Complete JS Course Syllabus

🔥 JavaScript – Learn Everything Series

📘 Phase 2: DOM, Events, Forms & Storage

🌐 Chapter 9: The DOM (Document Object Model)

🧠 What is the DOM?

The **DOM (Document Object Model)** is a programming interface that represents an HTML document as a **tree structure**.
JavaScript uses the DOM to **access, modify, add, and remove elements dynamically**.

The DOM converts HTML into **objects that JavaScript can understand and control**.

Think of the DOM as the **brain of the webpage** — JavaScript talks to the DOM, and the DOM updates the UI.

🌳 DOM Tree Structure

Every HTML document is converted into a **tree of nodes**:

- **Node** → Base unit of DOM
- **Element Node** → HTML tags (`div`, `p`, `button`)
- **Text Node** → Text inside elements
- **Comment Node** → HTML comments

Example:

```
document
 └── html
     ├── head
     └── body
         └── h1
             └── "Hello World"
```

🔍 Selecting DOM Elements

By ID

```
document.getElementById("title")
```

By Class Name

```
document.getElementsByClassName("item")
```

➡ Returns **HTMLCollection (live)**

By CSS Selector

```
document.querySelector(".box")
document.querySelectorAll(".box")
```

➡ `querySelectorAll` returns **NodeList (static)**

📝 Text & Content Properties

innerText

- Returns **only visible text**
- Affected by CSS

textContent

- Returns **all text**, including hidden
- Faster and more reliable

innerHTML

- Returns **HTML + text**
- Can inject tags

⚠️ Avoid `innerHTML` with user input → security risk

🧬 Attribute Manipulation

```
element.getAttribute("src")
element.setAttribute("id", "main")
element.removeAttribute("disabled")
```

Used to control **element behavior**, not styling.

🔧 Dynamic DOM Manipulation

```
document.createElement("li")
parent.appendChild(child)
parent.prepend(child)
parent.removeChild(child)
```

Used in:

- Todo apps
- Dynamic lists
- Interactive UI updates

🎨 Styling Elements

Inline Styles

```
element.style.backgroundColor = "red"
```

classList (Best Practice)

```
element.classList.add("active")
element.classList.remove("active")
element.classList.toggle("active")
```

⚠️ Common Confusions

innerText vs textContent vs innerHTML

Property	Hidden Text	HTML

innerText	❌	❌
textContent	✅	❌
innerHTML	✅	✅

HTMLCollection vs NodeList

- **HTMLCollection** → Live, auto-updates
- **NodeList** → Static snapshot

🧠 Mindset

DOM is **expensive to manipulate**.
Make **minimal changes**, batch updates, and avoid unnecessary reflows.

🧪 Practice Zone – 🌐 DOM (Document Object Model)

1. Create a `<ul>` dynamically and populate it using an array of strings.
2. Add three `<p>` elements inside a `<div>` using JavaScript.
3. Remove the last child element from a list using DOM methods.
4. Replace the text of an existing element using `textContent`.
5. Add a CSS class to an element using `classList.add()`.
6. Remove an attribute from an image element using `removeAttribute()`.
7. Compare outputs of `innerText`, `textContent`, and `innerHTML` on the same element.
8. Select elements using:
    - `getElementById`

    - `querySelectorAll`

9. Create an element, set its attribute, and append it to the DOM.
10. Explain the difference between **HTMLCollection** and **NodeList** using code.

🖱️ Chapter 10: Events & Event Handling

🧠 What are Events?

Events are **actions triggered by the user or browser**, such as clicking, typing, or submitting a form.

JavaScript listens to events and executes code in response.

🔗 Event Binding

```
element.addEventListener("click", handler)
element.removeEventListener("click", handler)
```

🖱️ Common Events

- `click`
- `input`
- `change`
- `submit`
- `mouseover`
- `keyup`

🎯 Event Object

Every event provides an object with useful information:

```
event.target
event.type
event.preventDefault()
```

🌊 Event Bubbling & Capturing

- **Bubbling (default)** → child → parent
- **Capturing** → parent → child

```
addEventListener("click", fn, true)
```

🧠 Event Delegation

Attach **one listener to a parent**, detect child using `event.target`.

✔ Better performance
✔ Works with dynamic elements

⚠️ Common Confusions

- `event.target` → element clicked
- `event.currentTarget` → element with listener
- Bubbling happens by default
- Capturing is rarely used

🧠 Mindset

Don’t attach listeners to every element.
Delegate smartly.

🧪 Practice Zone – 🖱️ Events & Event Handling

1. Attach a click event to a button that changes paragraph text.
2. Log `event.type` and `event.target` on a button click.
3. Change background color of a box on `mouseover` and reset on `mouseout`.
4. Detect key presses inside an input using keyboard events.
5. Demonstrate event bubbling using nested elements.
6. Demonstrate event capturing using `addEventListener`.
7. Stop event propagation between parent and child elements.
8. Attach a single event listener to a parent and handle child clicks (event delegation).
9. Remove an event listener after it executes once.
10. Compare `event.target` and `event.currentTarget` with an example.

📝 Chapter 11: Forms & Form Validation

🧠 Forms in JavaScript

Forms collect user data.
JavaScript validates and sanitizes this data before submission.

📥 Reading Input Values

```
input.value
textarea.value
select.value
```

🚫 Prevent Default Submission

```
form.addEventListener("submit", e => {
  e.preventDefault()
})
```

✅ Validation Techniques

HTML Validation

- `required`
- `pattern`
- `minlength`

JavaScript Validation

- Regex checks
- Custom error messages

🚨 Error Handling

- Show error text dynamically
- Highlight invalid fields
- Remove error when fixed

⚠️ Common Confusions

- `value` → form elements
- `textContent` → normal elements
- Submit ≠ button click

🧠 Mindset

Forms are **communication bridges**.
Validate deeply, never trust user input.

🧪 Practice Zone – 📝 Forms & Form Validation

1. Prevent default form submission and log input values.
2. Display input value inside a paragraph after form submission.
3. Show an error message if input field is empty.
4. Validate an email field using JavaScript regex.
5. Highlight invalid inputs using CSS classes.
6. Remove error message when input becomes valid.
7. Validate password length before submission.
8. Use the `pattern` attribute for basic input validation.
9. Compare form submission vs button click behavior.
10. Clear form fields after successful submission.

🕐 Chapter 12: Timers & Intervals

⏳ setTimeout

Runs code **once after delay**

```
setTimeout(fn, 3000)
clearTimeout(id)
```

🔁 setInterval

Runs code **repeatedly**

```
setInterval(fn, 1000)
clearInterval(id)
```

⚠️ Common Confusion

- `setInterval` may drift
- Recursive `setTimeout` is more accurate

🧠 Mindset

Time controls **UX rhythm** — loaders, alerts, animations.

🧪 Practice Zone – 🕐 Timers & Intervals

1. Display a message after 3 seconds using `setTimeout`.
2. Create a counter that increments every second using `setInterval`.
3. Stop an interval using a button click.
4. Automatically hide a notification after a delay.
5. Create a countdown timer from 10 to 0.
6. Replace `setInterval` with recursive `setTimeout`.
7. Start and stop a timer using two separate buttons.
8. Delay DOM manipulation using `setTimeout`.
9. Show current time updating every second.

10. Explain difference between `setTimeout` and `setInterval` with example.

📦 Chapter 13: Storage (LocalStorage, SessionStorage & Cookies)

🗄️ localStorage

- Persistent storage
- Data survives reload

```
localStorage.setItem("key","value")
localStorage.getItem("key")
localStorage.removeItem("key")
localStorage.clear()
```

🕒 sessionStorage

- Tab-based storage
- Cleared on tab close

📦 JSON Handling

Storage supports **only strings**:

```
JSON.stringify(obj)
JSON.parse(str)
```

🍪 Cookies (Basics)

- Stored as key=value
- Sent with every request
- Needs manual expiration

⚠️ Common Confusions

- localStorage is NOT secure
- Cookies are small & limited

🧠 Mindset

Use:

- localStorage → UI state
- cookies → authentication

🧪 Practice Zone – 📦 localStorage, sessionStorage & Cookies

1. Store a string value in `localStorage` and retrieve it.
2. Remove a specific item from `localStorage`.
3. Clear all data from `localStorage`.
4. Store an object using `JSON.stringify()` and retrieve it.
5. Save user name in `sessionStorage`.
6. Read stored data on page load.

7. Implement a “Remember Me” feature using storage.
8. Save theme preference and apply it after reload.
9. Compare `localStorage` and `sessionStorage` behavior.
10. Create and read a basic cookie using JavaScript.