

# DOM Manipulation

## Complete Practical Guide

Master JavaScript DOM Manipulation with Real Examples

© 2026 - JavaScript DOM Guide

# Table of Contents

1. Introduction to DOM
2. Selecting Elements
3. Changing Content
4. Changing HTML Inside
5. Changing Styles
6. Attributes Manipulation
7. Creating Elements
8. Removing Elements
9. Events (addEventListener)
10. Show/Hide Elements
11. Form Validation
12. Advanced Techniques
13. Best Practices
14. Common Pitfalls

# 1. Introduction to DOM

The Document Object Model (DOM) is a programming interface for web documents. It represents the page so that programs can change the document structure, style, and content. The DOM represents the document as nodes and objects; that way, programming languages can interact with the page.

## What is DOM?

The DOM is a tree-like structure where each HTML element is a node. JavaScript can access and manipulate these nodes to dynamically change the content, structure, and style of a web page without requiring a page reload.

## Why DOM Manipulation?

- Create dynamic and interactive web pages
- Respond to user actions in real-time
- Update content without page refresh
- Build modern single-page applications (SPAs)
- Enhance user experience with smooth interactions

## 2. Selecting Elements

Before you can manipulate elements, you need to select them. JavaScript provides multiple methods to select DOM elements:

### HTML:

```
<h1 id="title">Hello DOM</h1>
<p class="text">Paragraph 1</p>
<p class="text">Paragraph 2</p>
<button onclick="selectDemo()">Select</button>
```

### JavaScript:

```
function selectDemo() {
  let byId = document.getElementById("title");
  let byClass = document.getElementsByClassName("text");
  let byTag = document.getElementsByTagName("p");
  let query = document.querySelector(".text");
  let queryAll = document.querySelectorAll("p");
  console.log(byId, byClass, byTag, query, queryAll);
}
```

■ **Note:** `querySelector` and `querySelectorAll` are the most flexible and modern methods. They accept any CSS selector.

### 3. Changing Content

You can change the text content of elements using `innerText` or `textContent` properties:

#### HTML:

```
<p id="para">Old Text</p>
<button onclick="changeText()">Change Text</button>
```

#### JavaScript:

```
function changeText() {
  let p = document.getElementById("para");
  p.innerText = "New Text";
}
```

#### Difference between `innerText` and `textContent`:

- **innerText**: Returns visible text, aware of styling (hidden elements are ignored)
- **textContent**: Returns all text content, including hidden elements
- **innerHTML**: Returns or sets HTML content (covered in next section)

## 4. Changing HTML Inside

Use innerHTML to insert HTML elements and tags inside an element:

### HTML:

```
<div id="box">Normal</div>  
<button onclick="changeHTML()">Change HTML</button>
```

### JavaScript:

```
function changeHTML() {  
    document.getElementById("box").innerHTML = "<b>Bold Text</b>";  
}
```

■■ **Security Warning:** Be careful with innerHTML when inserting user-generated content as it can lead to XSS (Cross-Site Scripting) attacks. Use `textContent` for plain text.

## 5. Changing Styles

Modify CSS styles directly using the style property:

### HTML:

```
<p id="colorText">Color Me</p>
<button onclick="styleChange()">Change Style</button>
```

### JavaScript:

```
function styleChange() {
  let el = document.getElementById("colorText");
  el.style.color = "red";
  el.style.fontSize = "25px";
  el.style.backgroundColor = "yellow";
}
```

### CSS Property Names in JavaScript:

CSS properties with hyphens are converted to camelCase in JavaScript:

- background-color → backgroundColor
- font-size → fontSize
- border-radius → borderRadius
- margin-top → marginTop

## 6. Attributes Manipulation

Get, set, or remove HTML attributes using `getAttribute()`, `setAttribute()`, and `removeAttribute()`:

### HTML:

```
  
<button onclick="changeImage()">Change Image</button>
```

### JavaScript:

```
function changeImage() {  
  let img = document.getElementById("img");  
  img.setAttribute("src", "https://via.placeholder.com/150");  
  // Alternative: img.src = "https://via.placeholder.com/150";  
}
```

### Common Attribute Methods:

```
// Get attribute  
let value = element.getAttribute("href");  
  
// Set attribute  
element.setAttribute("disabled", "true");  
  
// Remove attribute  
element.removeAttribute("disabled");  
  
// Check if attribute exists  
if (element.hasAttribute("data-id")) { ... }
```

## 7. Creating Elements

Create new DOM elements dynamically using `createElement()` and `appendChild()`:

### HTML:

```
<ul id="list">
  <li>Item 1</li>
</ul>
<button onclick="addItem()">Add Item</button>
```

### JavaScript:

```
function addItem() {
  let li = document.createElement("li");
  li.innerText = "New Item";
  document.getElementById("list").appendChild(li);
}
```

### Other Insertion Methods:

```
// Insert before first child
parent.prepend(element);

// Insert after last child
parent.append(element);

// Insert before reference element
parent.insertBefore(newElement, referenceElement);

// Modern methods
element.before(newElement); // Insert before element
element.after(newElement);  // Insert after element
```

## 8. Removing Elements

Remove elements from the DOM using the `remove()` method:

### HTML:

```
<ul id="removeList">
  <li id="removeMe">Delete Me</li>
</ul>
<button onclick="removeItem()">Remove</button>
```

### JavaScript:

```
function removeItem() {
  let item = document.getElementById("removeMe");
  item.remove();
}
```

### Alternative Removal Methods:

```
// Old method (still works)
let item = document.getElementById("removeMe");
item.parentNode.removeChild(item);

// Remove all children
parent.innerHTML = ""; // Simple but loses event listeners

// Better way to remove all children
while (parent.firstChild) {
  parent.removeChild(parent.firstChild);
}
```

## 9. Events (addEventListener)

Handle user interactions using `addEventListener()`. This is the modern and recommended way to handle events:

### HTML:

```
<button id="eventBtn">Click Me</button>
```

### JavaScript:

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

### Common Event Types:

- **click**: User clicks on an element
- **mouseover**: Mouse moves over an element
- **mouseout**: Mouse moves away from an element
- **keydown**: User presses a key
- **keyup**: User releases a key
- **submit**: Form is submitted
- **change**: Input value changes
- **focus**: Element receives focus
- **blur**: Element loses focus

### Event Object:

```
element.addEventListener("click", function(event) {
  console.log(event.target); // Element that triggered event
  console.log(event.type);   // Event type (e.g., "click")
  event.preventDefault();   // Prevent default action
  event.stopPropagation();   // Stop event bubbling
});
```

## 10. Show/Hide Elements

Toggle element visibility by changing the display CSS property:

### HTML:

```
<p id="toggleText">Hide Me</p>
<button onclick="toggle()">Toggle</button>
```

### JavaScript:

```
function toggle() {
  let el = document.getElementById("toggleText");
  if (el.style.display === "none") {
    el.style.display = "block";
  } else {
    el.style.display = "none";
  }
}
```

### Alternative: Using classList:

```
// CSS
.hidden { display: none; }

// JavaScript
element.classList.toggle("hidden");

// Other classList methods
element.classList.add("active");
element.classList.remove("active");
element.classList.contains("active"); // Returns true/false
```

# 11. Form Validation

Validate user input before form submission:

## HTML:

```
<input type="text" id="name">  
<button onclick="validate()">Submit</button>
```

## JavaScript:

```
function validate() {  
  let name = document.getElementById("name").value;  
  if (name === "") {  
    alert("Name Required");  
  } else {  
    alert("Success");  
  }  
}
```

## Advanced Form Validation Example:

```
function validateEmail() {  
  let email = document.getElementById("email").value;  
  let emailPattern = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;  
  
  if (!emailPattern.test(email)) {  
    alert("Invalid email format");  
    return false;  
  }  
  return true;  
}
```

## 12. Advanced DOM Techniques

### 12.1 Event Delegation

Event delegation is a technique where you attach a single event listener to a parent element instead of multiple listeners to child elements:

```
// Instead of adding click event to each button
document.getElementById("buttonContainer")
  .addEventListener("click", function(e) {
    if (e.target.tagName === "BUTTON") {
      console.log("Button clicked:", e.target.textContent);
    }
  });
```

### 12.2 Document Fragment

Use DocumentFragment for better performance when adding multiple elements:

```
let fragment = document.createDocumentFragment();
for (let i = 0; i < 100; i++) {
  let li = document.createElement("li");
  li.textContent = "Item " + i;
  fragment.appendChild(li);
}
// Single DOM update instead of 100
document.getElementById("list").appendChild(fragment);
```

### 12.3 Data Attributes

Store custom data in HTML elements using data-\* attributes:

```
<!-- HTML -->
<div id="user" data-user-id="123" data-role="admin">John</div>

// JavaScript
let user = document.getElementById("user");
console.log(user.dataset.userId); // "123"
console.log(user.dataset.role);   // "admin"

// Set data attribute
user.dataset.status = "active";
```

# 13. Best Practices

## 1. Use Modern Selectors

Prefer `querySelector()` and `querySelectorAll()` for their flexibility and CSS selector support.

## 2. Cache DOM References

```
// Bad - searches DOM multiple times
document.getElementById("myDiv").style.color = "red";
document.getElementById("myDiv").style.fontSize = "20px";

// Good - cache the reference
let myDiv = document.getElementById("myDiv");
myDiv.style.color = "red";
myDiv.style.fontSize = "20px";
```

## 3. Minimize DOM Manipulation

Batch DOM changes together to improve performance. Use `DocumentFragment` or build HTML strings when creating multiple elements.

## 4. Use Event Delegation

Instead of attaching events to many child elements, attach one event to a parent and use event bubbling.

## 5. Avoid innerHTML for User Input

Using `innerHTML` with user-generated content can lead to XSS vulnerabilities. Use `textContent` or `createElement()` instead.

## 6. Clean Up Event Listeners

```
// Remove event listener when no longer needed
function handleClick() {
  console.log("Clicked");
}

element.addEventListener("click", handleClick);
// Later...
element.removeEventListener("click", handleClick);
```

# 14. Common Pitfalls to Avoid

## 1. Script Loading Before DOM

```
// Problem: Script runs before DOM is ready
let element = document.getElementById("myDiv"); // null!

// Solution 1: Place script at end of body
// Solution 2: Use DOMContentLoaded event
document.addEventListener("DOMContentLoaded", function() {
  let element = document.getElementById("myDiv");
  // Now it works!
});
```

## 2. Not Checking if Element Exists

```
// Bad - will throw error if element does not exist
document.getElementById("missing").style.color = "red";

// Good - check first
let element = document.getElementById("myDiv");
if (element) {
  element.style.color = "red";
}
```

## 3. Modifying Arrays While Iterating

```
// Bad - length changes during iteration
let items = document.querySelectorAll(".item");
for (let i = 0; i < items.length; i++) {
  items[i].remove(); // Changes items!
}

// Good - convert to array first
let items = Array.from(document.querySelectorAll(".item"));
items.forEach(item => item.remove());
```

## 4. Forgetting Event Context

```
// Problem: "this" context is lost
button.addEventListener("click", myObject.handleClick);

// Solution: Use arrow function or bind
button.addEventListener("click", () => myObject.handleClick());
// or
button.addEventListener("click", myObject.handleClick.bind(myObject));
```

## Quick Reference Guide

Task	Method/Property	Example
Select by ID	<code>getElementById()</code>	<code>document.getElementById("id")</code>
Select by class	<code>querySelector()</code>	<code>document.querySelector(".class")</code>
Select all	<code>querySelectorAll()</code>	<code>document.querySelectorAll("p")</code>
Change text	<code>innerText</code>	<code>element.innerText = "text"</code>
Change HTML	<code>innerHTML</code>	<code>element.innerHTML = "&lt;b&gt;text&lt;/b&gt;"</code>
Change style	<code>style</code>	<code>element.style.color = "red"</code>
Add class	<code>classList.add()</code>	<code>element.classList.add("active")</code>
Remove class	<code>classList.remove()</code>	<code>element.classList.remove("active")</code>
Toggle class	<code>classList.toggle()</code>	<code>element.classList.toggle("active")</code>
Create element	<code>createElement()</code>	<code>document.createElement("div")</code>
Add to DOM	<code>appendChild()</code>	<code>parent.appendChild(child)</code>
Remove from DOM	<code>remove()</code>	<code>element.remove()</code>
Add event	<code>addEventListener()</code>	<code>element.addEventListener("click", fn)</code>
Get attribute	<code>getAttribute()</code>	<code>element.getAttribute("href")</code>
Set attribute	<code>setAttribute()</code>	<code>element.setAttribute("id", "new")</code>

## Conclusion

DOM manipulation is a fundamental skill for creating interactive and dynamic web applications. By mastering these techniques, you can:

- Create responsive user interfaces that react to user input
- Build single-page applications without page reloads
- Dynamically update content based on user actions or data
- Validate forms and provide instant feedback
- Create smooth animations and transitions
- Optimize performance through efficient DOM manipulation

Remember to always prioritize performance, security, and user experience. Use modern JavaScript methods, cache DOM references, minimize reflows, and protect against XSS attacks.

Continue practicing these techniques and exploring advanced patterns like virtual DOM, shadow DOM, and modern frameworks that build upon these fundamental concepts.

**Happy Coding! ■**