**JAVASCRIPT IMPORTANT CONCEPTS**

***🌳 What is DOM?***

DOM means **Document Object Model**.

Think:

* You write an **HTML page**.
* Browser converts that HTML into a **tree-like structure**.
* That tree is called the **DOM**.

👉 Example HTML:

<!DOCTYPE html>

<html>

<head>

<title>My Page</title>

</head>

<body>

<h1>Hello World</h1>

<p>This is my first paragraph.</p>

</body>

</html>

👉 Browser turns this into a tree:

Document

└── html

├── head

│ └── title → "My Page"

└── body

├── h1 → "Hello World"

└── p → "This is my first paragraph."

Each box is called a **node** (or element).  
JavaScript can **access this tree** and change it.

**🛠 Why DOM is important?**

With DOM, JavaScript can:

1. **Read** the page → like checking what text is inside <h1>.
2. **Change** the page → like updating text, color, or images.
3. **Add / Remove** things → like creating a new button or deleting a paragraph.
4. **React to events** → like when you click a button, something happens.

Without DOM, your page would be just static text.

**🎯 How to Access DOM in JavaScript?**

We use the document object. Think of document as the **entry point** to the page.

**Common methods:**

1. **By ID**

<h1 id="title">Hello</h1>

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

console.log(heading.innerText); // "Hello"

1. **By Class**

<p class="note">First</p>

<p class="note">Second</p>

let notes = document.getElementsByClassName("note");

console.log(notes[0].innerText); // "First"

1. **By Tag**

<p>Apple</p>

<p>Mango</p>

let paras = document.getElementsByTagName("p");

console.log(paras.length); // 2

1. **Query Selector (CSS style)**

<div class="box">One</div>

<div class="box">Two</div>

let firstBox = document.querySelector(".box"); // First one

let allBoxes = document.querySelectorAll(".box"); // All of them

**✍️ Changing Content**

<h2 id="head">Old Text</h2>

let h = document.getElementById("head");

// Change text

h.innerText = "New Text";

// Add HTML inside

h.innerHTML = "<em>New Text</em>";

👉 Difference:

* innerText = only text.
* innerHTML = text + HTML tags.

**🎨 Changing Style**

<p id="para">Hello</p>

let p = document.getElementById("para");

p.style.color = "red";

p.style.fontSize = "20px";

👉 Better way is to add/remove classes:

.big { font-size: 30px; color: blue; }

p.classList.add("big");

**🧱 Creating and Adding New Elements**

<ul id="list">

<li>Apple</li>

</ul>

let ul = document.getElementById("list");

// Create a new <li>

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

li.innerText = "Mango";

// Add it inside <ul>

ul.appendChild(li);

Now the list becomes:

Apple

Mango

**❌ Removing Elements**

li.remove(); // deletes the li we created

**⚡ Events (interactions)**

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

<p id="msg"></p>

let btn = document.getElementById("btn");

let msg = document.getElementById("msg");

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

msg.innerText = "Button was clicked!";

});

👉 When you click the button, JavaScript **listens** and reacts.

**🧠 Summary (Easy to remember)**

* **DOM = Tree of your webpage.**
* Use document to reach inside.
* getElementById / querySelector = to **find**.
* innerText / innerHTML = to **change content**.
* .style / .classList = to **change design**.
* createElement / appendChild = to **add**.
* .remove() = to **delete**.
* addEventListener = to **make interactive**.

*Great question, Dinesh! DOM manipulation is the bread and butter of dynamic web development, and you're already well-versed in it. Here's a solid rundown of* ***common JavaScript DOM manipulation techniques*** *that every developer should have in their toolkit*:

**🧱 Accessing Elements**

* document.getElementById("id") – Selects an element by its ID.
* document.querySelector("selector") – Selects the first matching element.
* document.querySelectorAll("selector") – Selects all matching elements as a NodeList.
* document.getElementsByClassName("class") – Returns a live HTMLCollection.
* document.getElementsByTagName("tag") – Returns elements by tag name.

**✍️ Changing Content**

* element.textContent – Gets or sets the text inside an element.
* element.innerHTML – Gets or sets HTML content.
* element.innerText – Similar to textContent, but respects styling and hidden elements.

**🎨 Styling Elements**

* element.style.property = "value" – Directly sets inline styles.
* element.classList.add("className") – Adds a class.
* element.classList.remove("className") – Removes a class.
* element.classList.toggle("className") – Toggles a class on/off.

**🧬 Creating & Inserting Elements**

* document.createElement("tag") – Creates a new element.
* parent.appendChild(child) – Adds a child to the end.
* parent.insertBefore(newNode, referenceNode) – Inserts before a specific node.
* element.insertAdjacentHTML("position", html) – Injects HTML at a specific position (beforebegin, afterbegin, beforeend, afterend).

**🧹 Removing Elements**

* element.remove() – Removes the element from the DOM.
* parent.removeChild(child) – Removes a child from its parent.

**🎯 Event Handling**

* element.addEventListener("event", callback) – Attaches an event listener.
* element.removeEventListener("event", callback) – Removes an event listener.

**🔍 Attribute Manipulation**

* element.setAttribute("name", "value") – Sets an attribute.
* element.getAttribute("name") – Gets an attribute.
* element.removeAttribute("name") – Removes an attribute.

**🧪 Bonus: Useful DOM Utilities**

* element.closest("selector") – Finds the nearest ancestor matching the selector.
* element.matches("selector") – Checks if the element matches a selector.
* element.children / element.parentElement / element.nextElementSibling – DOM traversal.

Want to dive deeper into any of these with real-world examples or code snippets? Or maybe explore how these play out in React’s virtual DOM vs vanilla JS?

***📘 JavaScript Fetch API — Complete Notes***

**1. What is Fetch API?**

* The **Fetch API** provides a modern way to make HTTP requests in JavaScript.
* It replaces older methods like **XMLHttpRequest**.
* Returns a **Promise** that resolves to a Response object.

**2. Basic Syntax**

fetch(url, options)

.then(response => response.json()) // or .text(), .blob(), etc.

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

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

* url → the resource you want to fetch.
* options → optional object (method, headers, body, etc.).
* Returns a **Promise**.

**3. Common HTTP Methods in Fetch**

1. **GET** (default) → Retrieve data

fetch("https://api.example.com/products");

1. **POST** → Send new data

fetch("https://api.example.com/products", {

method: "POST",

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

body: JSON.stringify({ name: "Laptop", price: 50000 })

});

1. **PUT** → Replace existing data

fetch("https://api.example.com/products/1", {

method: "PUT",

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

body: JSON.stringify({ name: "Laptop", price: 45000 })

});

1. **PATCH** → Update part of the data

fetch("https://api.example.com/products/1", {

method: "PATCH",

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

body: JSON.stringify({ price: 42000 })

});

1. **DELETE** → Remove data

fetch("https://api.example.com/products/1", {

method: "DELETE"

});

👉 Extra:

* **HEAD** → Get only headers (no body).
* **OPTIONS** → Find out allowed methods.

**4. Handling Response**

Every fetch() returns a Response object.

Common methods:

* .json() → parse JSON data
* .text() → get plain text
* .blob() → for files (images, pdf, etc.)
* .arrayBuffer() → binary data

Example:

fetch("data.json")

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

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

**5. Using async/await (Cleaner)**

async function getData() {

try {

let response = await fetch("https://jsonplaceholder.typicode.com/posts");

let data = await response.json();

console.log(data);

} catch (err) {

console.error("Error:", err);

}

}

getData();

**6. When to Use Fetch (API use cases)**

* **GET data** from server → news, weather, products, etc.
* **POST data** to server → forms, new users, orders.
* **Update data** → profile edit, cart update.
* **Delete data** → remove item, delete account.

👉 In short: Use API when your app needs **dynamic data** (not hardcoded).

**7. Example Projects**

1. **Fetch local JSON file** (e.g., products.json) → show products in your site.
2. **Fetch random joke API** → display jokes on button click.
3. **Fetch random dog API** → show random images.
4. **Send form data with POST** → save user info.

**8. Error Handling**

Always handle errors with .catch() or try...catch.

fetch("https://wrong.url")

.then(res => {

if (!res.ok) throw new Error("Network response was not ok");

return res.json();

})

.catch(err => console.error("Error:", err));

**9. Fetch vs XMLHttpRequest**

* Fetch → Modern, promise-based, cleaner syntax.
* XHR → Older, callback-based, more complex.

**10. Quick Summary (Mnemonic)**

* **GET** → take
* **POST** → add
* **PUT** → replace
* **PATCH** → modify
* **DELETE** → remove