Module 8

Javascript Introduction

Theory Assignment :

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

JavaScript is a high-level, interpreted programming language mainly used to make web pages interactive and dynamic. It runs in the browser and works alongside HTML (structure) and CSS (design).

Role of Javascript :

1) Adds Interactivity
   - Handles user actions like button clicks, form submission, mouse movements, etc.
   - Example: Showing alerts, dropdown menus, sliders.
2) Dynamic Content Updates
   - Updates content without reloading the page (using DOM manipulation).
   - Example: Live search results, updating text or images dynamically.
3) Form Validation
   - Validates user input on the client side before sending data to the server.
   - Example: Checking email format or required fields.
4) Enhances User Experience
   - Creates smooth animations and transitions.
   - Improves performance and responsiveness of websites.
5) Backend Development
   - With environments like Node.js, JavaScript is also used for server-side development.

Q.2: How is JavaScript different from other programming languages like Python or Java?

Aspect	JavaScript	Python	Java
Primary Use	Web development (frontend & backend)	General-purpose, data science, AI	Enterprise applications, Android
Execution	Runs in the browser & server (Node.js)	Runs on interpreter	Runs on JVM
Typing	Dynamically typed	Dynamically typed	Statically typed
Syntax Style	C-style, flexible	Simple, readable	Verbose, strict
Object-Oriented	Prototype-based	Class-based	Class-based
Compilation	Interpreted (JIT compiled)	Interpreted	Compiled to bytecode
Concurrency	Event-driven, single-threaded	Multi-threading supported	Multi-threading supported

JavaScript is mainly used to make websites interactive and runs directly in the browse

Python is known for simplicity and is widely used in data science, automation, and AI.

Java is strongly typed, more structured, and commonly used in large enterprise systems.

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

The <script> tag in HTML is used to embed or link JavaScript code into a web page. It allows browsers to execute JavaScript to make web pages interactive and dynamic.

- To write inline JavaScript code directly inside an HTML file
- To link external JavaScript files
- To control page behavior such as form validation, events, and DOM manipulation

Linking an External JavaScript File to HTML

You can link an external JavaScript file using the src attribute of the <script> tag.

Steps:

1. Create a JavaScript file (e.g., script.js)
2. Write JavaScript code inside it
3. Link it to the HTML file

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

Lab Assignment (Task)

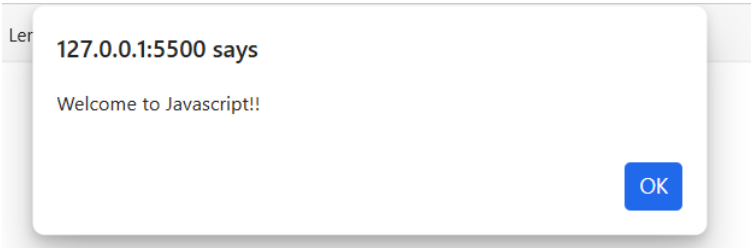
1) Create a simple HTML page and add a <script> tag within the page .

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Web Dev</title>
</head>
```

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

2) Write JavaScript code to display an alert box with the message "Welcome to JavaScript!" when the page loads.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Javascript</title>
    <script>
        alert("Welcome to Javascript!!");
    </script>
</head>
<body>
    <h1>Javascript Example</h1>
    <p>This page shows alert when it loads !!</p>
 </body>
</html>
```



Variables and Data Types

Theory Assignment

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

Variables in JavaScript are used to store data values (such as numbers, text, or objects) that can be used and changed during program execution.

There are three ways to declare Variables in Javascript :

1. var

- Oldest way to declare variables
- global-scoped
- Can be re-declared and updated

Ex : var name = "Aditi";

var age = 22;

2. let

- Introduced in ES6
- Block-scoped
- Can be updated but not re-declared in the same scope

Ex : let city = "Pune";

city = "Mumbai";

3. const

- Introduced in ES6
- Block-scoped
- Cannot be re-declared or updated

Ex: const country = "India";

// country = "USA"; ❌ Error

Q.2) Explain the different data types in JavaScript. Provide examples for each

JavaScript has primitive data types like Number, String, Boolean, Undefined, Null, Symbol, and BigInt, and non-primitive types like Object, Array, and Function, each used to store different kinds of data.

1. Number

Used for integers and floating-point numbers.

Ex : let age = 21;

let price = 99.5;

2. String

Used to store text (characters).

Ex: let name = "Aditi";

let message = 'Hello JavaScript';

3. Boolean

Represents true or false.

Ex: let isLoggedIn = true;

let isAdmin = false;

4. Undefined

A variable declared but not assigned a value.

Ex: let x;

console.log(x); // undefined

5. Null

Represents an intentional empty value.

Ex: let result = null;

6. Symbol (ES6)

Used to create unique identifiers.

Ex: let id = Symbol("id");

7. BigInt

Used for very large integers.

Ex: let bigNumber = 123456789012345678901n;

8. Object

Used to store data in key-value pairs.

Ex: let student = {

  name: "Aditi",

  age: 22

};

9. Array

Used to store multiple values in a list.

Ex: let colors = ["red", "blue", "green"];

10. Function

A block of reusable code.

Ex: function greet() {

  console.log("Hello!");

}

Q.3) What is the difference between undefined and null in JavaScript?

undefined

- Means a variable has been declared but not assigned a value
- Default value given by JavaScript
- Type is undefined

Ex: let x;

console.log(x); // undefined

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

null

- Means no value intentionally
- Assigned by the programmer
- Used to represent empty or unknown values
- Type is object (this is a known JavaScript bug)

let y = null;

console.log(y); // null

console.log(typeof y); // "object"

Lab Assignment (Task)

⇒ 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().

```
// String data type
let name = "Aditi";

// Number data type
let age = 22;

// Boolean data type
let isStudent = true;

// Null data type
let result = null;

// Undefined data type
let city;

console.log(name);
console.log(age);
console.log(isStudent);
console.log(result);
console.log(city);
```

JavaScript Operators

Theory Assignment

Q.1: What are the different types of operators in JavaScript? Explain with examples.

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators

1.Arithmetic Operators

Used to perform mathematical calculations.

```
Ex: let a = 10;

let b = 5;

console.log(a + b);
```

2. Assignment Operators

Used to assign values to variables.

```
Ex: let x = 10;

x += 5;

console.log(x);
```

3.Comparison Operators

Used to compare two values and return true or false.

```
Ex: let p = 5;

let q = "5";

console.log(p === q);
```

4. Logical Operators

Used to combine or invert conditions.

Ex: let age = 20;

let hasID = true;

console.log(age >= 18 && hasID);

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

The operators == and === are used to compare values, but they work differently.

== (Loose Equality)

- Compares only values
- Performs type conversion automatically
- Can give unexpected results

EX: 5 == "5" // true

true == 1 // true

null == undefined // true

=== (Strict Equality)

- Compares both value and data type
- No type conversion
- More reliable and recommended

EX: 5 === "5" // false

true === 1 // false

null === undefined // false

Lab Assignment (Task)

Create a JavaScript program to perform the following:

⇒ Add, subtract, multiply, and divide two numbers using arithmetic operators.

⇒ Use comparison operators to check if two numbers are equal and if one number is greater than the other.

⇒ Use logical operators to check if both conditions (e.g., a > 10 and b < 5) are true.

```
let a = 7;
let b = 5;

console.log("Addition:",a + b);
console.log("Subtraction:",a - b);
console.log("Multipliction:",a * b);
console.log("Division:",a/b);

console.log("Are a and b equal?", a == b);
console.log("Is a greater than b?", a > b);

console.log("Is a > 10 AND b < 5?", a > 10 && b < 5);
```

Control Flow (If-Else, Switch)

Theory Assignment

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

Control flow refers to the order in which JavaScript executes statements in a program.
By default, code runs from top to bottom, but control flow statements allow you to change this order based on conditions, loops, or function calls.

Common control flow statements include:

- if, if-else, else if
- switch
- Loops (for, while, do-while)
- break, continue, return

How if-else Statements Work

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

- The if block runs when the condition is true
- The else block runs when the condition is false

Ex:

```
let age = 20;
if (age >= 18) {
  console.log("You are eligible to vote.");
} else {
  console.log("You are not eligible to vote.");
}
```

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

A switch statement is used to execute different blocks of code based on the value of a single expression. It compares the expression with multiple possible values (**cases**) and runs the code for the matching case.

Syntax :

```
switch (expression) {
  case value1:
    break;
case value2:
    break;


  default:
}
```

How a switch statement works

1. The expression inside switch() is evaluated once.
2. Its value is compared with each case value using strict comparison (===).
3. When a match is found, the corresponding block of code executes.
4. The break statement stops further checking and exits the switch.
5. If no case matches, the default block runs (optional).

When should you use a switch statement instead of if-else?

Use a switch statement when:

- You are checking one variable or expression against multiple fixed values
- The conditions are based on equality

- You want cleaner and more readable code than long if-else chains

Lab Assignment (Task)

⇒ Task 1: Write a JavaScript program to check if a number is positive, negative, or zero using an if-else statement.

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

Task 1 :

```
let number = -5;


if (number > 0) {
  console.log("The number is Positive");
} else if (number < 0) {
  console.log("The number is Negative");
} else {
  console.log("The number is Zero");
}
```

Task 2:

```
let day = 3;


switch (day) {
  case 1:
    console.log("Monday");
```

```
    break;
  case 2:
    console.log("Tuesday");
    break;
  case 3:
    console.log("Wednesday");
    break;
  case 4:
    console.log("Thursday");
    break;
  case 5:
    console.log("Friday");
    break;
  case 6:
    console.log("Saturday");
    break;
  case 7:
    console.log("Sunday");
    break;
  default:
    console.log("Invalid day number");
}
```

Loops (For, While, Do-While)

Theory Assignment

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

In JavaScript, loops are used to execute a block of code repeatedly until a certain condition is met. The main types of loops are for, while, and do-while.

1)for Loop

A for loop is used when you know in advance how many times the loop should run.

Syntax

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

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

2) **while loop**

A while loop runs as long as the condition is true. It is used when the number of iterations is not fixed.

Syntax

```
while (condition) {
  // code to be executed
}
```

```
Ex : let i = 1;


while (i <= 5) {
```

```
  console.log(i);
  i++;
}
```

3) do – while loop

A do-while loop executes the code at least once, even if the condition is false.

Syntax

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

Ex : let i = 1;

```
do {
  console.log(i);
  i++;
} while (i <= 5);
```

Q.2) What is the difference between a while loop and a do-while loop?

Feature	**while loop**	**do-while loop**
Condition check	Before executing the loop body	After executing the loop body
Minimum execution	May execute **zero times**	Executes **at least once**
Syntax	while (condition) { }	do { } while (condition);
Use case	When the loop should run only if condition is true	When the loop must run at least once

while loop checks the condition first, so it may not run at all.

do-while loop runs the code first and checks the condition later, so it always executes at least once.

Lab Assignment (Task)

Task 1: Write a JavaScript program using a for loop to print numbers from 1 to 10.

Task 2: Create a JavaScript program that uses a while loop to sum all even numbers between 1 and 20

Task 3: Write a do-while loop that continues to ask the user for input until they enter a number greater than 10.

Task 1 :

```
for (let i = 1; i <= 10; i++) {
  console.log(i);
}
```

Task 2 :

```
let sum = 0;
let num = 2;

while (num <= 20) {
  sum += num;
  num += 2;
}
```

```
console.log("Sum of even numbers:", sum);
```

Task 3 :

```
let number;
do {
  number = prompt("Enter a number greater than 10:");
} while (number <= 10);
console.log("You entered:", number);
```

Functions

Theory Assignment

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

A function in JavaScript is a block of reusable code that performs a specific task. Instead of writing the same code again and again, you can place it inside a function and call it whenever needed.

Syntax for Declaring a Function:

```
function functionName(parameters) {
   // code to be executed
}
```

Syntax for Calling a Function:

```
functionName(arguments);
```

Q.2: What is the difference between a function declaration and a function expression?

Feature	Function Declaration	Function Expression
Definition	A function defined using the function keyword with a name	A function assigned to a variable
Syntax	function greet() { }	const greet = function() { }
Hoisting	Hoisted (can be called before declaration)	Not hoisted
Usage	Used when function is needed throughout the program	Used when function is needed conditionally or as a value
Name	Function has a name	Can be anonymous or named

Function Declaration Example:

```
greet();

function greet() {
   console.log("Hello World");
}
```

Function Expression Example:

```
greet();

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

Q.3: Discuss the concept of parameters and return values in functions.

1) Parameters

Parameters are variables listed in the function definition.
They receive values (called arguments) when the function is called.

Ex: function greet(name) { // name is a parameter

  console.log("Hello " + name);

}

greet("Aditi"); // "Aditi" is an argument

2) Return values

A return value is the result that a function sends back to where it was called.
The return keyword is used.

Ex: function add(a, b) {

  return a + b;

}

let sum = add(5, 3);

console.log(sum);

Lab Assignment (Task)

⇒ Task 1: Write a function greetUser that accepts a user's name as a parameter and displays a greeting message (e.g., "Hello, John!").

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

Task 1 :

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

greetUser("John");
```

Output :

Hello, John!

Task 2 :

```
function calculateSum(a, b) {
   return a + b;
}

let result = calculateSum(10, 20);
console.log(result);
```

Output :

30

Arrays

Theory Assignment

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

An array in JavaScript is a special variable that stores multiple values in a single variable.
Each value is stored at a numeric index, starting from 0.

Arrays are used to store lists of data like numbers, names, or objects.

How to Declare an Array :

1)Using Square Brackets (Most Common)

let fruits = [];

2) Using the Array Constructor

let fruits = new Array();

How to Initialize an Array :

1) Initialize with Values

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

2) Declare and Initialize Together

let numbers = [10, 20, 30, 40];

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

1) push()

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

Example**:**

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

fruits.push("Mango");

```
console.log(fruits);
```

2) pop()

Removes the last element from an array.

Example:

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

3**)** shift()

Removes the first element from an array.

Example**:**

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

4) unshift()

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

Example**:**

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

Lab Assignment (Task)

⇒ 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.

⇒ Task 2:

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

Task 1 :

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


fruits.push("mango");
fruits.shift();
console.log(fruits);
```

Output :

["banana", "cherry", "mango"]

Task 2 :

```
let numbers = [10, 20, 30, 40];
let sum = 0;

for (let i = 0; i < numbers.length; i++) {
   sum += numbers[i];
}
```

```
console.log("Sum of array elements:", sum);
```

Output :

Sum of array elements: 100

Objects

Theory Assignment

Q.1) What is an object in JavaScript? How are objects different from arrays?

An object in JavaScript is a collection of data stored as key–value pairs.
It is used to represent real-world entities by grouping related properties and behaviors.

Example of an Object

```
let student = {
   name: "Aditi",
   age: 21,
   course: "IT"
};
```

Feature	Object	Array
Data storage	Key–value pairs	Ordered list of values
Access method	Using keys	Using index numbers
Order	Not guaranteed	Maintains order
Use case	Represents real-world entities	Stores lists of similar items
Syntax	{}	[]

Q.2) Explain how to access and update object properties using dot notation and bracket notation.

In JavaScript, object properties can be accessed and updated in two ways:

1. Dot notation
2. Bracket notation

1 .Dot notation :

Accessing property :

```
let student = {
   name: "Aditi",
   age: 21
};
console.log(student.name);
```

Updating property :

```
student.age = 22;
console.log(student.age);
```

2.Bracket Notation

Accessing a Property

```
let student = {
   name: "Aditi",
   age: 21
};
console.log(student["name"]);
```

Updating a Property

```
student["age"] = 22;
```

Lab Assignment

Task :

⇒ Create a JavaScript object car with properties brand, model, and year. Use JavaScript to:

• Access and print the car's brand and model.

• Update the year property.

• Add a new property color to the car object.

Task :

```
let car = {
   brand: "Toyota",
   model: "Corolla",
   year: 2020
};
console.log("Brand:", car.brand);
console.log("Model:", car.model);

car.year = 2023;

car.color = "White";
console.log(car);
```

JavaScript Events

Theory Assignment

Q.1) What are JavaScript events? Explain the role of event listeners.

JavaScript events are actions or occurrences that happen in the browser, usually triggered by the user or the system. These events allow JavaScript to respond to interactions and make web pages interactive.

Common examples of events:

click – when a user clicks an element

mouseover – when the mouse moves over an element

keydown – when a key is pressed

submit – when a form is submitted

load – when a page finishes loading

Role of Event Listeners :

Key roles of event listeners:

- They detect user actions (like clicks or typing).
- They connect events to JavaScript functions.
- They help create dynamic and interactive web applications.
- They keep JavaScript code separate from HTML, improving readability and maintenance.

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

The addEventListener() method is used to attach an event handler to an HTML element. It tells JavaScript to *listen for a specific event* and execute a function when that event occurs.

Syntax :

element.addEventListener(event, function, useCapture);

How it Works (Step-by-step)

1. Select an HTML element.
2. Specify the event to listen for.
3. Provide a function to execute when the event happens.
4. When the event occurs, the function runs automatically.

Ex:

```
<button id="myBtn">Click Me</button>
<script>
  document.getElementById("myBtn").addEventListener("click", function() {
    alert("Button clicked!");
  });
</script>
```

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 lang="en">
<head>
  <meta charset="UTF-8">
  <title>Button Click Event</title>
</head>
<body>

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

  <script>
    document.getElementById("btn").addEventListener("click", function() {
      alert("Button clicked!");
    });
  </script>

</body>
</html>
```

DOM Manipulation

Theory Assignment

Q.1) What is the DOM (Document Object Model) in JavaScript? How does JavaScript interact with the DOM?

The DOM (Document Object Model) is a programming interface that represents an HTML document as a tree-like structure of objects.
Each part of the webpage (elements, attributes, text) is treated as an object (node) that JavaScript can access and manipulate.

Ex :

```
<html>
  <body>
    <h1>Hello</h1>
    <p>Welcome</p>
  </body>
</html>
```

How Does JavaScript Interact with the DOM?

JavaScript interacts with the DOM to read, change, add, or remove content and respond to user actions.

1) Accessing Elements

```
document.getElementById("title");
document.querySelector(".box");
```

2. Changing Content

```
document.getElementById("title").innerHTML = "New Text";
```

3. Modifying Styles

```
document.getElementById("title").style.color = "blue";
```

4. Adding Event Listeners

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

5. Creating & Removing Elements

```
let p = document.createElement("p");
p.textContent = "New Paragraph";
document.body.appendChild(p);
```

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

DOM Element Selection Methods in JavaScript

JavaScript provides different methods to select elements from the DOM so you can read, modify, or apply events to them.

1)getElementById()
Selects one element using its unique id.

Syntax:

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

2)getElementsByClassName()

Purpose:
Selects all elements that have a specific class name.

Syntax:

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

3)querySelector()
Selects the first element that matches a CSS selector.

Syntax:

document.querySelector("selector");

Lab Assignment (Task)

⇒ Create an HTML page with a paragraph (<p>) 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.

Task :

```
<!DOCTYPE html>
<html lang=”en”>
<head>
  <meta charset=”UTF-8”>
  <title>DOM Example</title>
</head>
<body>

  <p id=”text”>Hello, World!</p>

  <script>

    const para = document.getElementById(“text”);
```

```
    para.innerHTML = “JavaScript is fun!”;
   para.style.color = “blue”;
 </script>

</body>
</html>
```

JavaScript Timing Events (setTimeout, setInterval)

Theory Assignment

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

JavaScript provides timing functions that allow code to run after a delay or repeatedly at fixed intervals. These are commonly used for animations, notifications, and timed actions.

1)setTimeout()

setTimeout() executes a function once after a specified time delay.

Syntax :

setTimeout(function, delay);

2)setInterval()

setInterval() executes a function repeatedly at fixed time intervals.

Syntax :

setInterval(function, interval);

How They Are Used for Timing Events

- Delay showing messages or pop-ups
- Create countdowns or timers

- Run animations step-by-step
- Update content automatically at intervals

Lab Assignment (Task)

Theory Assignment :

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

Task 1:

```
<!DOCTYPE html>
<html>
<head>
  <title>Background Color Change</title>
</head>
<body>

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

  <script>
    setTimeout(function() {
      document.body.style.backgroundColor = "lightblue";
    }, 5000);
  </script>
```

```
    </body>
    </html>
```

Task 2:

```
<!DOCTYPE html>
<html>
<head>
  <title>Digital Clock</title>
</head>
<body>

  <h2 id="clock"></h2>

  <script>
    setInterval(function() {
      let now = new Date();
      document.getElementById("clock").innerHTML =
        now.toLocaleTimeString();
    }, 1000);
  </script>

</body>
</html>
```

JavaScript Error Handling

Theory Assignment

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

Error handling in JavaScript is the process of detecting, managing, and responding to runtime errors so that the program does not crash and continues to run smoothly.

1)try Block

- Contains code that may cause an error.
- JavaScript tries to execute this code.

2) catch Block

- Executes only if an error occurs in the try block.
- Receives the error object.

3) finally Block

- Executes always, whether an error occurs or not.
- Used for cleanup tasks.

Ex:

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

 <script>
   try {
     let result = 10 / x;   // x is not defined (error)
     console.log(result);
   } catch (error) {
```

```
    console.log("An error occurred:", error.message);
  } finally {
    console.log("Program execution completed.");
  }
 </script>

</body>
</html>
```

Q.2) Why is Error Handling Important in JavaScript Applications?

Error handling is important because it helps applications handle unexpected problems gracefully instead of crashing or behaving unpredictably.

1) Prevents Application Crashes
Without error handling, a single error can stop the entire program. Error handling ensures the app keeps running.

2) Improves User Experience
It allows you to show friendly error messages instead of broken pages or blank screens.

3) Helps in Debugging
Errors can be caught and logged, making it easier for developers to identify and fix issues.

4) Ensures Program Stability
Critical code continues to execute even if an error occurs, especially using the finally block.

5) Handles Unexpected Situations
Network failures, invalid inputs, or missing data can be managed safely.

6) Improves Application Reliability
Well-handled errors make applications more robust and professional.

Lab Assignment (Task) :

• Write a JavaScript program that attempts to divide a number by zero. Use try- catch to handle the error and display an appropriate error message.

```
<!DOCTYPE html>
<html>
<body>
<script>
  try {
    let a = 10;
    let b = 0;

    if (b === 0) {
      throw new Error("Cannot divide by zero");
    }
   let result = a / b;
    console.log(result);

  } catch (error) {
    console.log("Error:", error.message);
  }
</script>

</body>
</html>
```