DOM Manipulation & Events in JavaScript

Learn to interact with web pages dynamically

R.A: Assad Ullah Khan

in [assadullahkhan]()

[AssadUllahKhan]()

DOM Manipulation

What is the DOM?

Concept

The DOM (Document Object Model) is a programmatic representation of a web page.
It allows JavaScript to access, manipulate, and change HTML elements, attributes, and content dynamically.

- Think of the DOM as a tree-like structure of the webpage.
- Every HTML element is a node in this tree.
- Through the DOM, JS can interact with the page after it is loaded.

```
<!-- HTML -->
<p id="greeting">Hello, World!</p>

<!-- JavaScript -->
<script>
  const para = document.getElementById("greeting");
  console.log(para.textContent);     // Output: Hello, World!
</script>
```

Key Idea:

The DOM is the bridge between HTML and JavaScript, making web pages dynamic and interactive.

DOM Tree Structure

Concept

The DOM Tree Structure represents the HTML elements of a webpage as a hierarchical tree:

- Each HTML tag becomes a node in the tree.
- Nodes can have child nodes, parent nodes, and sibling nodes.
- JavaScript can traverse this tree to access or modify elements.

Example HTML

```html
<!DOCTYPE html>
<html>
 <body>
  <h1>Welcome</h1>
  <p>Hello, World!</p>
 </body>
</html>
```

DOM Tree Representation

```
html
 └── body
      ├── h1
      └── p
```

Key Points

- Parent Node: Node directly above the current node
- Child Node: Node directly below the current node
- Sibling Node: Nodes on the same level

Selecting Elements

Concept

Before you can manipulate HTML elements with JavaScript, you must select them.
JavaScript provides multiple methods to access elements in the DOM:

- getElementById() → Selects one element by its id.
- querySelector() → Selects the first element that matches a CSS selector.
- querySelectorAll() → Selects all elements that match a CSS selector (returns NodeList).

Example 1 – getElementById

```
<p id="message">Hello!</p>

<script>
  const msg = document.getElementById("message");
  msg.textContent = "Hello, JavaScript!";
</script>
```

Selecting Elements

Example 2 – querySelector

```
<p class="greet">Hi!</p>

<script>
  const greet = document.querySelector(".greet");
  greet.style.color = "blue";            // Changes text color
</script>
```

Key Points

- **getElementById()** → Returns single element (fast).
- **querySelector()** → Can use CSS selectors (flexible).
- Use **querySelectorAll()** → to work with multiple elements at once.

Changing Text and HTML

Concept

Once you've selected an element, you can change its content using the DOM.
JavaScript provides different properties to modify text or HTML inside elements:

- textContent → Changes or gets plain text (ignores HTML tags).
- innerHTML → Changes or gets HTML content (renders HTML tags).

```
<p id="message">Welcome!</p>

<script>
  const msg = document.getElementById("message");
  msg.textContent = "Hello, JavaScript Learners!";
</script>
```

Output:

The text inside the paragraph becomes **"Hello, JavaScript Learners!"**

Changing Text and HTML

Changing HTML

```
<div id="container">Old content</div>

<script>
  const div = document.getElementById("container");
  div.innerHTML = "<h2>New <em>Styled</em>
Content</h2>";
</script>
```

Output:

Displays: New Styled Content (with HTML formatting applied)

Changing CSS Styles

Concept

JavaScript can dynamically change CSS styles of HTML elements using the DOM style property. This allows you to modify colors, fonts, sizes, margins, and more — directly from your script.

Example 1 – Inline Style Change

```
<h2 id="heading">Welcome to JS!</h2>

<script>
  const title = document.getElementById("heading");
  title.style.color = "blue";
  title.style.fontSize = "30px";
  title.style.backgroundColor = "lightgray";
</script>
```

Result:

The heading turns blue, becomes 30px in size, and gets a gray background.

Changing CSS Styles

Example 2 – Toggle Styles with a Button

```
<p id="text">Click the button to change my color!</p>
<button onclick="changeColor()">Change Color</button>

<script>
 function changeColor() {
  const text = document.getElementById("text");
  text.style.color = "red";
 }
</script>
```

Output:

- When the user clicks the button, the paragraph text color changes to red.

Changing CSS Styles

Key Notes

- Styles applied via JS are inline styles (applied directly to the element).
- You can combine JS with CSS classes for cleaner styling.
- Prefer classList.add() or classList.toggle() for better control (we'll cover that soon).

Creating and Removing Elements

Creating Elements

- In JavaScript, you can create new HTML elements dynamically using the document.createElement() method.

Example:

```
const newDiv = document.createElement("div");
newDiv.textContent = "Hello, World!";
document.body.appendChild(newDiv);
```

This creates a new <div> and adds it to the webpage.

Removing Elements

You can remove elements using:

- element.remove() — removes the element itself.
- parent.removeChild(child) — removes a specific child from its parent.

Example:

```
const element = document.getElementById("myDiv");
element.remove();
```

Appending Child Elements

What is appendChild()?

- The appendChild() method is used to add a new element as the last child of a parent element in the DOM.
- It helps dynamically insert content into an existing HTML structure.

Example:

```
const parentDiv = document.getElementById("container");
const newPara = document.createElement("p");
newPara.textContent = "This is a new paragraph!";
parentDiv.appendChild(newPara);
```

👉 This code adds a new <p> inside the <div id="container">.

⚡ Key Notes

- You can only append one node at a time.
- The element being appended is moved, not copied, if it already exists elsewhere in the DOM.
- For adding multiple elements, you can use methods like append() or loop through elements.

Handling Attributes (setAttribute, getAttribute)

What are Attributes?

- Attributes define additional information about HTML elements.
  (e.g., src, href, id, class, alt etc.)

Accessing Attributes:

getAttribute("attributeName") → Gets the value of an attribute.

```
let link = document.querySelector("a");
console.log(link.getAttribute("href"));
```

Modifying Attributes

setAttribute("attributeName", "value") → Changes or adds an attribute.

```
link.setAttribute("href", "https://google.com");
```

Handling Attributes (setAttribute, getAttribute)

Removing Attributes

removeAttribute("attributeName") → Removes an existing attribute.

```
link.removeAttribute("target");
```

Tip:

Use attributes carefully — changing id, src, or class dynamically can affect styles or scripts.

Navigating Parent, Child, and Sibling Nodes

Understanding Node Relationships

- Every element in the DOM is connected like a family tree —where elements have parents, children, and siblings.

Accessing Parent Element

element.parentElement → Gets the parent node of an element.

```
const item = document.querySelector(".list-item");
console.log(item.parentElement);
```

Accessing Child Elements

element.children → Returns all child elements (HTMLCollection).

```
const list = document.querySelector("ul");
console.log(list.children);
```

Navigating Parent, Child, and Sibling Nodes

Accessing Sibling Elements

element.nextElementSibling → Gets the next sibling.
element.previousElementSibling → Gets the previous sibling.

```
console.log(item.nextElementSibling);
```

Tip:

DOM navigation helps in dynamic UI updates — like highlighting, deleting, or moving items in lists or menus.

Events in JavaScript

Event Listeners (click, mouseover, etc.)

Explanation:

- Event listeners are functions that wait for a specific user interaction (like a click, hover, or key press) and then run a piece of code when that event happens.

Common Event Types:

click → Triggered when a user clicks an element.
mouseover → Runs when the mouse hovers over an element.
mouseout → Runs when the mouse leaves an element.
keydown / keyup → Runs when a key is pressed or released.

Example:

```
const btn = document.getElementById("myBtn");

btn.addEventListener("click", function() {
  alert("Button Clicked!");
});
```

Event Listeners (click, mouseover, etc.)

Output:

When the button is clicked, an alert message appears — “Button Clicked!”.

Key Point:

You can attach multiple listeners to one element, and remove them later if needed.

Introduction to Events in JavaScript

Explanation:

- Events are actions or occurrences that happen in the browser — usually triggered by user interaction (like clicking, typing, or submitting a form). JavaScript allows you to “listen” and “respond” to these actions dynamically.

Common Event Examples:

- Clicking a button
- Typing in an input box
- Submitting a form
- Hovering over an image

Example:

```
function showMessage() {
  console.log("Event triggered!");
}

document.getElementById("myButton").onclick = showMessage;
```

Event Types in JavaScript

Explanation:

- JavaScript supports different categories of events depending on what the user interacts with — mouse, keyboard, or form elements.

Mouse Events:

Used for actions with the mouse or touchpad.

```
onclick          // when an element is clicked
onmouseover      // when mouse hovers over an element
onmouseout       // when mouse leaves an element
ondblclick       // on double-click
```

Keyboard Events:

Triggered by keyboard interactions.

```
onkeydown        // when a key is pressed down
onkeyup          // when a key is released
onkeypress       // when a key is pressed and held
```

Event Types in JavaScript

Form Events:

Fired during form interactions.

```
onsubmit    // when form is submitted
onchange    // when input value changes
onfocus     // when input gets focus
onblur      // when input loses focus
```

Keyboard Events:

```
document.querySelector("input").onchange = () => {
  alert("Value changed!");
};
```

Key Point:

Each event type helps capture different user actions to make your web app dynamic and user-friendly.

Handling Events with Functions

Explanation:

- Event handling means writing functions that respond when a user interacts with a webpage (like clicking a button or typing in a box).Functions can be defined separately and linked to events for cleaner, reusable code.

Syntax Example:

```
function showMessage() {
  alert("Button Clicked!");
}

const btn = document.getElementById("myBtn");
btn.addEventListener("click", showMessage);
```

Handling Events with Functions

Explanation of Code:

- The function showMessage() runs when the button is clicked.
- You don’t use parentheses () while passing the function — otherwise, it runs immediately instead of waiting for the event.

Key Tip:

- Always use separate functions for event handling to make your code modular and easy to debug.

Inline Event Handlers & addEventListener()

Concept:

There are two common ways to handle events in JavaScript:

1) Inline Event Handlers (written directly in HTML)

2) addEventListener() (preferred, modern method)

Inline Event Handler Example

```
<button onclick= "alert('Button Clicked!')">Click Me</button>
```

Not recommended for large projects because it mixes HTML and JS.

addEventListener() Example

```
const btn = document.querySelector("button");
btn.addEventListener("click", () => {
  alert("Button Clicked using addEventListener!");
});
```

Inline Event Handlers & addEventListener()

Advantages:

- Keeps JS separate from HTML
- Allows multiple event listeners on one element
- Easier to remove later using removeEventListener()

Removing Event Listeners

Concept:

Sometimes you need to remove an event listener after it has done its job — for example, to prevent multiple triggers or improve performance.
You can do this using removeEventListener().

Example:

```
function showMessage() {
  console.log("Button clicked!");
  button.removeEventListener("click", showMessage); // removes listener
after first click
}

const button = document.querySelector("button");
button.addEventListener("click", showMessage);
```

Output:

```
Button clicked!
```

Removing Event Listeners

Important Notes:

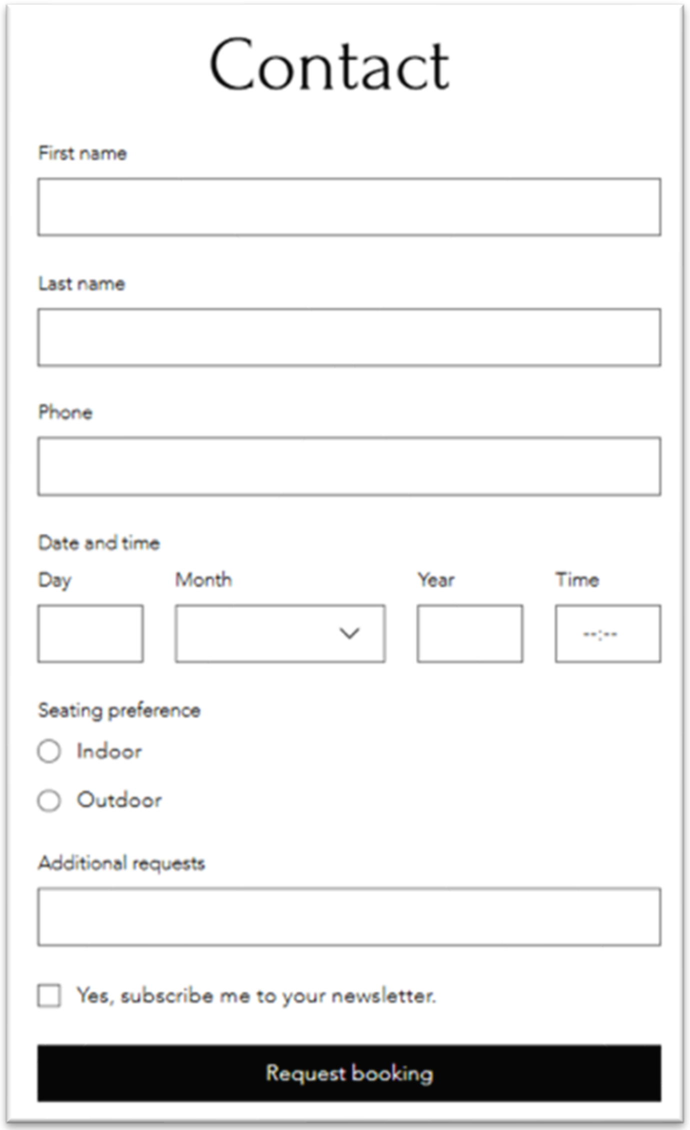
- You must use the same function reference (not an anonymous function) when removing the listener.
- Works only if both addEventListener and removeEventListener refer to the same event type and function.

Form Handling & Validation

Concept:

Form handling allows JavaScript to capture, validate, and process user inputs before submitting data to the server.

Validation ensures users enter correct and complete information (like checking if email is valid or password isn't empty).



Example:

```
<form id="loginForm">
  <input type="text" id="username" placeholder="Enter username">
  <input type="password" id="password" placeholder="Enter password">
  <button type="submit">Login</button>
</form>

<script>
document.getElementById("loginForm").addEventListener("submit",
function(event) {
  event.preventDefault(); // stops form from submitting automatically

  const username = document.getElementById("username").value;
  const password = document.getElementById("password").value;

  if (username === "" || password === "") {
    alert("All fields are required!");
  } else {
    alert("Login successful!");
  }
});
</script>
```

Any Question

Thank You