JavaScript Exercises - Local Community Event Portal

# 1. JavaScript Basics & Setup

Goal: Initialize the JS environment.  
  
HTML:  
<script src="main.js"></script>  
  
JavaScript:  
console.log("Welcome to the Community Portal");  
  
window.onload = () => {  
 alert("Page fully loaded!");  
};

# 2. Syntax, Data Types, and Operators

Goal: Use proper data types and operations.  
  
JavaScript:  
const eventName = "Music Fest";  
const eventDate = "2025-06-15";  
let availableSeats = 50;  
  
const eventInfo = `${eventName} on ${eventDate} has ${availableSeats} seats available.`;  
console.log(eventInfo);  
  
availableSeats--;  
console.log(`Seats left: ${availableSeats}`);

# 3. Conditionals, Loops, and Error Handling

Goal: Apply conditions and handle invalid data.  
  
JavaScript:  
const events = [  
 { name: "Yoga Class", date: "2025-05-25", seats: 0 },  
 { name: "Music Fest", date: "2025-06-15", seats: 20 }  
];  
  
events.forEach(event => {  
 const isUpcoming = new Date(event.date) > new Date();  
 const hasSeats = event.seats > 0;  
 if (isUpcoming && hasSeats) {  
 console.log(`${event.name} is available.`);  
 }  
});  
  
function register(event) {  
 try {  
 if (event.seats <= 0) throw "No seats available!";  
 event.seats--;  
 } catch (err) {  
 console.error(err);  
 }  
}

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

Goal: Encapsulate logic and use closures.  
  
JavaScript:  
function addEvent(events, newEvent) {  
 events.push(newEvent);  
}  
  
function registerUser(event) {  
 if (event.seats > 0) event.seats--;  
}  
  
function filterEventsByCategory(events, category) {  
 return events.filter(e => e.category === category);  
}  
  
function createCategoryTracker() {  
 let count = 0;  
 return function register() {  
 count++;  
 console.log(`Registrations for category: ${count}`);  
 };  
}  
  
function filterEvents(events, callback) {  
 return events.filter(callback);  
}

# 5. Objects and Prototypes

Goal: Model real-world entities using objects.  
  
JavaScript:  
function Event(name, date, seats) {  
 this.name = name;  
 this.date = date;  
 this.seats = seats;  
}  
  
Event.prototype.checkAvailability = function () {  
 return this.seats > 0 && new Date(this.date) > new Date();  
};  
  
const musicEvent = new Event("Concert", "2025-07-01", 30);  
console.log(Object.entries(musicEvent));

# 6. Arrays and Methods

Goal: Use array methods for CRUD operations.  
  
JavaScript:  
const events = [];  
events.push({ name: "Workshop on Baking", category: "workshop" });  
events.push({ name: "Live Music", category: "music" });  
  
const musicEvents = events.filter(e => e.category === "music");  
const displayCards = events.map(e => `Event: ${e.name}`);  
console.log(displayCards);

# 7. DOM Manipulation

Goal: Render events using JS.  
  
JavaScript:  
const container = document.querySelector("#eventContainer");  
  
function renderEvents(events) {  
 container.innerHTML = "";  
 events.forEach(event => {  
 const card = document.createElement("div");  
 card.innerText = `${event.name} - ${event.date}`;  
 container.appendChild(card);  
 });  
}

# 8. Event Handling

Goal: Respond to user actions.  
  
JavaScript:  
document.querySelectorAll(".registerBtn").forEach(btn => {  
 btn.onclick = () => alert("Registered!");  
});  
  
document.querySelector("#categoryFilter").onchange = (e) => {  
 filterEventsByCategory(events, e.target.value);  
};  
  
document.querySelector("#searchInput").onkeydown = (e) => {  
 if (e.key === "Enter") console.log("Searching:", e.target.value);  
};

# 9. Async JS, Promises, Async/Await

Goal: Use asynchronous logic for remote operations.  
  
JavaScript:  
// Using fetch and then  
fetch("https://mockapi.io/events")  
 .then(res => res.json())  
 .then(data => console.log(data))  
 .catch(err => console.error("Error:", err));  
  
// Using async/await  
async function fetchEvents() {  
 document.querySelector("#loading").style.display = "block";  
 try {  
 const res = await fetch("https://mockapi.io/events");  
 const data = await res.json();  
 console.log(data);  
 } finally {  
 document.querySelector("#loading").style.display = "none";  
 }  
}

# 10. Modern JavaScript Features

Goal: Use ES6+ features.  
  
JavaScript:  
const displayEvent = ({ name, date }) => `${name} is on ${date}`;  
  
function greetUser(name = "Guest") {  
 console.log(`Hello, ${name}`);  
}  
  
const [firstEvent, ...restEvents] = events;  
const clonedEvents = [...events];

# 11. Working with Forms

Goal: Connect form inputs to JavaScript.  
  
JavaScript:  
document.querySelector("#regForm").onsubmit = (e) => {  
 e.preventDefault();  
 const form = e.target;  
 const name = form.elements["name"].value;  
 const email = form.elements["email"].value;  
 const event = form.elements["event"].value;  
  
 if (!name || !email) {  
 document.querySelector("#error").innerText = "All fields required!";  
 } else {  
 console.log("Form submitted:", { name, email, event });  
 }  
};

# 12. AJAX & Fetch API

Goal: Simulate backend communication.  
  
JavaScript:  
async function submitRegistration(userData) {  
 try {  
 const res = await fetch("https://mockapi.io/register", {  
 method: "POST",  
 headers: { "Content-Type": "application/json" },  
 body: JSON.stringify(userData)  
 });  
 const result = await res.json();  
 setTimeout(() => alert("Registration successful!"), 1000);  
 } catch (err) {  
 alert("Submission failed.");  
 }  
}

# 13. Debugging and Testing

Goal: Use browser tools to inspect and fix issues.  
  
Tips:  
- Use Chrome DevTools Console and Network tab.  
- Add breakpoints to inspect variables.  
- Log form submission steps and check fetch request payload.

# 14. jQuery and JS Frameworks

Goal: Understand and use jQuery.  
  
JavaScript (jQuery):  
$("#registerBtn").click(() => {  
 alert("jQuery Registration Clicked");  
});  
  
$(".event-card").fadeIn();  
$(".event-card").fadeOut();  
  
Benefit of frameworks (React/Vue):  
- Component-based structure  
- Efficient DOM updates  
- State management and maintainability