**1. JavaScript Implementation**

**A. Variables, Functions, Conditions, and Loops**

This section covers core JavaScript concepts that are fundamental for building logic in web applications.

**1. Variables**

Variables in JavaScript store data that can be used and manipulated later. JavaScript provides three ways to declare variables:

**var (Function-scoped, can be redeclared)**

* var is globally or function-scoped.
* It can be redeclared and updated.
* It does not have block scope, meaning it can be accessed outside of a block {}.

Example:

var name = "John";

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

var name = "Alice";

console.log(name); // Output: Alice

**let (Block-scoped, cannot be redeclared in the same scope)**

* let is block-scoped, meaning it is only accessible within the {} block it was defined in.
* It cannot be redeclared within the same scope but can be updated.

Example:

let age = 25;

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

age = 30;

console.log(age); // Output: 30

// let age = 35; // This would throw an error because `age` is already declared in this scope

**const (Block-scoped, cannot be redeclared or updated)**

* const is also block-scoped but **cannot be reassigned** after declaration.
* Useful for defining constants that should not change (like PI, API keys, etc.).

Example:

const PI = 3.14159;

console.log(PI); // Output: 3.14159

// PI = 3.14; // Error: Assignment to a constant variable

**Best Practices for Using Variables:**  
 Use const when the value should never change.  
 Use let for variables that might change later.  
 Avoid var due to its scoping issues.

**2. Functions**

Functions are blocks of reusable code that perform specific tasks. There are different types of functions in JavaScript.

**Function Declaration**

A named function that can be called anywhere in the script before or after its declaration due to hoisting.

function greet() {

console.log("Hello, World!");

}

greet(); // Output: Hello, World!

**Function Expression**

An anonymous function assigned to a variable. Unlike function declarations, it cannot be called before it is defined.

const add = function(a, b) {

return a + b;

};

console.log(add(5, 3)); // Output: 8

**Arrow Function (ES6 Feature)**

A shorter syntax for writing functions.

const multiply = (a, b) => a \* b;

console.log(multiply(4, 5)); // Output: 20

**3. Conditions (if-else, switch-case)**

Conditional statements allow the program to execute different blocks of code based on conditions.

**if-else Statement**

js

CopyEdit

let temperature = 30;

if (temperature > 35) {

console.log("It's too hot!");

} else if (temperature > 20) {

console.log("The weather is pleasant.");

} else {

console.log("It's quite cold.");

}

**switch-case Statement**

Used when multiple conditions depend on a single variable.

let day = "Monday";

switch (day) {

case "Monday":

console.log("Start of the work week.");

break;

case "Friday":

console.log("Weekend is near!");

break;

default:

console.log("A regular day.");

}

**4. Loops (for, while, do-while)**

Loops help in executing the same block of code multiple times.

**for loop**

Used when the number of iterations is known.

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

console.log("Iteration number: " + i);

}

**while loop**

Used when the number of iterations is unknown but based on a condition.

js

CopyEdit

let i = 1;

while (i <= 5) {

console.log("Iteration number: " + i);

i++;

}

**do-while loop**

Runs at least once, even if the condition is false.

let j = 10;

do {

console.log("Value of j: " + j);

j++;

} while (j < 10);

**Real-World Example: Creating a Dynamic User List**

Using loops and conditions, we can generate a dynamic table of users.

let users = ["Alice", "Bob", "Charlie"];

let table = "<table border='1'><tr><th>Name</th></tr>";

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

table += `<tr><td>${users[i]}</td></tr>`;

}

table += "</table>";

document.write(table);

**Expected Output:**  
A dynamically generated table of users.

| **Name** |
| --- |
| Alice |
| Bob |
| Charlie |

**B. Pop-up Boxes (Alerts, Confirm, and Prompt)**

Pop-up boxes in JavaScript are used to interact with users by providing information, getting user input, or confirming actions. There are three main types of pop-ups:

1. **Alert Box** – Displays a simple message.
2. **Confirm Box** – Asks the user for confirmation (OK/Cancel).
3. **Prompt Box** – Gets input from the user.

**1. Alert Box**

An alert() box is used to display an informational message to the user. It **does not return any value** and is mainly used to notify users about something.

**Example: Displaying an Alert When a Form is Submitted**

function showAlert() {

alert("Form submitted successfully!");

}

**HTML**

<button onclick="showAlert()">Submit</button>

**Expected Output:**

When the user clicks the "Submit" button, a pop-up appears saying:

"Form submitted successfully!"

**Use Cases:**

Notify users about successful form submissions.  
 Show warnings or important messages.  
 Alert users about validation errors.

**2. Confirm Box**

A confirm() box is used to get confirmation from the user. It returns a boolean value:  
 **OK** returns true.  
 **Cancel** returns false.

**Example: Confirm Before Deleting an Item**

function confirmDeletion() {

let userResponse = confirm("Are you sure you want to delete this item?");

if (userResponse) {

alert("Item deleted!");

} else {

alert("Action cancelled.");

}

}

**HTML**

<button onclick="confirmDeletion()">Delete Item</button>

**Expected Output:**

1. A pop-up appears:

"Are you sure you want to delete this item?"

1. If the user clicks "OK" → another pop-up:

"Item deleted!"

1. If the user clicks "Cancel" → another pop-up:

"Action cancelled."

**Use Cases:**

Asking users to confirm before deleting items.  
 Preventing accidental actions.  
 Verifying important decisions (e.g., logout, exit).

**3. Prompt Box**

A prompt() box is used to collect user input. It returns the entered value as a string. If the user clicks "Cancel," it returns null.

**Example: Ask for the User's Name and Display It on the Page**

function askName() {

let userName = prompt("What is your name?");

if (userName) {

document.getElementById("greeting").innerText = "Hello, " + userName + "!";

} else {

alert("You didn't enter a name.");

}

}

**HTML**

<button onclick="askName()">Enter Your Name</button>

<p id="greeting"></p>

**Expected Output:**

1. A pop-up appears:

"What is your name?"

1. If the user types "Aman" and clicks OK, the page updates to show:  
   **"Hello, Aman!"**
2. If the user clicks Cancel, an alert appears:

"You didn't enter a name."

**Use Cases:**

Collecting simple input from users.  
 Customizing messages dynamically.  
 Asking for user preferences.

**Real-World Scenario: Combining Alert, Confirm, and Prompt in a Single Workflow**

**Task:**

A shopping website asks users for their name, then confirms if they want to purchase an item, and finally displays a thank-you message.

function shoppingProcess() {

let userName = prompt("Welcome! What's your name?");

if (!userName) {

alert("Name is required to continue.");

return;

}

let confirmation = confirm(userName + ", do you want to buy this product?");

if (confirmation) {

alert("Thank you, " + userName + "! Your order has been placed.");

} else {

alert("No worries, " + userName + "! Maybe next time.");

}

}

**HTML**

<button onclick="shoppingProcess()">Start Shopping</button>

**Expected Output:**

1. A prompt asks for the user’s name.
2. A confirm box asks if they want to buy the product.
3. If they confirm, an alert thanks them. If they cancel, an alert says "Maybe next time."

**C. Event Handling & DOM Manipulation**

JavaScript event handling and DOM (Document Object Model) manipulation allow web pages to respond dynamically to user interactions. This section focuses on how to:

Handle user interactions like clicks, hovers, and input changes.  
 Modify webpage elements dynamically (text, styles, images, lists).  
 Create interactive UI components.

**1. Event Handling in JavaScript**

An **event** occurs when the user interacts with a webpage (e.g., clicking a button, moving the mouse, typing in a text field). JavaScript can **listen** for these events and execute code when they happen.

**Adding Event Listeners**

There are multiple ways to attach event listeners to HTML elements:

**Method 1: Using onclick in HTML**

html

<button onclick="changeText()">Click Me</button>

<p id="demo">Original Text</p>

<script>

function changeText() {

document.getElementById("demo").innerText = "Text changed!";

}

</script>

**Pros:** Simple and easy to implement.  
 **Cons:** Mixing JavaScript with HTML is not a good practice.

**Method 2: Using addEventListener() (Best Practice)**

html

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

<p id="demo">Original Text</p>

<script>

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

document.getElementById("demo").innerText = "Text changed!";

});

</script>

**Pros:** Keeps JavaScript separate from HTML, improving maintainability.  
 **Allows multiple event listeners** on the same element.

**2. Changing Text, Style, and Images**

**Changing Text Dynamically**

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

document.getElementById("text").innerText = "Hello, JavaScript!";

});

<button id="btn">Change Text</button>

<p id="text">Click the button above</p>

**Expected Output:**  
When clicked, the paragraph text changes to "Hello, JavaScript!".

**Changing Styles (CSS) Dynamically**

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

document.getElementById("box").style.backgroundColor = "blue";

document.getElementById("box").style.color = "white";

});

html

<button id="styleBtn">Change Style</button>

<p id="box" style="width: 100px; padding: 10px; background: lightgray;">Styled Box</p>

**Expected Output:**  
The box changes color when the button is clicked.

**Changing Images Dynamically**

function changeImage() {

document.getElementById("image").src = "new-image.jpg";

}

html

<img id="image" src="old-image.jpg" width="200">

<button onclick="changeImage()">Change Image</button>

**Expected Output:**  
The image changes when the button is clicked.

**3. Implementing Hover Effects Using onmouseover and onmouseout**

**Example: Change Background Color on Hover**

document.getElementById("hoverBox").addEventListener("mouseover", function() {

this.style.backgroundColor = "yellow";

});

document.getElementById("hoverBox").addEventListener("mouseout", function() {

this.style.backgroundColor = "lightgray";

});

html

<div id="hoverBox" style="width: 100px; height: 100px; background: lightgray; text-align: center; line-height: 100px;">

Hover Me

</div>

When hovered, the background changes to yellow.  
 When the mouse leaves, it changes back to gray.

**4. Creating a Dynamic List (Add/Remove Items)**

**Example: To-Do List with Add & Remove Functionality**

function addItem() {

let itemText = document.getElementById("itemInput").value;

if (itemText.trim() === "") return; // Prevent empty items

let listItem = document.createElement("li");

listItem.innerText = itemText;

let deleteBtn = document.createElement("button");

deleteBtn.innerText = "❌";

deleteBtn.onclick = function() {

this.parentElement.remove();

};

listItem.appendChild(deleteBtn);

document.getElementById("itemList").appendChild(listItem);

document.getElementById("itemInput").value = ""; // Clear input field

}

<input type="text" id="itemInput" placeholder="Enter item">

<button onclick="addItem()">Add Item</button>

<ul id="itemList"></ul>

Items can be added dynamically.  
 Clicking ❌ removes the item from the list.

**5. Real-World Example: Form Auto-Updating on Input**

**Task: Show live preview of user input**

document.getElementById("nameInput").addEventListener("input", function() {

document.getElementById("preview").innerText = "Hello, " + this.value + "!";

});

html

<input type="text" id="nameInput" placeholder="Type your name">

<p id="preview">Hello!</p>

As the user types, the text updates in real-time.

**D. Forms & Validation (Using Regular Expressions)**

Form validation ensures that users enter correct and secure information before submitting a form. JavaScript, along with **regular expressions (RegEx)**, is used to validate fields like name, email, phone number, and password.

In this section, we will learn how to:  
 Validate form fields using JavaScript and RegEx.  
 Show real-time validation messages.  
 Ensure strong password requirements.

**1. Basic Form Structure**

Let's start with a simple **HTML form**:

html

<form id="registrationForm">

<label>Name:</label>

<input type="text" id="name" required>

<small id="nameError" style="color: red;"></small>

<label>Email:</label>

<input type="email" id="email" required>

<small id="emailError" style="color: red;"></small>

<label>Phone Number:</label>

<input type="text" id="phone" required>

<small id="phoneError" style="color: red;"></small>

<label>Password:</label>

<input type="password" id="password" required>

<small id="passwordError" style="color: red;"></small>

<button type="submit">Register</button>

</form>

**2. JavaScript Validation Using Regular Expressions**

**Validation Logic**

* **Name**: Only alphabets allowed, minimum 3 characters.
* **Email**: Must follow proper format (e.g., user@example.com).
* **Phone Number**: Must be exactly **10 digits** and contain only numbers.
* **Password**: Must contain uppercase, lowercase, number, and special character.

document.getElementById("registrationForm").addEventListener("submit", function(event) {

event.preventDefault(); // Prevent form submission if validation fails

let isValid = true; // Track validation status

// Name Validation (Only alphabets, min 3 characters)

let name = document.getElementById("name").value;

let namePattern = /^[A-Za-z ]{3,}$/;

if (!namePattern.test(name)) {

document.getElementById("nameError").innerText = "Enter a valid name (Only letters, min 3 characters)";

isValid = false;

} else {

document.getElementById("nameError").innerText = "";

}

// Email Validation

let email = document.getElementById("email").value;

let emailPattern = /^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/;

if (!emailPattern.test(email)) {

document.getElementById("emailError").innerText = "Enter a valid email (e.g., user@example.com)";

isValid = false;

} else {

document.getElementById("emailError").innerText = "";

}

// Phone Number Validation (10 digits only)

let phone = document.getElementById("phone").value;

let phonePattern = /^[0-9]{10}$/;

if (!phonePattern.test(phone)) {

document.getElementById("phoneError").innerText = "Enter a valid 10-digit phone number";

isValid = false;

} else {

document.getElementById("phoneError").innerText = "";

}

// Password Validation (Uppercase, lowercase, number, special character)

let password = document.getElementById("password").value;

let passwordPattern = /^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\d)(?=.\*[@$!%\*?&])[A-Za-z\d@$!%\*?&]{8,}$/;

if (!passwordPattern.test(password)) {

document.getElementById("passwordError").innerText = "Password must be at least 8 characters long, contain uppercase, lowercase, number, and special character";

isValid = false;

} else {

document.getElementById("passwordError").innerText = "";

}

// If all validations pass, submit the form

if (isValid) {

alert("Form submitted successfully!");

}

});

**3. Real-Time Validation (Instant Feedback)**

Instead of waiting until the user submits the form, we can show live validation messages **as they type**.

**Example: Live Email Validation**

document.getElementById("email").addEventListener("input", function() {

let emailPattern = /^[a-zA-Z0

**E. JavaScript Standard Objects & AJAX**

JavaScript provides built-in objects that help in performing various operations like handling dates, mathematical calculations, and making asynchronous requests using AJAX. In this section, we will cover:

**Date Object** – Display current date and time.  
 **Math Object** – Perform mathematical operations (e.g., generating random numbers).  
 **AJAX (Asynchronous JavaScript and XML)** – Fetch data from an API (e.g., OpenWeather API).

**1. Working with the JavaScript Date Object**

The **Date Object** allows you to work with date and time. It can be used to display the current date, time, or manipulate date values.

**Example: Display the Current Date and Time**

let currentDate = new Date();

console.log(currentDate); // Outputs full date and time

**Displaying the Date on a Webpage**

html

<p id="dateDisplay"></p>

<script>

document.getElementById("dateDisplay").innerText = new Date().toLocaleString();

</script>

toLocaleString() formats the date based on the user's local settings.

**2. JavaScript Math Object**

The **Math Object** provides mathematical functions like generating random numbers, rounding values, and finding square roots.

**Generating a Random Number (Between 1 and 100)**

let randomNumber = Math.floor(Math.random() \* 100) + 1;

console.log("Random Number: " + randomNumber);

Math.random() generates a number between 0 and 1.  
 Math.floor() rounds it down to a whole number.

**Finding the Square Root of a Number**

let num = 25;

console.log("Square Root: " + Math.sqrt(num)); // Output: 5

**3. Fetching Data Using AJAX (Asynchronous JavaScript and XML)**

AJAX allows us to **fetch data from a server without refreshing the page**. It is commonly used to load API data dynamically.

**Fetching Data from an API Using fetch()**

Let's retrieve live **weather data** using the OpenWeather API.

html

<input type="text" id="cityInput" placeholder="Enter city">

<button onclick="getWeather()">Get Weather</button>

<p id="weatherOutput"></p>

<script>

function getWeather() {

let city = document.getElementById("cityInput").value;

let apiKey = "YOUR\_API\_KEY"; // Replace with your OpenWeather API Key

let url = `https://api.openweathermap.org/data/2.5/weather?q=${city}&appid=${apiKey}&units=metric`;

fetch(url)

.then(response => response.json())

.then(data => {

let temp = data.main.temp;

document.getElementById("weatherOutput").innerText = `Temperature: ${temp}°C`;

})

.catch(error => console.error("Error fetching weather data:", error));

}

</script>

Uses fetch() to get data from OpenWeather API.  
 Displays the temperature dynamically on the page.

**4. Summary of Key Learnings**

By completing this section, you will:  
 Work with **Date Object** to display current date and time.  
Use **Math Object** to generate random numbers and perform calculations.  
 Fetch **real-time data from an API** using AJAX and display it dynamically.