**Javascript Essentials And Advanced**

1. **JavaScript Introduction**

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

Ans. : JavaScript is a dynamic, programming language used to create interactive and dynamic features for websites. It's a core technology of the web, allowing developers to manipulate HTML and CSS, handle user interactions, and create engaging user experiences.

**Role of JavaScript in Web Development:**

**Interactivity**: JavaScript allows users to interact with a webpage through elements like buttons, forms, sliders, and other UI components.

**DOM Manipulation**: JavaScript can read and modify the Document Object Model (DOM), enabling real-time updates to the page content without refreshing.

**Form Validation**: It is widely used to validate user inputs before they are sent to a server.

**Animations and Effects**: JavaScript can animate HTML elements, add effects, and enhance the overall user experience.

**Backend Communication**: Using AJAX or fetch APIs, JavaScript can communicate with servers to send/receive data asynchronously.

**Frameworks and Libraries**: Tools like React, Angular, and Vue are built on JavaScript and are widely used to develop modern web applications.

### ****Question 2: How is JavaScript different from other programming languages like Python or Java?****

**Answer:**

| **Feature** | **JavaScript** | **Python** | **Java** |
| --- | --- | --- | --- |
| **Type** | Scripting language for web | General-purpose, interpreted language | Object-oriented, compiled language |
| **Syntax** | C-like syntax | Readable and simple syntax | Strict, verbose syntax |
| **Execution** | Runs in browsers | Runs via interpreter | Compiled to bytecode |
| **Use Case** | Frontend & Backend web development | Web, AI, ML, automation, data science | Enterprise apps, Android apps |
| **Environment** | Browser-based or Node.js | Python Interpreter | Java Virtual Machine (JVM) |
| **Typing** | Dynamically typed | Dynamically typed | Statically typed |

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

**Answer:**

The <script> tag in HTML is used to embed or reference JavaScript code in a web page. It can either contain inline JavaScript code or link to an external JavaScript file.

Inline example : <script> alert(“Inline Javascript”)</script>

Externel example: <script src=”Javascript.js”></script>

· This tag is usually placed **before the closing** </body> **tag** for better performance.

· The browser loads and executes the JavaScript from the given file.

**Lab Assignment**

### ****Task:**** Create a simple HTML page and add a <script> tag within the page. 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">

  <title>JavaScript Alert Example</title>

</head>

<body>

  <p>This is a simple HTML page with a JavaScript alert.</p>

  <script>

    alert("Welcome to JavaScript!");

  </script>

</body>

**How it works:**

When the page loads in the browser, the script runs automatically.

The alert() function displays a popup with the welcome message.

**2 Variables and Data Types**

## ****Theory Assignment****

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

In JavaScript, **variables** are containers used to store data values. They allow programmers to store, update, and retrieve information in a program.

There are three ways to declare variables in JavaScript:

1. **var** – The traditional way. It is function-scoped and allows re-declaration.

var x = 10;

var x = 20; // Allowed

**2. let** – Introduced in ES6. It is block-scoped and does **not** allow re-declaration within the same scope.

let y = 15;

y = 25; // Allowed//

let y = 30; // Error

1. **const** – Also block-scoped. It must be initialized at declaration and cannot be reassigned.

const z = 5;

// z = 10; // Error

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

JavaScript supports the following main data types:

1. **String** - Text data enclosed in quotes

Let name = "Amit";

2 **Number** - Numeric data (Integer or Float)

let age = 25;

  let price = 99.99;

3 **Boolean** - true or false value

let isStudent = true;

4 **Undefined** - A variable declared but not assigned any value

let x;

5 **Null** - Represents intentional absence of value

let emptyValue = null;

6 **Object** - A collection of key-value pairs

let person = { name: "Amit", age: 25 };

7 **Array** - A special type of object used to store ordered data

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

8 **Symbol** - A unique and immutable primitive value

let sym = Symbol("id");

9 **BigIng** - For vary large integers beyond the safe range of number

let bigNum = 1234567890123456789012345678901234567890n;

### ****Question 3: What is the difference between**** undefined ****and**** null ****in JavaScript?****

| **Feature** | **undefined** | **null** |
| --- | --- | --- |
| Meaning | A variable is declared but not assigned a value. | Represents Intentionaly absence value |
| Type | Undefined | Object |
| Assigned By | JavaScript by default | By developer |
| Example | let x; console.log(x); // undefined | let x = null; console.log(x); // null |

## ****Lab Assignment****

**Task**

### ****JavaScript Program: Declare Variables and Log Values & Types****

let name = "Amit";           // String

let age = 21;                // Number

let isStudent = true;        // Boolean

let emptyValue = null;       // Null

let notAssigned;             // Undefined

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 emptyValue:", emptyValue, "| Type:", typeof emptyValue);

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

### ****Expected output in console****

Value of name: Amit | Type: string

Value of age: 21 | Type: number

Value of isStudent: true | Type: boolean

Value of emptyValue: null | Type: object

Value of notAssigned: undefined | Type: undefined

**JavaScript Operators**

## ****Theory Assignment****

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

Javascript operators are used to perform different types of mathematical and logical computations.

There are different types of JavaScript operators:

#### 1. ****Arithmetic Operators****

These are used to perform basic mathematical operations.

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| + | Addition | 5 + 3 = 8 |
| - | Subtraction | 5 - 2 = 3 |
| \* | Multiplication | 4 \* 2 = 8 | |
| / | Division | 10 / 2 = 5 |
| % | Modulus (remainder) | 10 % 3 = 1 |
| ++ | Increment | let x = 5; x++; // x becomes 6 |
| -- | Decrement | let y = 5; y--; // y becomes 4 |

#### 2. ****Assignment Operators****

Used to assign values to variables.

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| = | Assign | let x = 10; |
| += | Add and assign | x += 5; // x = x + 5 |
| -= | Subtract and assign | x -= 2; // x = x - 2 |
| \*= | Multiply and assign | x \*= 3; // x = x \* 3 |
| /= | Divide and assign | x /= 2; // x = x / 2 |
| %= | Modulus and assign | x %= 4; // x = x % 4 |

#### 3. ****Comparison Operators****

Used to compare two values.

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| == | Equal to | 5 == '5' // true |
| === | Equal value and type | 5 === '5' // false |
| != | Not equal | 5 != 8 // true |
| !== | Not equal value or type | 5 !== '5' // true |
| > | Greater than | 8 > 5 // true |
| < | Less than | 3 < 5 // true |
| >= | Greater than or equal | 5 >= 5 // true |
| <= | Less than or equal | 3 <= 4 // true |

#### 4. ****Logical Operators****

Used to combine multiple conditions.

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| && | Logical AND | (a > 5 && b < 10) |
| || | Logical OR | (a > 5 || b < 10) |
| ! | Logical NOT | !(a > 5) |

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

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| == | Compares only **value**, performs type coercion | 5 == '5' // true |
| === | Compares **value and type** strictly | 5 === '5' // false |

* == allows different types and tries to convert them to match.
* === checks for both **value and data type** to be exactly the same.

**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 onenumber is greater than the other. Use logical operators to check if both conditions (e.g., a > 10 and b < 5)are true.**

<!DOCTYPE html>

<html>

<head>

  <title>JavaScript Operator Assignment</title>

</head>

<body>

  <h2>JavaScript Operator Task</h2>

  <script>

    let a = 15;

    let b = 5;

    // Arithmetic Operators

    console.log("Arithmetic Operations:");

    console.log("a + b = " + (a + b));

    console.log("a - b = " + (a - b));

    console.log("a \* b = " + (a \* b));

console.log("a / b = " + (a / b));

    // Comparison Operators

    console.log("\nComparison Operations:");

    console.log("a == b: " + (a == b));

    console.log("a > b: " + (a > b));

    // Logical Operators

    console.log("\nLogical Operation:");

    console.log("a > 10 && b < 10: " + (a > 10 && b < 10));

    console.log("a > 20 || b < 10: " + (a > 20 || b < 10));

    console.log("!(a < 10): " + !(a < 10));

  </script>

</body>

</html>

Output

**Arithmetic Operations:**

a + b = 20

a - b = 10

a \* b = 75

a / b = 3

**Comparison Operations:**

a == b: false

a > b: true

**Logical Operation:**

a > 10 && b < 10: true

a > 20 || b < 10: true

!(a < 10): true

**Control Flow (If-Else, Switch)**

**Theory Assignment**

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

**Control Flow** refers to the order in which individual statements, instructions, or function calls are executed or evaluated in a program.

In JavaScript, control flow can be altered using conditional statements like if, else if, else, switch, and loops.

#### ****if-else Statement:****

* The if statement checks a condition.
* If the condition is true, the code inside if runs.
* If not, the code inside else runs (if provided).

Example:

let number = 10;

if (number > 0) {

  console.log("Positive number");

} else if (number < 0) {

  console.log("Negative number");

} else {

  console.log("Zero");

}

Output: Positive number

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

#### ****Switch Statement:****

The switch statement evaluates an expression and executes code based on **matching** case **values**.

#### ****Syntax:****

switch (expression) {

  case value1:

    // Code block

    break;

  case value2:

    // Code block

    break;

  default:

    // Default code block

}

* Each case is compared with the expression.
* If a match is found, the corresponding block is executed.
* The break stops the execution from falling through to other cases.

#### ****Use switch instead of if-else when:****

* You have **many specific values to check** (like days, menu options, etc.).
* It makes code more readable and organized compared to multiple else if statements.

**Lab Assignment**

**Task 1: Check if a number is positive, negative, or zero using if-else**

<!DOCTYPE html>

<html>

<head>

  <title>If-Else Example</title>

</head>

<body>

  <h2>Check if a Number is Positive, Negative, or Zero</h2>

  <script>

    let num = parseFloat(prompt("Enter a number:"));

    if (num > 0) {

      console.log("The number is Positive.");

    } else if (num < 0) {

      console.log("The number is Negative.");

    } else {

      console.log("The number is Zero.");

    }

  </script>

</body>

</html>

**Task 2: Display day of the week using switch statement**

<!DOCTYPE html>

<html>

<head>

  <title>Switch Case Day</title>

</head>

<body>

  <h2>Display Day of the Week</h2>

  <script>

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

    switch(dayNumber) {

      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. Enter between 1 and 7.");

    }

  </script>

</body>

</html>

**Loops (For, While, Do-While)**

**Theory Assignment**

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

JavaScript provides three main types of loops to execute a block of code multiple times:

#### ****1.**** for ****loop****

Used when the number of iterations is known.

Syntax:

for (initialization; condition; increment/decrement) {

  // code to run

}

Example:

  for (let i = 1; i <= 5; i++) {

    console.log("Number: " + i);

  }

#### ****2.**** while ****loop****

Used when the number of iterations is unknown and depends on a condition.

Syntax:

while (condition) {

  // code to run

}

Example:

let i = 1;

while (i <= 5) {

  console.log("Number: " + i);

  i++;

}

#### ****3.**** do-while ****loop****

Similar to while, but guarantees at least one execution before checking the condition.

Syntax:

do {

  // code to run

} while (condition);

Example:

let i = 1;

do {

  console.log("Number: " + i);

  i++;

} while (i <= 5);

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

| **Feature** | **while loop** | **do-while loop** |
| --- | --- | --- |
| Condition Check | At the beginning | At the end |
| Executes at least once | ❌ No – only if the condition is true | ✅ Yes – runs once even if condition is false |
| Syntax | while(condition) { ... } | do { ... } while(condition); |

Example:

let x = 15;

// while won't run because x > 10 is false

while (x < 10) {

  console.log("Inside while loop");

}

// do-while will run once

do {

  console.log("Inside do-while loop");

} while (x < 10);

**Lab Assignment**

### ****Task 1: JavaScript program using a for loop to print numbers from 1 to 10****

<!DOCTYPE html>

<html>

<head>

  <title>For Loop Example</title>

</head>

<body>

  <script>

    console.log("Numbers from 1 to 10 using for loop:");

    for (let i = 1; i <= 10; i++) {

      console.log(i);

    }

  </script>

</body>

</html>

Task 2: JavaScript program using while loop to sum all even numbers from 1 to 20

<!DOCTYPE html>

<html>

<head>

  <title>Sum Even Numbers with While</title>

</head>

<body>

  <script>

    let sum = 0;

    let i = 1;

    while (i <= 20) {

      if (i % 2 === 0) {

        sum += i;

      }

      i++;

    }

    console.log("Sum of even numbers from 1 to 20: " + sum);

  </script>

</body>

</html>

**Task 3: Do-while loop to keep asking user input until number > 10**

<!DOCTYPE html>

<html>

<head>

  <title>Do While Loop Input</title>

</head>

<body>

  <script>

    let number;

    do {

      number = parseInt(prompt("Enter a number greater than 10:"));

    } while (number <= 10 || isNaN(number));

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

  </script>

</body>

</html>

**Functions**

**Theory Assignment**

### ****Question 1: What are functions in JavaScript?****

Functions are reusable blocks of code designed to perform a specific task. You can define a function once and call (use) it multiple times.

Syntax for Declaring and Calling a Function

      // Function Declaration

function functionName(parameters) {

  // code to execute

}

// Calling the function

functionName(arguments);

Example:

function sayHello(name) {

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

}

sayHello("Amit");

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

| **Feature** | **Function Declaration** | **Function Expression** |
| --- | --- | --- |
| Syntax | function greet() {} | const greet = function() {} |
| Hoisting | ✅ Hoisted (can be called before declared) | ❌ Not hoisted |
| Name | Always named | Can be anonymous or named |

### ****Question 3: What are parameters and return values in functions?****

* **Parameters** are variables listed in the function definition.
* **Arguments** are the actual values passed to the function.
* **Return values** are results that the function sends back using return.

Example:

function add(a, b) {

  return a + b; // returning the result

}

let result = add(5, 3); // passing arguments

console.log(result); // 8

Lab Assignment

**Task 1: greetUser function with parameter**

<script>

function greetUser(name) {

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

}

greetUser("John");

</script>

Output: Hello, John!

**Task 2: calculateSum function with parameters and return**

<script>

function calculateSum(a, b) {

  return a + b;

}

let total = calculateSum(5, 7);

console.log("Sum:", total);

</script>

Output: Sum:12

**Arrays**

**Theory Assignment**

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

An **array** is a special variable that can hold multiple values at once.

**Syntax:**

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

Arrays use **index numbers** (starting from 0) to access elements.

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

| **Method** | **Description** | **Example** |
| --- | --- | --- |
| push() | Adds item to the **end** of the array | arr.push("mango"); |
| pop() | Removes **last** item from the array | arr.pop(); |
| shift() | Removes **first** item from the array | arr.shift(); |
| unshift() | Adds item to the **start** of the array | arr.unshift("grape"); |

**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.**

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

fruits.push("orange");    // Add to end

fruits.shift();           // Remove first item

console.log(fruits);

Output: ["banana", "cherry", "orange"]

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

let numbers = [2, 4, 6, 8, 10];

let sum = 0;

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

  sum += numbers[i];

}

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

Output: Sum of all numbers: 30

### ****Theory Assignment****

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

An **object** in JavaScript is a data structure used to store **key-value pairs**. Each key (also called a property name) is a **string**, and each value can be any data type—number, string, array, function, another object, etc.

**Example:**

let person = {

  name: "Amit",

  age: 25,

  isStudent: true

};

**Differences between objects and arrays:**

| **Feature** | **Object** | **Array** |
| --- | --- | --- |
| Structure | Key-value pairs | Indexed list |
| Key Type | Strings (property names) | Numeric indexes (0, 1, 2, ...) |
| Best For | Storing structured data | Storing ordered lists |
| Access Pattern | object.key or object["key"] | array[0], array[1], etc. |

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

You can **access** and **update** object properties using two methods:

**Dot Notation**  
Syntax: object.propertyName  
Example:

let person = { name: "Amit" };

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

person.name = "Rahul";    // Update

**Bracket Notation**  
Syntax: object["propertyName"]  
Example:

console.log(person["name"]); // Access

person["name"] = "Raj";      // Update

Use **bracket notation** when the property name is stored in a variable or has special characters.

Lab Assignment

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. J

// Step 1: Create the car object

let car = {

  brand: "Toyota",

  model: "Corolla",

  year: 2020

};

// Step 2: Access and print brand and model

console.log("Brand:", car.brand);    // Using dot notation

console.log("Model:", car["model"]); // Using bracket notation

// Step 3: Update the year property

car.year = 2023;

// Step 4: Add a new property: color

car.color = "Red";

// Output the updated car object

console.log("Updated Car Object:", car);

**JavaScript Events**

### ****Theory Assignment****

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

**JavaScript events** are actions or occurrences that happen in the browser, often as a result of user interactions. Examples include clicking a button, submitting a form, hovering over an element, pressing a key, etc.

An **event listener** is a function in JavaScript that waits for an event to occur on a specific element and then executes code in response.

**Role of event listeners:**

* Detect and respond to events.
* Allow interaction between the user and the webpage.
* Keep JavaScript and HTML separate (for cleaner code).

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

The addEventListener() method is used to attach an event handler to a specific element without overwriting existing event handlers.

**Syntax:**

element.addEventListener(eventType, functionToCall);

* eventType: A string like "click", "mouseover", "keydown", etc.
* functionToCall: A callback function that runs when the event happens.

**Example:**

document.getElementById("myBtn").addEventListener("click", function() {

  alert("Button clicked!");

});

**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>JavaScript Event Listener Example</title>

</head>

<body>

  <h2>Click the Button Below</h2>

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

  <script>

    // Get the button element

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

    // Add a click event listener

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

      alert("Button clicked!");

    });

  </script>

</body>

</html>

**DOM Manipulation**

**Theory Assignment**

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

The **DOM (Document Object Model)** is a **tree-like structure** that represents the entire content of an HTML document. It treats every HTML element as an **object** that can be accessed and manipulated using JavaScript.

**JavaScript interacts with the DOM** to:

* Read and change the content of elements.
* Modify styles, classes, and attributes.
* Create, remove, or move HTML elements.
* Handle events like clicks, typing, form submissions, etc.

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

**1 getElementById("id")**

* Selects a **single element** with the given ID.
* Returns the **first match**.

Example:

let title = document.getElementById("mainTitle");

2 getElementsByClassName("className")

* Returns a **live HTMLCollection** of all elements with the given class.
* You can access elements like an array: elements[0], elements[1], etc.

Example:

let items = document.getElementsByClassName("list-item");

querySelector("selector")

* Returns the **first element** that matches a **CSS selector**.
* Supports classes (.class), IDs (#id), and tag names.

Example:

let paragraph = document.querySelector("p");

let button = document.querySelector("#submitBtn");

**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>

</head>

<body>

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

  <script>

    // Select the paragraph element by ID

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

    // Change the text content

    para.textContent = "JavaScript is fun!";

    // Change the text color to blue

    para.style.color = "blue";

  </script>

</body>

</html>

**JavaScript Timing Events (setTimeout, setInterval)**

### ****Theory Assignment****

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

setTimeout()

* Executes a function **once** after a specified delay (in milliseconds).
* Used to **delay** actions like animations, alerts, or content changes.

**Syntax:**

**setTimeout(function, delayInMilliseconds);**

setInterval()

* Repeatedly calls a function **every** specified number of milliseconds.
* Useful for tasks like clocks, timers, and animations that need updates at intervals.

**Syntax:**

**setInterval(function, intervalInMilliseconds);**

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

setTimeout(function() {

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

}, 2000); // 2000 milliseconds = 2 seconds

### ****Lab Assignment****

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

<!DOCTYPE html>

<html>

<head>

  <title>Change Background</title>

</head>

<body>

  <h2>The background will change in 5 seconds...</h2>

  <script>

    setTimeout(function() {

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

    }, 5000); // 5 seconds

  </script>

</body>

</html>

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

<!DOCTYPE html>

<html>

<head>

  <title>Digital Clock</title>

  <style>

    #clock {

      font-size: 2em;

      font-family: monospace;

    }

  </style>

</head>

<body>

  <h2>Digital Clock</h2>

  <div id="clock">--:--:--</div>

  <script>

    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');

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

    }

    // Initial call

    updateClock();

    // Update every 1 second

    setInterval(updateClock, 1000);

  </script>

</body>

</html>

**JavaScript Error Handling**

**Theory Assignmen**

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

**Error handling** in JavaScript allows you to manage runtime errors gracefully without stopping the program abruptly. JavaScript uses try, catch, and finally blocks to handle exceptions.

* try **block**: Code that might throw an error is placed here.
* catch **block**: Executes if an error occurs in the try block. It can access the error object.
* finally **block** (optional): Executes regardless of whether an error occurred or not.

**Example:**

try {

  let result = someUndefinedFunction(); // This will throw an error

} catch (error) {

  console.log("An error occurred:", error.message);

} finally {

  console.log("This will run no matter what.");

}

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

Error handling is important because it:

* Prevents the entire application from crashing.
* Helps debug and trace issues.
* Provides user-friendly error messages.
* Improves the **stability** and **reliability** of your application.
* Allows fallback mechanisms or cleanup actions.

**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>

<head>

  <title>Division by Zero</title>

</head>

<body>

  <h2>Division Error Handling</h2>

  <p id="output"></p>

  <script>

    try {

      let numerator = 10;

      let denominator = 0;

      if (denominator === 0) {

        throw new Error("Division by zero is not allowed.");

      }

      let result = numerator / denominator;

      document.getElementById("output").textContent = "Result: " + result;

    } catch (error) {

      document.getElementById("output").textContent = "Error: " + error.message;

    } finally {

      console.log("Division attempt complete.");

    }

  </script>

</body>

</html>