**1. JavaScript Basics & Setup**

**Objective:**

Set up the script and confirm it runs properly.

**HTML:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<title>Community Portal</title>

</head>

<body>

<h1>Welcome to the Community Portal</h1>

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

</body>

</html>

**main.js:**

console.log("Welcome to the Community Portal");

window.onload = function () {

alert("Page fully loaded!");

};

**2. Syntax, Data Types, and Operators**

**main.js:**

const eventName = "Community Yoga";

const eventDate = "2025-07-10";

let availableSeats = 25;

console.log(`Event: ${eventName} on ${eventDate}. Seats: ${availableSeats}`);

function register() {

if (availableSeats > 0) {

availableSeats--;

console.log(`Registered! Seats left: ${availableSeats}`);

}

}

**3. Conditionals, Loops, and Error Handling**

const events = [

{ name: "BBQ Bash", date: "2025-07-10", seats: 10 },

{ name: "Old Fest", date: "2024-12-01", seats: 0 },

];

events.forEach((event) => {

const eventDate = new Date(event.date);

const today = new Date();

if (eventDate > today && event.seats > 0) {

console.log(`Upcoming: ${event.name}`);

} else {

console.log(`Skipping: ${event.name}`);

}

});

function safeRegister(event) {

try {

if (event.seats <= 0) throw "Event full!";

event.seats--;

console.log("Registered for", event.name);

} catch (e) {

console.error("Registration error:", e);

}

}

**4. Functions, Scope, Closures, Higher-Order Functions**

function addEvent(name, category) {

return { name, category };

}

function registerUser(event) {

console.log(`User registered for ${event.name}`);

}

function filterEventsByCategory(events, category) {

return events.filter(e => e.category === category);

}

// Closure to track total registrations

function createRegistrationTracker() {

let total = 0;

return function () {

total++;

return total;

};

}

const trackYoga = createRegistrationTracker();

console.log(trackYoga()); // 1

**5. Objects and Prototypes**

function Event(name, date, seats) {

this.name = name;

this.date = date;

this.seats = seats;

}

Event.prototype.checkAvailability = function () {

return this.seats > 0;

};

const yoga = new Event("Yoga", "2025-07-10", 5);

console.log(Object.entries(yoga));

console.log(yoga.checkAvailability());

**6. Arrays and Methods**

let allEvents = [];

allEvents.push({ name: "Music Night", category: "music" });

allEvents.push({ name: "Art Class", category: "art" });

const musicEvents = allEvents.filter(e => e.category === "music");

const cards = allEvents.map(e => `Event: ${e.name}`);

console.log(cards);

**7. DOM Manipulation**

const container = document.querySelector("#eventContainer");

function renderEventCard(event) {

const card = document.createElement("div");

card.className = "event-card";

card.innerHTML = `<h3>${event.name}</h3><button onclick="register('${event.name}')">Register</button>`;

container.appendChild(card);

}

**8. Event Handling**

document.querySelector("#categoryFilter").onchange = function (e) {

const selected = e.target.value;

// Filter logic...

};

document.querySelector("#searchBox").addEventListener("keydown", function (e) {

console.log("Typed:", e.key);

});

function register(eventName) {

alert(`You registered for ${eventName}`);

}

**9. Async JS, Promises, Async/Await**

// Using Promises

fetch("events.json")

.then(res => res.json())

.then(data => console.log(data))

.catch(err => console.error(err));

// Using async/await

async function loadEvents() {

document.getElementById("spinner").style.display = "block";

try {

const res = await fetch("events.json");

const data = await res.json();

console.log(data);

} finally {

document.getElementById("spinner").style.display = "none";

}

}

**10. Modern JavaScript Features**

function greet(name = "Guest") {

console.log(`Hello, ${name}`);

}

const event = { name: "Workshop", date: "2025-07-01" };

const { name, date } = event;

const clone = [...allEvents]; // Spread operator

**11. Working with Forms**

<form id="regForm">

<input name="name" required />

<input name="email" type="email" required />

<select name="event">

<option value="music">Music Night</option>

</select>

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

</form>

<div id="errorMsg"></div>

<script>

document.getElementById("regForm").addEventListener("submit", function (e) {

e.preventDefault();

const form = e.target;

const name = form.elements["name"].value;

const email = form.elements["email"].value;

if (!name || !email.includes("@")) {

document.getElementById("errorMsg").textContent = "Invalid input!";

return;

}

alert("Submitted!");

});

</script>

**12. AJAX & Fetch API**

function submitForm(data) {

fetch("https://mockapi.com/register", {

method: "POST",

headers: { "Content-Type": "application/json" },

body: JSON.stringify(data),

})

.then(res => res.json())

.then(() => alert("Success!"))

.catch(() => alert("Error!"));

setTimeout(() => {

console.log("Simulating delay...");

}, 1500);

}

**13. Debugging and Testing**

* Use **Console tab** for console.log() and errors
* Use **Sources → Breakpoints** to step through code
* Use **Network tab** to inspect fetch requests and responses
* Example:

console.log("Submitting form");

debugger;

fetch("/api").then(r => r.json()).then(data => console.log(data));

**14. jQuery and JS Frameworks**

<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>

<script>

$("#registerBtn").click(() => {

alert("Registered!");

});

$(".event-card").fadeIn();

$(".event-card").fadeOut();

</script>

**Framework Note:**

Frameworks like **React** or **Vue** improve state management, offer component reuse, and make UI updates more efficient and maintainable in large apps.