Js Assignment

Js Introduction

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

- JavaScript is a high-level, interpreted programming language that is widely used to make web pages interactive and dynamic.

Role of Js

1 Interactivity and User Experience:

- Adds dynamic behavior to web pages, such as dropdown menus, modal windows, and carousels.
- Enhances user experience by allowing seamless interaction without page reloads.

2 DOM Manipulation:

- Allows developers to dynamically modify the HTML and CSS of a webpage after it has been loaded.
- Enables features like live content updates, form validation, and animations.

3 Event Handling:

- JavaScript listens for user events like mouse clicks, key presses, and touch gestures to trigger specific actions.

4 **Asynchronous Operations**:

- Supports asynchronous programming using techniques like **AJAX** and **fetch** APIs, enabling real-time data fetching without refreshing the page.
- Plays a key role in single-page applications (SPAs).

5 **Cross-Platform Development**:

- Frameworks like **React Native** allow developers to use JavaScript for building mobile apps.
- Tools like **Electron.js** are used to create desktop applications.

6 **Integration with Other Web Technologies**:

- Works seamlessly with HTML and CSS to build robust web applications.
- Often used with libraries (e.g., jQuery) and frameworks (e.g., React, Angular, Vue.js) to speed up development and manage complexity.

7 **Server-Side Development**:

- With Node.js, JavaScript extends its capabilities to the server, enabling full-stack development using a single language.

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

Differences Between JavaScript, Python, and Java

1 Primary Use:

- JavaScript: Best for web development (interactive webpages, client-side scripts).
- Python: General-purpose, popular for data science, machine learning, and scripting.
- Java: Ideal for large-scale enterprise systems, Android apps, and backend development.

2 Execution Environment:

- JavaScript: Runs in browsers and on servers (via Node.js).
- Python: Runs on any system with a Python interpreter.
- Java: Runs on the Java Virtual Machine (JVM).

3 Typing System:

- JavaScript: Dynamically typed (types are inferred).
- Python: Dynamically typed, but focuses on simplicity and readability.
- Java: Statically typed (variable types must be explicitly declared).

4 Syntax and Complexity:

- JavaScript: Lightweight and flexible, but can become complex in larger applications.
- Python: Known for its simple, clean, and beginner-friendly syntax.
- Java: More verbose, with strict syntax rules.

5 Performance:

- JavaScript: Optimized for web tasks, fast in browser environments.
- Python: Slower for computational tasks but excellent for rapid development.
- Java: Faster than Python, suitable for high-performance applications.

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

- The <script> tag is used in HTML to embed JavaScript code or reference external JavaScript files. It allows developers to add interactivity and dynamic behavior to web pages.

  Two way script tag

  1. **inline script tag:-** JavaScript code is written directly within the <script> tag.

     Ex.
     ```
     <script>
       alert("hello")
       </script>
     ```

  2. **external script tag**:- An external JavaScript file is linked using the src attribute.

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

Lab Assignment(Task)

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

Input:- <!DOCTYPE html>

```
<html lang="en">
 <head>
 <meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Simple HTML with Script</title>
</head>
<body>
<script>
Alert(“hello js!”)
</script>
</body>
</html>
```

Output:-pop ub box print hello js!

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

Input:- <!DOCTYPE html>

```
<html lang="en">
 <head>
 <meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Simple HTML with Script</title>
</head>
<body>
<script>
Alert(“Welcome toJavaScript!”)
</script>
</body>
```

</html>

Output:-pop ub box print Welcome toJavaScript!

JavaScript Operators

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

1. **Arithmetic operators:-** Addition, Subtraction, Multiplication, Division, Modulus (Remainder), Exponentiation, Increment,and Decrement included arithmetic operators.
2. **Assignment operators:-** Assign, Add and Assign, Subtract and Assign, Multiply and Assign, Divide and Assign, Modulus and Assign,and Exponentiation and Assign included assignment operators.
3. **Comparison operators**:- Equal to, Strict Equal to, Not Equal to, Strict Not Equal to, Greater than, Less than, Greater than or Equal to and Less than or Equal to included comparison operators.
4. **Logical operators:-** Logical AND, Logical OR and Logical NOT included logical operators.

   Example:- // Arithmetic Operators
   ```
   let sum = 10 + 5; // 15

   // Assignment Operators
   let a = 10;
   a += 5; // 15

   // Comparison Operators
   console.log(5 === "5"); // false

   // Logical Operators
   let isTrue = (10 > 5) && (5 < 8); // true
   ```

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

== is only check value not check datatype. And === is check both value and datatype.

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

Input:- let num1=10

```
let num2=10/19

document.write(num1==num2)
```

output:-true/false

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

```
var a=15

var b=2

document.write(a>10 && b<5 )
```

output:-true

Control Flow (If-Else, Switch)

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

if-else statements are used to execute a block of code conditionally, based on whether an expression evaluates to true or false.

example:- Write a JavaScript program to check if a number is positive, negative, or zero usingan if-else statement.

```
 let number = 1;//-1,0

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.");

}
```

Output:-positive,negative,zero

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

A switch statement is a control structure used to evaluate an expression against multiple possible values and execute the corresponding block of code. It is an alternative to a series of if-else statements when dealing with multiple discrete values.

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

```
 var day = 9;//changeable
   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 input. Please enter a number between 1 and 7.");
}
```

Output:- Invalid input. Please enter a number between 1 and 7.

Loops (For, While, Do-While)

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

1. For Loop

A for loop is used when the number of iterations is known. It includes an initialization, a condition, and an increment/decrement statement.

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

```
 for( var a=1;a<=10;a++){
        document.write(a)
      }
```

Output:-1 to 10 print

2. While Loop

A while loop is used when the number of iterations is not known in advance, and it continues as long as the condition is true.

Example:- Create a JavaScript program that uses a while loop to sum all even numbers

```
let sum = 0;
        let num = 0;
        let limit = 100; // Set the limit for the sum of even numbers
        while (num <= limit) {
          if (num % 2 === 0) {
            sum= sum+num
          }
          num++;
        }
        document.write("The sum of all even numbers from 0 to 100 is: "+ sum);
```

output:- The sum of all even numbers from 0 to 100 is: 2550

3. Do-While Loop

A do-while loop is similar to a while loop, but it guarantees at least one execution of the block before checking the condition.

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

```
var number;
```

```
    do {
        number = parseInt(prompt("Please enter a number greater than
10:"));
    } while (number <= 10);


    document.write("You entered a number greater than 10: " + number);
```

output:-100

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

 while loop: Checks the condition before executing the code block. If the condition is false initially, the loop won't run.

 do-while loop: Executes the code block at least once and checks the condition after running the code block. If the condition is false after the first execution, it will stop, but the loop always runs at least once.

Functions

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

definition:-collection of code that code we will use for particular task.

Types:- 1 tradistion fun - fun keyword and fun name

2 anonymous fun - fun keyword no name

3 arrow fun - no fun keyword and name //es6

Syntax & example:- 1 tradistion fun

```
function addtocart(){


    let a=5;
    let b=6;


    let c= a+b;


    document.write(c)
```

2 anonymous fun

```
 let v=    function(){


                  let a=50;
                  let b=60;


                   let c= a+b;


                 document.write(c)
```

3 arrow fun

```
 let n=  ()=>{


               let a=50;
                  let b=20;


                   let c= a+b;


                 document.write(c)


              }
```

Declaring & calling:- n()

Functionname_()

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

Function Declaration:

A function declaration defines a named function using the function keyword, followed by the function name, parameters, and body.

Suntax & example:-

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

Function declarations are **hoisted**, meaning the JavaScript engine loads them during the compilation phase. As a result, you can invoke the function before its actual definition in the code.

```
greet("Alice"); // Output: Hello, Alice!
```

Function Expression:

A function expression involves defining a function and assigning it to a variable. Function expressions can be anonymous or named.

```
const greet = function(name)

{

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

};
```

Function expressions are **not hoisted**. This means you cannot invoke them before their definition; attempting to do so results in an error.

```
greet("Alice"); // Error: Cannot access 'greet' before initialization
```

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

Parameters:-

Parameters are placeholders specified in a function's definition, representing values that the function expects to receive when invoked. They enable functions to operate on different inputs without altering the function's code.

Example:- function multiply(a, b) {

  return a * b;

}In this example, a and b are parameters of the multiply function. When calling this function, you provide **arguments**—the actual values corresponding to these parameters.

Calling the Function:

```
let product = multiply(4, 5); // product is now 20
```

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

```
function greetUser(name) {
  alert("Hello, " + name + "!");
}
greetUser("John"); // Output: Hello, John!
```

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

```
function calculateSum(a,b){
            return c;
            let c=a+b;
           document.write(c)
       }
       let result=calculateSum(10,20)
document.write(result); // This will display 30 on the webpage
```

Arrays

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

collection of data with diff diff datatype

example:-

```
let mixedArray = [42, "hello", true, { name: "Alice" }, [1, 2, 3]];
console.log(mixedArray)
```

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

Push():-

```
let a=[1,2,3]
a.push(4)
console.log(a)//1,2,3,4
```

pop():-

```
let a=[1,2,3]
a.pop()
console.log(a)//1,2
```

```
shift()

let a=[1,2,3]

a.shift()

console.log(a)//2,3

unshift()

let a=[1,2,3]

a.unshift(54)

console.log(a)//54,1,2,3
```

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("orange");

// Remove the first fruit from the array

fruits.shift();

// Log the modified array to the console

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

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

```
 function sumOfElements(arr) {

          let total = 0;

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

             total += arr[i];

          }

          return total;

       }

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

       const result = sumOfElements(numbers);
```

```
document.write("Sum of elements:", result);//15
```

Objects

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

object is a non primitve datatype. Object multiple data store. object store data property and value form.

Feature	**Object**	**Array**
Structure	Collection of key-value pairs	Ordered list of values
Data Access	Accessed by **keys** (strings or symbols)	Accessed by **indices** (numerical)
Order	No guaranteed order of properties	Maintains order of elements
Use Case	Best for representing real-world entities	Best for ordered collections

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

1. Dot Notation

Dot notation is the most straightforward and commonly used method for accessing and updating object properties. It involves using a period (.) between the object name and the property name.

```
const person = {
   name: "Alice",
   age: 30,
   isEmployed: true
};

console.log(person.name);  // Outputs: Alice
console.log(person.age);   // Outputs: 30
```

Updating Properties with Dot Notation:

```
person.age = 31;  // Updates the 'age' property

console.log(person.age);  // Outputs: 31
```

2. Bracket Notation

Bracket notation allows you to access and update object properties using square brackets ([]). The property name is specified as a string inside the brackets.

Accessing Properties with Bracket Notation:

```
const person = {
   "first name": "Alice",
   age: 30,
   isEmployed: true
};


console.log(person["first name"]);  // Outputs: Alice
console.log(person["age"]);        // Outputs: 30
```

Updating Properties with Bracket Notation:

```
person["age"] = 31;  // Updates the 'age' property
console.log(person["age"]);  // Outputs: 31
```

Task: ⇒ Create a JavaScript object car with properties brand, model, and year. UseJavaScript to: • Access and print the car’s brand and model. • Update the year property. • Add a new property color to the car object.

```
 let car={
          brand:"bmw",
          model:"m2",
          year:2025
       }
       document.write(car.brand)
       document.write(car.model)
       car["year"]=2025
       document.write(car["year"])
       car.color="red"
```

```
document.write(car.color)
```

JavaScript Events

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

In JavaScript, events are actions or occurrences—such as user interactions or browser-triggered activities—that can be detected and responded to within a web application. Common examples include clicking a button, loading a webpage, pressing a key, or moving the mouse over an element.

Event listeners play a crucial role in handling these events. An event listener is a function that waits for a specific event to occur on a particular element and then executes a block of code when that event happens. This mechanism allows developers to define how a webpage should respond to different user interactions, enabling the creation of dynamic and interactive web applications.

To add an event listener to an element, the addEventListener() method is commonly used. This method allows multiple event listeners to be attached to the same element without overwriting existing ones, and it can handle different types of events.

Here's an example of how to use addEventListener() to handle a button click event: // Select the button element

```
const button = document.querySelector('button');

// Define the function to be executed when the event occurs

function changeColor() {

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

}

// Attach the event listener to the button

button.addEventListener('click', changeColor);
```

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

The addEventListener() method in JavaScript allows you to attach an event handler to a specific element, enabling your code to respond to user interactions such as clicks, key presses, or mouse movements.

```
<!DOCTYPE html>

<html lang="en">

<head>
```

```
  <meta charset="UTF-8">
  <title>addEventListener Example</title>
</head>
<body>
  <button id="myButton">Click Me!</button>
  <script>
    // Select the button element by its ID
    const button = document.getElementById('myButton');
    // Define the event handler function
    function handleClick() {
      alert('Button was clicked!');
    }
    // Attach the event listener to the button
    button.addEventListener('click', handleClick);
  </script>
</body>
</html>
```

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 Alert</title>
</head>
<body>
  <!-- Button element -->
  <button id="alertButton">Click Me!</button>
  <!-- JavaScript to handle the button click event -->
  <script>
    // Select the button element by its ID
    const button = document.getElementById('alertButton');

    // Define the function to be executed when the button is clicked
    function showAlert() {
      alert('Button clicked!');
    }

    // Attach the event listener to the button
    button.addEventListener('click', showAlert);
  </script>
</body>
</html>
```

DOM Manipulation

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

The Document Object Model (DOM) is a programming interface for web documents. It represents the structure of a document—such as an HTML or XML file—as a tree of objects, allowing programming languages like JavaScript to interact with and manipulate the document's content, structure, and style.

JavaScript Interaction with the DOM:

JavaScript interacts with the DOM to create dynamic and interactive web pages. Through the DOM, JavaScript can:

- **Access Elements:** Retrieve elements using methods like getElementById, getElementsByClassName, or querySelector.

  ```
  // Access an element by its ID
  const element = document.getElementById('myElement');
  ```

  Modify Content: Change the text or HTML content of elements.

  ```
  // Change the inner HTML of an element
  ```

element.innerHTML = 'New Content';

Change Styles: Alter the CSS styles of elements.

// Change the background color of an element

element.style.backgroundColor = 'lightblue';

Handle Events: Respond to user interactions by attaching event listeners.

// Add a click event listener to a button

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

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

  alert('Button clicked!');

});

Manipulate Structure: Add, remove, or rearrange elements in the document.

// Create a new paragraph element

const newParagraph = document.createElement('p');

newParagraph.textContent = 'This is a new paragraph.';

// Append the new paragraph to the body

document.body.appendChild(newParagraph);

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

1. getElementById()

- **Purpose:** Selects a single element based on its unique id attribute.
- **Syntax:** document.getElementById('idName')
- **Returns:** The element object with the specified id, or null if no such element exists.
- **Example:**

  // HTML: <div id="content">...</div>

  const element = document.getElementById('content');

  2. getElementsByClassName()

  - **Purpose:** Selects all elements that have a specified class name.

- **Syntax:** document.getElementsByClassName('className')
- **Returns:** A live HTMLCollection of elements with the specified class name(s). This collection updates automatically when the document changes.
- **Example:**

```
// HTML: <p class="text">Paragraph 1</p>

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

const elements = document.getElementsByClassName('text');
```

3. querySelector()

- **Purpose:** Selects the first element that matches a specified CSS selector.
- **Syntax:** document.querySelector('selector')
- **Returns:** The first element that matches the selector, or null if no matches are found.
- **Example:**

```
// HTML: <p class="intro">Introduction</p>

const element = document.querySelector('.intro');
```

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 lang="en">

<head>

  <meta charset="UTF-8">

  <title>Change Paragraph with JavaScript</title>

</head>

<body>

  <!-- Paragraph with initial text -->

  <p id="myParagraph">Hello, World!</p>
  <!-- JavaScript to modify the paragraph -->
  <script>
    // JavaScript code will be added here
  </script>
</body>
</html>
<script>
  // Select the paragraph element by its ID
  const paragraph = document.getElementById('myParagraph');

  // Change the text inside the paragraph
  paragraph.textContent = 'JavaScript is fun!';

  // Change the color of the paragraph to blue
  paragraph.style.color = 'blue';
</script>
```

JavaScript Timing Events (setTimeout, setInterval)

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

setTimeout()

- Purpose: Executes a function or a block of code once after a specified delay.
- Syntax: setTimeout(function, delay, ...args)
    - function: The function to execute.
    - delay: Time in milliseconds to wait before executing the function.
    - ...args: Optional arguments to pass to the function.
- Returns: A unique identifier (timeout ID) that can be used to cancel the timeout with clearTimeout().
- Example:

```
function greet(name) {
  console.log(`Hello, ${name}!`);
}
// Execute 'greet' after 2 seconds
const timeoutId = setTimeout(greet, 2000, 'Alice');
```

- **Use Case:** Ideal for delaying the execution of a function by a specified amount of time.

setInterval()

- **Purpose:** Executes a function or a block of code repeatedly at specified intervals.
- **Syntax:** setInterval(function, interval, ...args)
  - function: The function to execute.
  - interval: Time in milliseconds between each execution of the function.
  - ...args: Optional arguments to pass to the function.
- **Returns:** A unique identifier (interval ID) that can be used to cancel the interval with clearInterval().
- **Example:**

```
let count = 0;

function incrementCounter() {
  count += 1;
  console.log(`Counter: ${count}`);
  if (count === 5) {
    clearInterval(intervalId); // Stop after 5 iterations
  }
}

// Execute 'incrementCounter' every second
const intervalId = setInterval(incrementCounter, 1000);
```

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

```
function greet() {
  console.log('Hello, World!');
}

// Execute 'greet' after 2 seconds (2000 milliseconds)

setTimeout(greet, 2000);
```

⇒ Task 1: • Write a program that changes the background color of a webpage after 5 secondsusing setTimeout().

```
function changeBackgroundColor() { document.body.style.backgroundColor = 'lightblue'; } // Set a timeout to change the background color after 5 seconds setTimeout(changeBackgroundColor, 5000);
```

⇒ Task 2: • Create a digital clock that updates every second using setInterval().

```
#clock { font-size: 48px; font-family: 'Arial', sans-serif; text-align: center; margin-top: 20%; }

<script>

  // Function to update the clock

  function updateClock() {

    const now = new Date();

    const hours = String(now.getHours()).padStart(2, '0');

    const minutes = String(now.getMinutes()).padStart(2, '0');

    const seconds = String(now.getSeconds()).padStart(2, '0');

    const timeString = `${hours}:${minutes}:${seconds}`;

    document.getElementById('clock').textContent = timeString;

  }

  // Update the clock immediately

  updateClock();

  // Set an interval to update the clock every second

  setInterval(updateClock, 1000);

 </script>
```

JavaScript Error Handling

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

In JavaScript, error handling allows developers to manage runtime errors gracefully, ensuring that the application continues to function smoothly even when unexpected situations arise. The primary mechanism for error handling in JavaScript is the try...catch...finally statement, which enables the detection and handling of exceptions.

try...catch...finally Statement

The try...catch...finally statement consists of three blocks:

1. try Block: Contains the code that may potentially throw an error.
2. catch Block: Handles the error if one occurs in the try block.
3. finally Block: Contains code that executes regardless of whether an error occurred, typically used for cleanup operations.

Example:-

```
function divide(a, b) {
  try {
    if (b === 0) {
      throw new Error('Division by zero is not allowed.');
    }
    return a / b;
  } catch (error) {
    console.error(`Error: ${error.message}`);
  } finally {
    console.log('Execution of the divide function is complete.');
  }
}
const result = divide(10, 0);
console.log(result); // undefined
```

Question 2: Why is error handling important in JavaScript applications?

Error handling is important in JavaScript applications for the following key reasons:

1. Prevents Application Crashes: Without proper error handling, unanticipated errors can cause the application to crash or behave unpredictably. By catching and handling errors, the application can continue running smoothly, even when unexpected issues occur.

2. Improves User Experience: When errors are caught and managed correctly, users are presented with clear, informative messages instead of the application failing silently or displaying cryptic errors. This creates a more reliable and user-friendly experience.
3. Ensures Application Stability: Error handling allows the application to handle edge cases and unexpected conditions gracefully, ensuring that it remains stable and functions as expected under various scenarios.
4. Aids in Debugging and Maintenance: Proper error handling provides detailed error messages and logs, making it easier for developers to debug and identify the source of issues. This contributes to more maintainable code in the long term.
5. Reduces Data Loss and Inconsistencies: By handling errors that could otherwise result in data corruption or loss (e.g., database or network failures), error handling ensures that data is not compromised and that the application can recover gracefully.

   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.

```
function divide(a, b) {
  try {
    if (b === 0) {
      throw new Error("Cannot divide by zero.");
    }
    return a / b;
  } catch (error) {
    console.error(`Error: ${error.message}`);
    alert(`Error: ${error.message}`);
  }
 }
 // Attempt to divide 10 by 0
 divide(10, 0);
```