ASSIGNMENT

Module 6 – Mernstack – Javascript Essential and Advanced

• JavaScript Introduction

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

JavaScript is a high-level, interpreted programming language mainly used to create interactive and dynamic content on websites. It is one of the "core technologies" of the web, alongside HTML and CSS.

Role of JavaScript in Web Development

- Interactivity
    - Enables user dialogs, animations, form validations, and visual effects.
    - Examples: Pop-ups, sliders, drag-and-drop interfaces.
- Dynamic Content
    - Updates web pages in real time without reloading (e.g., data refresh, interactive calendars, live search).
    - Uses DOM manipulation to change HTML/CSS based on user actions.
- Client-Side Scripting
    - Runs in the browser, reducing server load and providing instant feedback.
    - Controls what users see and experience directly.
- Integration with HTML/CSS
    - HTML structures the webpage, CSS styles it, and JavaScript adds behavior/functionality.
    - Can alter HTML elements and apply new styles interactively.
- Frameworks and Libraries
    - Modern web apps rely on frameworks like React, Angular, or Vue—built using JavaScript.
    - Libraries (like jQuery or Lodash) simplify common tasks.
- Asynchronous Operations

- Handles API calls, remote data fetching, timers, event handling (using AJAX/Fetch API).
- Allows building Single-Page Applications (SPAs) for smooth user experiences.

• Question 2: How is JavaScript different from other programming languages like Python or Java?
Ans.
Here's a clear comparison of JavaScript vs Python vs Java:

Aspect	JavaScript	Python	Java
Main Usage	Web development (browser, frontend), server apps	General-purpose, data science, ML	Enterprise, mobile, web backend
Execution	Interpreted (JIT in browsers)	Interpreted	Compiled (JVM; bytecode)
Syntax	C-like, dynamic, simple for web	Simple and readable	Verbose, statically typed
Typing	Dynamically typed	Dynamically typed	Statically typed
Platform	Runs in browsers (client-side), Node.js (server)	Runs anywhere with interpreter	Runs on Java Virtual Machine (JVM)

Concurrency	Event-driven, single-threaded (non-blocking)	Multi-threading, async supported	True multi-threading
OOP Support	Prototype-based objects	Class-based objects	Class-based objects
Libraries	Large ecosystem for web	Huge, especially in ML/data	Very large, especially enterprise
Use Cases	Interactive UIs, SPAs, web APIs	Scripts, ML, web, automation	Banking, Android, big web apps

• Question 3: Discuss the use of <script> tag in HTML. How can you link an external JavaScript file to an HTML document?
Ans.

Use of `<script>` Tag in HTML

The `<script>` tag in HTML is used to embed JavaScript code or link to external JavaScript files.
JavaScript helps in adding interactivity, dynamic behavior, and functionality to web pages.

Where can `<script>` be placed?

- Inside the `<head>` section
- At the end of the `<body>` section (recommended for better performance)

Example (JavaScript inside HTML):

```
<script>
    document.write("Hello from JavaScript!");
```

```
</script>
```

Linking an External JavaScript File

To keep HTML clean and modular, JavaScript code can be written in a separate `.js` file and linked using the `src` attribute of the `<script>` tag.

Syntax:

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

Example:

```
<!DOCTYPE html>
<html>
<head>
    <title>External JS Example</title>
    <script src="main.js"></script> <!-- Linking JS file -->
</head>
<body>
    <h1>Welcome!</h1>
</body>
</html>
```

main.js file:

```
alert("JavaScript File Loaded Successfully!");
```

Best Practice

Place the `<script>` tag before closing `</body>` so the HTML can load first:

```
<script src="main.js"></script>
</body>
</html>
```

• Variables and Data Types

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

Variables in JavaScript are containers used to store data values.
Example: a number, text string, boolean value, object, etc.

Variables allow us to reuse and manipulate data in a program.

Declaring Variables in JavaScript

JavaScript provides three keywords to declare variables:

Keyword	Change Value	Scope	Hoisting	Usage
`var`	Yes	Function-scoped	Yes (initialized as undefined)	Old method, used before ES6
`let`	Yes	Block-scoped	Yes (not initialized)	Used for variables that change
`const`	No	Block-scoped	Yes (not initialized)	Used for constant values

Syntax Examples

1 Using `var`

```
var name = "John";
name = "David";  // Reassign allowed
```

2 Using `let`

```
let age = 25;
age = 30;  // Reassign allowed
```

3 Using `const`

```
const PI = 3.14;
PI = 3.15; // Error: reassignment not allowed
```

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

1 Primitive Data Types

Primitive values are stored directly in memory and are immutable.

Data Type	Description	Example
String	Textual data enclosed in quotes	`"Hello"`, `'A'`
Number	Numeric values, including integers & decimals	`25`, `3.14`
Boolean	True or false values	`true`, `false`
Undefined	A variable declared but not assigned	`let x; console.log(x)` → `undefined`
Null	Represents no value or empty value	`let y = null;`
BigInt	Large integer values beyond Number limit	`1234567890123456789 0n`
Symbol	Unique and immutable values, mostly used as object keys	`let sym = Symbol("id");`

Example Code for Primitive Types

```
let name = "John";        // String
let age = 21;             // Number
let isStudent = true;     // Boolean
let city;                 // Undefined
let data = null;          // Null
let bigNumber = 9007199254740991n; // BigInt
```

```
let id = Symbol('id');   // Symbol
```

2. Non-Primitive (Reference) Data Types

Stored as references (memory locations), mutable in nature.

Data Type	Description	Example
Object	Collection of properties in key-value form	`{name: "John", age: 21}`
Array	Ordered list of values	`[1, 2, 3, 4]`
Function	Block of code that executes when called	`function greet(){}`

Example Code for Non-Primitive Types:

```
let person = {
  name: "Amit",
  age: 22
}; // Object

let colors = ["red", "green", "blue"]; // Array

function add(a, b) {
  return a + b;
} // Function
```

• Question 3: What is the difference between undefined and null in JavaScript?
Ans.:

undefined

- Automatically assigned to a variable that is declared but not given a value.
- Means "value is not assigned"
- Type of `undefined` is "undefined"

Example:

```
let a;
console.log(a); // undefined
console.log(typeof a); // "undefined"
```

Null

- Manually assigned by the programmer.
- Means "empty" or "no value"
- Type of `null` is "object" (this is actually a JavaScript bug but kept for compatibility)

Example:

```
let b = null;
console.log(b); // null
console.log(typeof b); // "object"
```

• JavaScript Operators

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

1.Arithmetic Operators

Used to perform mathematical operations.

Operator	Description	Example

+	Addition	10 + 5 → 15
-	Subtraction	10 - 5 → 5
*	Multiplication	10 * 5 → 50
/	Division	10 / 2 → 5
%	Modulus (remainder)	10 % 3 → 1
**	Exponentiation	2 ** 3 → 8
++	Increment	a++
--	Decrement	a--

Example:

```
let a = 10;
let b = 3;
console.log(a + b);  // 13
console.log(a % b);  // 1
```

2.Assignment Operators

Used to assign values to variables.

Operator	Meaning	Example	Result
=	Assign value	x = 10	x becomes 10
+=	Add & assign	x += 5	x = x + 5
-=	Subtract & assign	x -= 2	x = x - 2
*=	Multiply & assign	x *= 3	x = x * 3
/=	Divide & assign	x /= 2	x = x / 2

%=	Modulus & assign	x %= 4	x = x % 4

Example:

```
let x = 10;
x += 5; // 15
console.log(x);
```

3. Comparison Operators

Used to compare two values. Always return true or false.

Operator	Description	Example
==	Equal to (checks only value)	5 == "5" → true
===	Strict equal (checks value + type)	5 === "5" → false
!=	Not equal	5 != 4 → true
!==	Strict not equal	5 !== "5" → true
>	Greater than	10 > 5 → true
<	Less than	3 < 7 → true
>=	Greater or equal	5 >= 5 → true
<=	Less or equal	3 <= 2 → false

Example:

```
console.log(10 > 5);    // true
console.log(5 === "5"); // false
```

4. Logical Operators

Used to combine multiple conditions.

Operator	Meaning	Example	Output
&&	Logical AND	true && false	false

`||`	Logical OR	`true || false`	true
`!`	Logical NOT	`!true`	false

Example:

```
let a = 10;
let b = 20;

console.log(a > 5 && b > 10); // true
console.log(a < 5 || b > 10); // true
console.log(!(a > b));        // true
```

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

== vs === in JavaScript

Operator	Name	Checks Value	Checks Data Type	Example Result
==	Loose Equality	Yes	No	"5" == 5 → true
===	Strict Equality	Yes	Yes	"5" === 5 → false

Example Code

```
console.log(5 == "5");  // true (value same, type ignored)
console.log(5 === "5"); // false (type different: number vs string)

console.log(true == 1);   // true
console.log(true === 1);  // false
```

• Control Flow (If-Else, Switch)

• Question 1: What is control flow in JavaScript? Explain how if-else statements work with an example.

Ans.:

Control Flow refers to the order in which statements are executed in a JavaScript program. Normally, code runs from top to bottom, but using control structures like:

`if-else`
`switch`
`loops`
functions

We can change the normal flow based on conditions or logic.

if-else Statement

`if-else` is used to make decisions in a program.

How it works:

- `if` block executes only when the condition is true
- `else` block executes when the condition is false
- You can also use `else if` for multiple conditions

Example:

```
let age = 18;

if (age >= 18) {
  console.log("You are eligible to vote.");
} else {
  console.log("You are not eligible to vote.");
}
```

Output:

```
You are eligible to vote.
```

Another Example with else-if:

```
let marks = 75;
```

```
if (marks >= 90) {
  console.log("Grade A");
} else if (marks >= 75) {
  console.log("Grade B");
} else {
  console.log("Grade C");
}
```

• Question 2: Describe how switch statements work in JavaScript. When should you use a switch statement instead of if-else?
Ans.:

Switch Statement in JavaScript

A switch statement is used to perform different actions based on different values of a variable or expression.
It is useful when you want to compare the same variable with multiple possible values.

Syntax:

```
switch(expression) {
  case value1:
    // code to run if expression == value1
    break;

  case value2:
    // code to run if expression == value2
    break;

  default:
    // code to run if no case matches
}
```

Key Points:

Keyword	Purpose
`switch`	Checks a variable/expression
`case`	A match condition to compare with the expression

`break`	Stops execution and exits the switch
`default`	Runs only if no case matches

Example:

```
let day = 3;
let dayName;

switch(day) {
  case 1:
    dayName = "Monday";
    break;
  case 2:
    dayName = "Tuesday";
    break;
  case 3:
    dayName = "Wednesday";
    break;
  default:
    dayName = "Invalid Day";
}

console.log(dayName); // Output: Wednesday
```

When Should You Use `switch` Instead of `if-else`

Use switch when:

You are checking the same variable against many values
There are multiple fixed cases (e.g., menu options, weekdays, button actions)
Code readability and structure matters

Use if-else when:

Conditions involve ranges or complex comparisons
You check multiple variables or logical expressions

• Loops (For, While, Do-While)

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

Ans.:

1. for Loop

Used when the number of iterations (times to run) is known.

Syntax:

```
for (initialization; condition; increment/decrement) {
    // code to run
}
```

Example:

```
for (let i = 1; i <= 5; i++) {
    console.log("Number: " + i);
}
```

This will print numbers 1 to 5.

2. while Loop

Used when the number of iterations is not known beforehand.
It runs as long as the condition is true.

Syntax:

```
while (condition) {
    // code to run
}
```

Example:

```
let i = 1;
while (i <= 5) {
    console.log("Count: " + i);
    i++;
}
```

This prints 1 to 5 until condition becomes false.

3. do-while Loop

Similar to while, but executes the block at least once even if the condition is false, because condition is checked after running the code.

Syntax:

```
do {
    // code to run
} while (condition);
```

Example:

```
let i = 1;
do {
    console.log("Value: " + i);
    i++;
} while (i <= 5);
```

Executes the code block first, then checks the condition.

Summary Table :

Loop Type	Condition Checked	Executes At Least Once?	Best Used When
for	Before each iteration	No	Counted loops (# of times known)
while	Before each iteration	No	Unknown iterations, condition-based
do-while	After each iteration	Yes	Must run first, then repeat if true

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

Ans.:

Key Differences

Feature	while Loop	do-while Loop

Condition check	Before executing the loop block	After executing the loop block
Minimum execution	May not run even once	Runs at least once
Syntax style	Condition-first	Code-first

Example to Show the Difference

while Loop:

```
let num = 5;

while (num < 5) {
    console.log("Inside while loop");
}
```

Condition is false from the start → output: Nothing

do-while Loop:

```
let num = 5;

do {
    console.log("Inside do-while loop");
} while (num < 5);
```

Runs the statement first even though the condition is false → output:

```
Inside do-while loop
```

• Functions

• Question 1: What are functions in JavaScript? Explain the syntax for declaring and calling a function.
Ans.:

Functions:

- Reuse code multiple times

- Makes code modular and easy to understand
- Helps in debugging and organization

Syntax: Declaring a Function

```
function functionName(parameters) {
    // code to execute
}
```

Example:

```
function greet(name) {
    console.log("Hello, " + name);
}
```

Syntax: Calling (Invoking) a Function

```
functionName(arguments);
```

Example:

```
greet("John");  // Output: Hello, John
```

How it works:

1. Declaration creates the function (but doesn't run it).
2. Calling executes the function with given values.

Another Example Without Parameters

```
function sayHello() {
    console.log("Welcome to JavaScript!");
}

sayHello();  // Calling the function
```

Summary Table

Term	Meaning
Function declaration	Defines the function
Function call	Runs the function
Parameters	Variables inside function
Arguments	Actual values passed

• Question 2: What is the difference between a function declaration and a function expression?
Ans.:

Function Declaration

A function is defined with the `function` keyword — on its own.

Example:

```
function greet() {
    console.log("Hello!");
}
greet(); // Calling the function
```

Key Feature:

Hoisted → Can be called before its definition in the code.

```
greet(); // Works!
function greet() {
    console.log("Hello!");
}
```

Function Expression

A function is assigned to a variable.

Example:

```
const greet = function() {
    console.log("Hello!");
};
greet();
```

Key Feature:

Not hoisted → Cannot be called before the line where it is defined.

```
greet(); //  Error: greet is not defined yet
const greet = function() {
    console.log("Hello!");
};
```

Summary Table

Feature	Function Declaration	Function Expression
Hoisting	Yes (can be used before declared)	No (must be defined before use)
Syntax	Standalone `function`	Assigned to a variable
Use case	Common for regular functions	Common for callbacks & anonymous functions

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

Parameters

Parameters are variables in the function definition that act as placeholders for values.

Example:

```
function add(a, b) {  // a and b are parameters
    console.log(a + b);
}
add(5, 3); // → 5 and 3 are arguments
```

➡ Parameters = inside function
➡ Arguments = actual values passed while calling

Return Values

A function can return a result using the `return` keyword.
This allows the function to send back a value.

Example:

```
function multiply(x, y) {
    return x * y; // returns the result
}

let result = multiply(4, 6);
console.log(result); // Output: 24
```

➡ `return` stops the function execution and gives a value back

Key Points

Feature	Description
Parameters	Input placeholders in function definition
Arguments	Actual input values given during function call
Return value	Output the function sends back
return keyword	Ends the function and returns a value

• Arrays

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

An array is a collection of elements stored in ordered, indexed form.

Array index starts from 0
(1st element → index 0, 2nd → index 1, and so on)

Example:

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

Declaring an Array

You can declare an array in two common ways:

1. Using square brackets (recommended)

```
let arr = [];
```

2. Using Array constructor

```
let arr = new Array();
```

Initializing an Array (with values)

```
let numbers = [10, 20, 30, 40];
let mixed = ["Hello", 25, true, null];
```

Using Array constructor:

```
let colors = new Array("Red", "Green", "Blue");
```

Accessing Array Elements

```
let cars = ["BMW", "Audi", "Tesla"];
console.log(cars[0]); // Output: BMW
```

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

1. push( )

Adds one or more elements to the end of an array.

Example:

```
let fruits = ["Apple", "Banana"];
fruits.push("Mango");
console.log(fruits); // ["Apple", "Banana", "Mango"]
```

2. pop( )

Removes the last element from an array and returns it.

Example:

```
let fruits = ["Apple", "Banana", "Mango"];
let removed = fruits.pop();
```

```
console.log(fruits); // ["Apple", "Banana"]
console.log(removed); // "Mango"
```

3. shift()

Removes the first element from an array and returns it.

Example:

```
let colors = ["Red", "Green", "Blue"];
let first = colors.shift();
console.log(colors); // ["Green", "Blue"]
console.log(first);  // "Red"
```

4. unshift()

Adds one or more elements to the beginning of an array.

Example:

```
let colors = ["Green", "Blue"];
colors.unshift("Red");
console.log(colors); // ["Red", "Green", "Blue"]
```

• Objects

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

Object:

Example:

```
let person = {
    name: "John",
    age: 25,
    isStudent: true
};
```

- `name`, `age`, `isStudent` → keys (properties)
- `"John"`, `25`, `true` → values

Objects are used to represent real-world entities in a structured way.

How to Access Object Values?

```
console.log(person.name);      // Dot notation
console.log(person["age"]);    // Bracket notation
```

• Question 2: Explain how to access and update object properties using dot notation and bracket notation.
Ans.:

1. Dot Notation

Most commonly used.
Works when property names are simple (no spaces, no special characters).

Accessing:

```
let student = {
    name: "Amit",
    age: 20
};

console.log(student.name); // Output: Amit
```

Updating:

```
student.age = 21;
console.log(student.age); // Output: 21
```

2. Bracket Notation

Used when:

- Property name has spaces

- Property name is stored in a variable
- Using special characters or numbers

Accessing:

```
console.log(student["name"]); // Output: Amit
```

Updating:

```
student["age"] = 22;
console.log(student["age"]); // Output: 22
```

Special Use Case with Variables

```
let key = "name";
console.log(student[key]); // Output: Amit
```

(You cannot use `student.key` here — it would look for a property literally named `"key"`)

• JavaScript Events

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

What are JavaScript Events?

JavaScript events are actions or occurrences that happen in a web page, usually triggered by the user or the browser.
These events allow the webpage to respond to user interactions.

Common Examples of Events:

Event	Trigger
`click`	User clicks a button or element

`mouseover`	Mouse pointer moves over an element
`keydown`	A key is pressed on the keyboard
`submit`	Form is submitted
`load`	Web page finishes loading

Events make web pages interactive and dynamic.

Role of Event Listeners

An event listener is a function in JavaScript that waits for an event to occur on a specific element.
When the event happens, the event listener executes a function (event handler) in response.

Syntax:

```
element.addEventListener("eventName", functionName);
```

Example:

HTML

```
<button id="btn">Click Me</button>
```

JavaScript

```
document.getElementById("btn").addEventListener("click", function()
{
    alert("Button clicked!");
});
```

When the button is clicked, the event listener detects the event and runs the function.

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

The `addEventListener()` method in JavaScript is used to attach a function (event handler) to an HTML element so it can respond when a specific event occurs.

How `addEventListener()` Works

It listens for an event (like click, mouseover, keydown, etc.) and runs the specified function when that event happens.

Syntax:

```
element.addEventListener("eventType", eventHandlerFunction);
```

Parameters:

Parameter	Meaning
`"eventType"`	The event name as a string (e.g., `"click"`, `"keyup"`)
`eventHandlerFunction`	The function to execute when the event occurs

Example:

HTML:

```
<button id="myBtn">Click Here</button>
<p id="message"></p>
```

JavaScript:

```
document.getElementById("myBtn").addEventListener("click", function() {
    document.getElementById("message").textContent = "Button was clicked!";
});
```

What happens?

The event listener waits for a click on the button
When clicked → The function runs → The text gets updated

Benefits of `addEventListener()`

Allows multiple event handlers on the same element
Keeps JavaScript separate from HTML
Works with modern event types (like custom events)

• DOM Manipulation

• Question 1: What is the DOM (Document Object Model) in JavaScript? How does JavaScript interact with the DOM?
Ans.:

DOM in JavaScript:

DOM (Document Object Model) is a programming interface that represents a webpage as a tree structure of objects.

When a webpage loads, the browser converts the HTML into a DOM tree.

Example DOM structure:

```
Document
 └── html
      ├── head
      └── body
           ├── h1
           └── button
```

Every element (like `<div>`, `<p>`, `<img>`, etc.) becomes an object in the DOM

JavaScript can access, modify, add, or remove these elements using DOM methods

How JavaScript Interacts with the DOM :

JavaScript uses DOM methods to *dynamically* change webpage content and style.

JavaScript can:

Change HTML content
Change CSS styles
Add or remove elements
Handle events (click, keypress, etc.)

Example: Changing Content Using DOM

```
<h2 id="title">Hello</h2>
<button onclick="changeText()">Click Me</button>
```

```
function changeText() {
    document.getElementById("title").textContent = "Welcome to JavaScript
DOM!";
}
```

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

1. getElementById()

Selects a single element using its unique ID
Returns one element

Syntax:

```
document.getElementById("idName");
```

Example:

```
<p id="msg">Hello</p>
<script>
    let element = document.getElementById("msg");
    console.log(element.textContent); // Output: Hello
</script>
```

2. getElementsByClassName()

✔ Selects multiple elements with the same class name
✔ Returns an HTMLCollection (like an array but not exactly)

Syntax:

```
document.getElementsByClassName("className");
```

Example:

```
<p class="item">One</p>
<p class="item">Two</p>
<script>
    let items = document.getElementsByClassName("item");
    console.log(items[0].textContent); // Output: One
```

```
</script>
```

3. querySelector()

Selects the first matching element based on a CSS selector
Very flexible

Syntax:

```
document.querySelector("CSS selector");
```

Example:

```
<h2 class="title">Heading</h2>
<script>
    let title = document.querySelector(".title");
    console.log(title.textContent); // Output: Heading
</script>
```

You can use:

- `#idName` → ID selector
- `.className` → Class selector
- `tagName` → Tag selector

• JavaScript Timing Events (setTimeout, setInterval)

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

JavaScript Timing Functions

JavaScript provides built-in timing functions to schedule tasks:

1. setTimeout()

This method executes a function once after a specified time (in milliseconds).

Syntax:

```
setTimeout(function, time);
```

Example:

```
setTimeout(() => {
  console.log("Hello! This message appears after 3 seconds");
}, 3000); // 3000 ms = 3 seconds
```

Use case:
Show a message after some delay
Animate elements after loading
Redirect user after a delay

2. setInterval()

This method repeatedly executes a function at fixed time intervals.

Syntax:

```
setInterval(function, time);
```

Example:

```
setInterval(() => {
  console.log("This message prints every 2 seconds");
}, 2000);
```

Use case:
Live clock updates
Repeated animations
Auto-refresh elements

Stopping the Timers

Both functions return a timer ID so we can stop them using:

```
clearTimeout(timerID);
clearInterval(intervalID);
```

Example: Stop setInterval after 5 seconds:

```
let count = 0;
let timer = setInterval(() => {
  count++;
  console.log(count);

  if (count === 5) {
    clearInterval(timer);
    console.log("Stopped!");
  }
}, 1000);
```

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

Example :

```
<!DOCTYPE html>
<html>
<body>

<h2>setTimeout() Example</h2>
<button onclick="showMessage()">Click Me</button>
<p id="output"></p>

<script>
function showMessage() {
    setTimeout(() => {
        document.getElementById("output").innerHTML = "Message displayed
after 2 seconds!";
    }, 2000);
}
</script>

</body>
</html>
```

• JavaScript Error Handling

• Question 1: What is error handling in JavaScript? Explain the try, catch, and finally blocks with an example.
Ans.:

try – catch – finally Blocks:

Block	Purpose
try	Contains code that may cause an error
catch	Executes if an error occurs in the `try` block and handles the error
finally	(Optional) Always executes at the end, whether an error occurred or not

Syntax:

```
try {
    // Code that might cause an error
}
catch (error) {
    // Code to handle the error
}
finally {
    // Code that always executes
}
```

Example:

```
try {
    let num = 10;
    console.log(num.toUpperCase()); // Error: number has no
toUpperCase()
}
catch (error) {
    console.log("An error occurred: " + error.message);
}
finally {
    console.log("Execution completed!");
}
```

Output:

```
An error occurred: num.toUpperCase is not a function
```

```
Execution completed!
```

• Question 2: Why is error handling important in JavaScriptapplications?
Ans.:

Error Handling:

Reason	Explanation
Prevents app crashes	Without error handling, a single error can stop the entire script
Improves user experience	Shows helpful messages instead of a broken or frozen webpage
Easier debugging	Helps developers understand what went wrong using error details
Handles unpredictable situations	Errors from user input, APIs, network issues, or missing data can be managed
Increases application reliability	Makes the app more stable and secure

Example:

If a form expects a number but the user types text → error may occur
With error handling → display a message like *"Please enter a valid number"* instead of breaking the code