DOM Manipulation & Events in JavaScript

Learn to interact with web pages dynamically

R.A: Assad Ullah Khan

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

```
<!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 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 .

⚡ 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.

Any Question

Thank You