• Question 3: Discuss the concept of parameters and return values in functions

**1. Parameters in Functions**

* **Parameters** are **placeholders** for values that a function can receive when it is called.
* They allow functions to work with **different inputs** without changing the function’s code.
* You define parameters **inside the parentheses** of the function.

**Syntax Example:**

function greet(name) { // 'name' is a parameter

console.log("Hello, " + name + "!");

}

greet("Alice"); // Passes "Alice" as an argument → Output: Hello, Alice!

greet("Bob"); // Passes "Bob" as an argument → Output: Hello, Bob!

**Notes:**

* The values you pass when calling the function are called **arguments**.
* Functions can have **multiple parameters**:

function add(a, b) { // a and b are parameters

console.log(a + b);

}

add(5, 3); // Output: 8

**2. Return Values in Functions**

* A **return value** is the **output** a function sends back to the code that called it.
* Use the return keyword to return a value.
* Once a return is executed, the function **stops executing**.

Lab Assignment:

• Task 1: • Write a function greetUserthat accepts a user’s name as a parameter and displaysa greeting message (e.g., "Hello, John!").

// Function to greet the user

function greetUser(name) {

console.log("Hello, " + name + "!");

alert("Hello, " + name + "!"); // Optional: display in an alert box

}

// Call the function with a name

greetUser("John"); // Output: Hello, John!

greetUser("Alice"); // Output: Hello, Alice!

Task 2: • Create a JavaScript function calculateSumthat takes two numbers as parameters, adds them, and returns the result.

// Function to calculate the sum of two numbers

function calculateSum(num1, num2) {

return num1 + num2; // Return the sum

}

// Call the function and store the result

let result = calculateSum(5, 8);

console.log("The sum is:", result); // Output: The sum is: 13

Question 1: What is an array in JavaScript? How do you declare and initialize an array?

**What is an Array?**

* An **array** is a **special type of object** in JavaScript used to **store multiple values in a single variable**.
* Arrays can hold **values of any type**, including numbers, strings, booleans, objects, or even other arrays.
* Each value in an array is called an **element**, and each element has an **index** starting from 0.

**Example Conceptually:**

let fruits = ["Apple", "Banana", "Mango"];

// "Apple" → index 0, "Banana" → index 1, "Mango" → index 2

**2. How to Declare an Array**

You can declare an array in **two main ways**:

**a) Using Square Brackets (Recommended)**

let numbers = [1, 2, 3, 4, 5];

let fruits = ["Apple", "Banana", "Mango"];

**b) Using the Array Constructor**

let numbers = new Array(1, 2, 3, 4, 5);

let fruits = new Array("Apple", "Banana", "Mango");

**Notes:**

* Using **square brackets []** is simpler and more commonly used.
* Arrays can contain **mixed data types**:

let mixedArray = [1, "Hello", true, null];

**3. Accessing Array Elements**

* Use **index** to access elements:

let fruits = ["Apple", "Banana", "Mango"];

console.log(fruits[0]); // Output: Apple

console.log(fruits[2]); // Output: Mango

• Question 2: Explain the methods push(), pop(), shift(), and unshift()used in arrays.

**. push()**

* **Purpose:** Adds **one or more elements to the end** of an array.
* **Returns:** The **new length** of the array.

**Syntax:**

array.push(element1, element2, ...);

**Example:**

let fruits = ["Apple", "Banana"];

fruits.push("Mango"); // Add "Mango" at the end

console.log(fruits); // Output: ["Apple", "Banana", "Mango"]

**2. pop()**

* **Purpose:** Removes the **last element** from an array.
* **Returns:** The **removed element**.

**Syntax:**

array.pop();

**Example:**

let fruits = ["Apple", "Banana", "Mango"];

let removed = fruits.pop();

console.log(removed); // Output: "Mango"

console.log(fruits); // Output: ["Apple", "Banana"]

**3. shift()**

* **Purpose:** Removes the **first element** from an array.
* **Returns:** The **removed element**.

**Syntax:**

array.shift();

**Example:**

let fruits = ["Apple", "Banana", "Mango"];

let removed = fruits.shift();

console.log(removed); // Output: "Apple"

console.log(fruits); // Output: ["Banana", "Mango"]

**4. unshift()**

* **Purpose:** Adds **one or more elements to the beginning** of an array.
* **Returns:** The **new length** of the array.

**Syntax:**

array.unshift(element1, element2, ...);

**Example:**

let fruits = ["Banana", "Mango"];

fruits.unshift("Apple"); // Add "Apple" at the beginning

console.log(fruits); // Output: ["Apple", "Banana", "Mango"]

Lab Assignment:

Task 1: • Declare an array of fruits (["apple", "banana", "cherry"]). Use JavaScript to: • Add a fruit to the end of the array. • Remove the first fruit from the array. • Log the modified array to the console.

// Declare an array of fruits

let fruits = ["apple", "banana", "cherry"];

// Add a fruit to the end of the array

fruits.push("mango"); // Add "mango" at the end

// Remove the first fruit from the array

fruits.shift(); // Removes "apple"

// Log the modified array

console.log(fruits); // Output: ["banana", "cherry", "mango"]

Task 2: • Write a program to find the sum of all elements in an array of numbers.

// Array of numbers

let numbers = [2, 5, 8, 10, 3];

// Variable to store the sum

let sum = 0;

// Loop through the array and add each element to sum

for (let i = 0; i < numbers.length; i++) {

sum += numbers[i];

}

// Display the result

console.log("The sum of all elements is:", sum);

Question 1: What is an object in JavaScript? How are objects different from arrays?

**What is an Object in JavaScript?**

* An **object** is a **collection of key-value pairs** (also called properties).
* Each **key (property name)** is a string, and the **value** can be any data type: number, string, boolean, array, function, or even another object.
* Objects are used to **store structured data** and represent **real-world entities**.

**Syntax Example:**

let person = {

name: "John",

age: 25,

isStudent: true

};

console.log(person.name); // Output: John

console.log(person.age); // Output: 25

**Explanation:**

* name, age, isStudent → keys (properties)
* "John", 25, true → values
* Access values using **dot notation** (person.name) or **bracket notation** (person["name"]).

Question 2: Explain how to access and update object properties using dot notation andbracket notation

**Dot Notation**

* **Access a property:** Use object.propertyName.
* **Update a property:** Assign a new value using =.

**Example:**

let person = {

name: "John",

age: 25,

city: "New York"

};

// Access properties

console.log(person.name); // Output: John

console.log(person.age); // Output: 25

// Update a property

person.age = 26;

console.log(person.age); // Output: 26

**Key Points:**

* Simple and readable.
* Works only with **valid identifiers** (no spaces or special characters in property names).

**2. Bracket Notation**

* **Access a property:** Use object["propertyName"].
* **Update a property:** Assign a new value using =.
* **Useful for:**
  + Property names with **spaces or special characters**
  + Property names stored in a **variable**

**Example:**

let person = {

"full name": "John Doe",

age: 25,

city: "New York"

};

// Access properties

console.log(person["full name"]); // Output: John Doe

console.log(person["city"]); // Output: New York

// Update a property

person["city"] = "Los Angeles";

console.log(person.city); // Output: Los Angeles

// Using a variable as key

let key = "age";

console.log(person[key]); // Output: 25

Lab Assignment:

Task: • Create a JavaScript object carwith properties brand, model, and year. Use JavaScript to: • Access and print the car’s brand and model. • Update the yearproperty. • Add a new property colorto the car object.

// Create a car object

let car = {

brand: "Toyota",

model: "Corolla",

year: 2020

};

// Access and print the car's brand and model

console.log("Brand:", car.brand); // Output: Brand: Toyota

console.log("Model:", car.model); // Output: Model: Corolla

// Update the year property

car.year = 2023;

console.log("Updated Year:", car.year); // Output: Updated Year: 2023

// Add a new property color

car.color = "Red";

console.log("Car object after adding color:", car);

// Output: { brand: "Toyota", model: "Corolla", year: 2023, color: "Red" }

Explanation:

Create object:

javascript

Copy code

let car = { brand: "Toyota", model: "Corolla", year: 2020 };

Initializes a car object with three properties.

Access properties:

javascript

Copy code

car.brand, car.model

Uses dot notation to get values.

Update property:

javascript

Copy code

car.year = 2023;

Changes the value of year.

Add new property:

javascript

Copy code

car.color = "Red";

Adds a new property color dynamically.

Question 1: What are JavaScript events? Explain the role of event listeners

**What Are JavaScript Events?**

* **Events** are **actions or occurrences** that happen in the browser, which JavaScript can respond to.
* Examples of events include:
  + **User interactions:** click, double-click, mouseover, mouseout, keypress, input, submit
  + **Browser events:** load, resize, scroll
  + **Other events:** change, focus, blur

**Example:**  
When a user clicks a button, the click event occurs.

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

<script>

// Event handler using onclick

document.getElementById("myButton").onclick = function() {

alert("Button was clicked!");

};

</script>

• Question 2: How does the addEventListener()method work in JavaScript? Provide an example.

**. What is addEventListener()?**

* addEventListener() is a **method used to attach an event handler** to an HTML element.
* It **listens** for a specified event (like click, mouseover, keydown) and executes a function when the event occurs.
* Preferred over inline event attributes (like onclick) because it allows **multiple event listeners** on the same element and keeps HTML clean.

**2. Syntax**

element.addEventListener(event, function, useCapture);

| **Parameter** | **Description** |
| --- | --- |
| event | The type of event to listen for (e.g., "click", "mouseover"). |
| function | The function to execute when the event occurs. |
| useCapture | Optional boolean. true for capturing phase, false for bubbling phase (default is false). |

Lab Assignment

• Task: • Create a simple webpage with a button that, when clicked, displays an alert saying"Button clicked!" using JavaScript event listeners.

<!DOCTYPE html>

<html>

<head>

<title>Button Click Example</title>

<style>

/\* Optional: make button look nicer \*/

button {

padding: 10px 20px;

font-size: 16px;

cursor: pointer;

}

</style>

</head>

<body>

<h2>Click the Button</h2>

<button id="myButton">Click Me!</button>

<script>

// Select the button element

const button = document.getElementById("myButton");

// Add a click event listener

button.addEventListener("click", function() {

alert("Button clicked!");

});

</script>

</body>

</html>

Question 1: What is the DOM (Document Object Model) in JavaScript? How does JavaScriptinteract with the DOM?

**What is the DOM?**

* **DOM** stands for **Document Object Model**.
* It is a **programming interface** for HTML and XML documents.
* The DOM **represents the page as a tree of objects** (nodes), where each HTML element is an object that can be **accessed, modified, added, or deleted** using JavaScript.

**Key Points:**

* The DOM **turns a webpage into a structured document** that JavaScript can manipulate.
* Every HTML element, attribute, and text is a **node** in the DOM tree.

**Example Conceptually:**

<p id="para">Hello World</p>

* <p> → element node
* id="para" → attribute node
* "Hello World" → text node

**2. How JavaScript Interacts with the DOM**

JavaScript can **read and modify** the DOM dynamically, which allows you to:

1. **Access elements**
   * By ID, class, tag, or CSS selector.
2. let para = document.getElementById("para");
3. **Change content**
4. para.textContent = "Hello JavaScript!";
5. **Modify styles**
6. para.style.color = "blue";
7. para.style.fontSize = "20px";
8. **Add or remove elements**
9. let newPara = document.createElement("p");
10. newPara.textContent = "New paragraph added!";
11. document.body.appendChild(newPara);
12. **Handle events**
13. para.addEventListener("click", function() {
14. alert("Paragraph clicked!");

});

Question 2: Explain the methods getElementById(), getElementsByClassName(),and querySelector()used to select elements from the DOM.

**getElementById()**

* **Purpose:** Selects a single HTML element **by its unique ID**.
* **Returns:** The **element object** if found, otherwise null.
* **Syntax:**

let element = document.getElementById("elementId");

**Example:**

<p id="para1">Hello World</p>

<script>

let para = document.getElementById("para1");

para.textContent = "Hello JavaScript!";

</script>

* Only **one element** can have a given ID.
* Best for selecting **unique elements**.

**2. getElementsByClassName()**

* **Purpose:** Selects **all elements** that have a specific class name.
* **Returns:** An **HTMLCollection** (similar to an array but not exactly an array).
* **Syntax:**

let elements = document.getElementsByClassName("myClass");

**Example:**

<p class="text">Paragraph 1</p>

<p class="text">Paragraph 2</p>

<script>

let paragraphs = document.getElementsByClassName("text");

console.log(paragraphs.length); // Output: 2

paragraphs[0].style.color = "blue"; // Change first paragraph's color

</script>

* Use **indexing** to access individual elements (elements[0]).

**3. querySelector()**

* **Purpose:** Selects the **first element** that matches a **CSS selector**.
* **Returns:** A single element object.
* **Syntax:**

let element = document.querySelector("selector");

**Example:**

<p class="text">Hello World</p>

<script>

let para = document.querySelector(".text"); // Selects first element with class "text"

para.style.fontWeight = "bold";

</script

Lab Assignment:

Task: • Create an HTML page with a paragraph (

) that displays "Hello, World!". • Use JavaScript to: • Change the text inside the paragraph to "JavaScript is fun!". • Change the color of the paragraph to blue.

<!DOCTYPE html>

<html>

<head>

<title>DOM Manipulation Example</title>

<style>

/\* Optional: initial style \*/

p {

font-size: 18px;

}

</style>

</head>

<body>

<p id="myParagraph">Hello, World!</p>

<script>

// Select the paragraph element by its ID

const para = document.getElementById("myParagraph");

// Change the text content

para.textContent = "JavaScript is fun!";

// Change the text color

para.style.color = "blue";

</script>

</body>

</html>

Question 1: Explain the setTimeout()and setInterval()functions in JavaScript. Howare they used for timing events?

**setTimeout()**

* **Purpose:** Executes a function **once** after a specified delay (in milliseconds).
* **Syntax:**

setTimeout(function, delay, param1, param2, ...);

| **Parameter** | **Description** |
| --- | --- |
| function | The function to execute after the delay |
| delay | Time in milliseconds to wait before executing the function |
| param1, ... | Optional parameters to pass to the function |

**Example:**

function greet() {

alert("Hello after 3 seconds!");

}

// Call greet after 3000 milliseconds (3 seconds)

setTimeout(greet, 3000);

* The function greet() will run **only once** after 3 seconds.

**Use Case:**

* Delaying actions, like showing a popup after a few seconds.

**2. setInterval()**

* **Purpose:** Executes a function **repeatedly at specified intervals** (in milliseconds).
* **Syntax:**

setInterval(function, interval, param1, param2, ...);

| **Parameter** | **Description** |
| --- | --- |
| function | The function to execute repeatedly |
| interval | Time in milliseconds between each execution |
| param1, ... | Optional parameters to pass to the function |

Question 2: Provide an example of how to use setTimeout()to delay an action by 2 seconds.

<!DOCTYPE html>

<html>

<head>

<title>setTimeout Example</title>

</head>

<body>

<button id="delayButton">Click Me</button>

<script>

const button = document.getElementById("delayButton");

button.addEventListener("click", function() {

// Delay the action by 2000 milliseconds (2 seconds)

setTimeout(function() {

alert("This message appears after 2 seconds!");

}, 2000);

});

</script>

</body>

</html>

Lab Assignment

• Task 1: • Write a program that changes the background color of a webpage after 5 secondsusing setTimeout(). • Task 2: • Create a digital clock that updates every second using setInterval().

**Task 1: Change Background Color After 5 Seconds**

<!DOCTYPE html>

<html>

<head>

<title>setTimeout Example</title>

</head>

<body>

<h2>Background will change in 5 seconds</h2>

<script>

// Delay 5000 milliseconds (5 seconds)

setTimeout(function() {

document.body.style.backgroundColor = "lightblue";

}, 5000);

</script>

</body>

</html>

**Task 2: Digital Clock Using setInterval**

<!DOCTYPE html>

<html>

<head>

<title>Digital Clock</title>

<style>

#clock {

font-size: 30px;

font-weight: bold;

font-family: monospace;

}

</style>

</head>

<body>

<h2>Digital Clock</h2>

<div id="clock">00:00:00</div>

<script>

function updateClock() {

const now = new Date();

let hours = now.getHours().toString().padStart(2, '0');

let minutes = now.getMinutes().toString().padStart(2, '0');

let seconds = now.getSeconds().toString().padStart(2, '0');

document.getElementById("clock").textContent = `${hours}:${minutes}:${seconds}`;

}

// Call updateClock every 1000 milliseconds (1 second)

setInterval(updateClock, 1000);

// Initialize clock immediately

updateClock();

</script>

</body>

</html>