
1. JavaScript DOM Selection

◊ What is DOM?

- The **DOM (Document Object Model)** represents the structure of your HTML page as a tree.
 - JavaScript uses the DOM to **select, read, and manipulate** elements on the page.
-

◊ DOM Selection Methods Summary

Method	Selector Type	Returns	Live?	Use Case
getElementById()	id	Single element/null	✗ No	When selecting one unique element
getElementsByClassName()	.class	HTMLCollection	✓ Yes	To select all elements with a class
querySelector()	CSS selector	First match/null	✗ No	For one element using any CSS selector
querySelectorAll()	CSS selector	NodeList (all)	✗ No	To select all matches using CSS selectors

◊ 1. getElementById("id")

- ✓ Selects **1 element** by ID
- ✓ Returns a **single element** or null

```
• <p id="title">Hello</p>
• let title = document.getElementById("title");
• title.style.color = "green";
• 
```

◊ 2. getElementsByClassName("class")

- ✓ Selects **all elements** with a class
- ✓ Returns an **HTMLCollection** (live)

```
• <p class="note">Note 1</p>
• <p class="note">Note 2</p>
• let notes = document.getElementsByClassName("note");
• notes[1].style.color = "blue";
•
```

◊ 3. querySelector("selector")

- Selects the **first matching** element
 - Works with #id, .class, tag, etc.
 - Returns **single element** or null
- ```
• let title = document.querySelector("#title");
• let note = document.querySelector(".note");
•
```
- 

### ◊ 4. querySelectorAll("selector")

- Selects **all elements** matching a CSS selector
  - Returns a **NodeList** (not live)
- ```
• let items = document.querySelectorAll(".item");
• items[0].style.color = "red";
•
```
-

- Returns: **NodeList** (a list of all matching elements)
 - Use: To get **multiple elements** using **CSS selectors**
 - Not live (does **not auto-update** when DOM changes)
-

⌚ Example:

```
<p class="item">Item 1</p>
<p class="item">Item 2</p>
let items = document.querySelectorAll(".item");
console.log(items.length);      // 2
items[0].style.color = "red";   // Changes first item text to red
```

⚡ Quick Recap Table:

Method	Selector Type	Returns	Live?
getElementsByClassName	Class name	HTMLCollection	<input checked="" type="checkbox"/> Yes
getElementById	ID only	Single Element/null	<input checked="" type="checkbox"/> No

Method	Selector Type	Returns	Live?
querySelector	CSS selector	First Match/null	✗ No
querySelectorAll	CSS selector	NodeList (all)	✗ No

☒ 2. innerText vs textContent vs innerHTML in JavaScript

Feature	innerText	textContent	innerHTML
📝 Returns	Visible text only (no tags)	All text , incl. hidden	HTML + text
👁️ Visibility	Respects CSS (<code>display: none</code> , etc.)	Ignores CSS	Shows raw HTML
✍️ Whitespace	Cleaned up	Preserved	Preserved

☒ Quick Examples:

```
<h1 class="test">Hello this is lambodar <span style="display: none;">this is test text</span></h1>

test.innerText
'Hello this is lambodar'
test.textContent
'Hello this is lambodar this is test text'
test.innerHTML
'Hello this is lambodar <span style="display: none;">this is test text</span>'
```

☑ Best Use:

- Use `innerText` → for what's shown to users.
- Use `textContent` → for **pure text processing**.
- Use `innerHTML` → for **inserting/retrieving HTML**.

3. `getAttribute()` vs `setAttribute()` in JavaScript

`getAttribute()` vs `setAttribute()` in JavaScript

Feature	<code>getAttribute()</code>	<code>setAttribute()</code>
 Purpose	Reads an attribute's value	Sets/updates an attribute's value
 Works On	Any HTML attribute (like <code>href</code> , <code>id</code>)	Any valid HTML attribute
 Use Case	Get custom or built-in attr values	Add/change attributes dynamically
 Note	Doesn't read properties like <code>el.value</code>	Doesn't affect JS properties directly

Syntax:

```
element.getAttribute("attrName")
element.setAttribute("attrName", "value")
```

Example:

```
<a id="link" href="https://example.com" target="_blank">Click me</a>
const link = document.getElementById("link");

link.getAttribute("href");           // 🔍 "https://example.com"
link.setAttribute("href", "#");    // 🚫 Changes href to "#"
link.setAttribute("title", "Go!"); // ✨ Adds title attribute
```

! Common Attributes You Can Access:

- `href`, `src`, `alt`, `id`, `class`, `type`, `name`, `value`, `title`, `data-*` attributes
-



Ways to Select Element + Use `getAttribute()` and `setAttribute()`

Selector Method	Get Attribute Example	Set Attribute Example
<code>getElementById()</code>	<code>document.getElementById("myId").getAttribute("href")</code>	<code>document.getElementById("myId").setAttribute("href", "#")</code>
<code>getElementsByClassName()</code>	<code>document.getElementsByClassName("btn")[0].getAttribute("data-value")</code>	<code>document.getElementsByClassName("btn")[0].setAttribute("data-value", "123")</code>
<code>getElementsByTagName()</code>	<code>document.getElementsByTagName("img")[0].getAttribute("src")</code>	<code>document.getElementsByTagName("img")[0].setAttribute("alt", "Image")</code>
<code>querySelector()</code>	<code>document.querySelector(".card").getAttribute("title")</code>	<code>document.querySelector(".card").setAttribute("title", "Card Title")</code>
<code>querySelectorAll()</code>	<code>document.querySelectorAll("p")[0].getAttribute("id")</code>	<code>document.querySelectorAll("p")[0].setAttribute("id", "para1")</code>

Reminder:

- `.getAttribute("name")` → gets the value
- `.setAttribute("name", "value")` → sets or updates the value

4. How to Apply Styles in JavaScript

1. Using `.style.property`

```
element.style.color = "red";
element.style.backgroundColor = "yellow";
```

- ✓ Directly applies inline styles
-

2. Using `.style.cssText` (set multiple at once)

```
element.style.cssText = "color: white; background: black; padding: 10px;";
```

3. Using `setAttribute("style", "...")`

```
element.setAttribute("style", "font-size: 18px; border: 1px solid gray;");
```

4. Using `classList.add()` / `remove()` / `toggle()`

(Best for reusable styles via CSS classes)

```
element.classList.add("active");
element.classList.remove("hidden");
element.classList.toggle("dark-mode");
```

5. Using ES6+ Loop + Arrow Functions

```
// Apply same style to multiple elements
document.querySelectorAll(".box").forEach(el => {
  el.style.borderRadius = "10px";
  el.style.boxShadow = "0 0 10px rgba(0,0,0,0.2);"
```

```
});
```

6. Using `Object.assign()` (for bulk inline styles)

```
Object.assign(element.style, {
  color: "blue",
  fontWeight: "bold",
  marginTop: "20px"
});
```

Most Used Style Properties in Real Projects:

- `color, backgroundColor`
 - `fontSize, fontWeight, textAlign`
 - `margin, padding, border, borderRadius`
 - `display, flex, grid`
 - `width, height, position, zIndex`
-

Bonus Tip — Toggle Dark Mode Example (ES6):

```
const toggle = document.getElementById("toggleTheme");
toggle.addEventListener("click", () => {
  document.body.classList.toggle("dark-mode");
});
```

5. Access Parent Sibling & Children Elements

New trick to select element for every time we have to write id,class and then select element so easy way for testing your code go to chrome dev tool
html code > hover on element >right click copy js path

then just create var and pase this js path as selected element it will get selected and stores in var

1. Parent Element

-
- element.parentElement
-
-  Goes **one level up**
-  Ignores text nodes (safe to use)

Example:

```
<div class="cl">
    <h3>"hello world"</h3>
    <h3>"hello world"</h3>
    <h3>"hello world"</h3>
</div>

-----
const myel=document.querySelector('h3')

--myel
<h3>"hello world"</h3>

-- myel.parentElement
<div class="cl">...</div>

-- myel.parentElement.parentElement
<body>...</body>

-- myel.parentElement.parentElement.parentElement
```

```
<html lang="en"><head>...</head><body>...</body></html>
```

2. Children Elements

element.children

- Returns **HTMLCollection** of child elements (no text nodes)

- element.firstChild
- element.lastElementChild
-

```
<div class="cl">
    <h3>"hello world"</h3>
    <h3>"hello world"</h3>
    <h3>"hello world"</h3>
</div>
```

```
> const myel = document.querySelector("body > div")
< undefined
> myel
<  > <div class="cl">...</div>
> myel.children
<  > HTMLCollection(3) [h3, h3, h3]
> myel.childNodes
<  > NodeList(7) [text, h3, text, h3, text, h3, text]
>
```

3. Sibling Elements

```
element.previousElementSibling  
element.nextElementSibling
```

- Accesses elements **beside** the current one

Example:

```
<h3>"hello world"</h3>  
<h3>"hello world"</h3>  
<h3>"hello world"</h3>  
</div>
```

```
> myel  
<-- ▼<div class="c1">  
    <h3>"hello world1"</h3>  
    <h3>"hello world2"</h3>  
    <h3>"hello world3"</h3>  
  </div>  
> myel.children[0].nextElementSibling  
<-- <h3>"hello world2"</h3>
```



6. Node vs Element — What's the Difference?

Node (General Building Block)

- A **generic** type — everything in the DOM is a Node.
- Types of nodes:
 - Element node (<div>, <p>)
 - Text node ("Hello")
 - Comment node (<!-- comment -->)
 - Document node (document)
 - etc.

Example:

```
document.body.childNodes;  
// Returns ALL nodes: elements, text, comments, etc.
```

Element (Specific Type of Node)

- A **node that is an HTML element**
- Subtype of Node
- Only represents tags like <div>, <p>, , etc.

Example:

```
document.body.children;  
// Returns ONLY element nodes (ignores text/comments)
```

Key Differences:

Feature	Node	Element
Type	General (includes text, etc.)	Specific (just HTML tags)
Example Nodes	Element, Text, Comment	<div>, <h1>, , etc.
Common Accessors	.childNodes, .firstChild	.children, .firstElementChild
Useful When	Parsing <i>everything</i> in DOM	Working with HTML <i>structure</i>

Quick Code Comparison:

```
// Includes text, comment, etc.  
console.log(document.body.childNodes);  
  
// Only actual HTML elements  
console.log(document.body.children);
```

7. appendChild() & cloneNode() in JavaScript

[project link](#)

1. appendChild()

Adds a node as the **last child** of a parent element. Its like push on webpage anything created with js

```
parent.appendChild(child);
```

Example:

```
<div class="cl1">
    <h3>"hello this is part of class1"</h3>
</div>

<div class="cl2">
    <h3>"hello this is part of class2"</h3>
</div>

<h3 class="testText">Hi this is test text </h3>
```

```
const class1= document.querySelector(".cl1")
const class2= document.querySelector(".cl2")
const testText= document.querySelector('.testText')

class1.appendChild(testText)
```

```
▼<div class="cl1"> == $0
  <h3>"hello this is part of class1"</h3>
  <h3 class="testText">Hi this is test text </h3>
</div>
```

Example 2:

```
const list = document.querySelector("ul");
const newItem = document.createElement("li");
newItem.textContent = "New Item";

list.appendChild(newItem); // Adds <li> to end of <ul>
```

2. element.cloneNode()

Creates a **copy of a node** (shallow or deep).

```
element.cloneNode(true); // Deep clone (with children)  
element.cloneNode(false); // Shallow clone (just the element)
```

Example:

```
const card = document.querySelector(".card");  
const copy = card.cloneNode(true);  
  
document.body.appendChild(copy); // Add the cloned card to body
```

Common Use:

```
const node = document.getElementById("box");  
const clone = node.cloneNode(true); // Clone whole element  
  
document.body.appendChild(clone); // Insert at end of body
```

Key Points:

Method	Purpose	Notes
appendChild()	Insert a node at the end	Moves existing or new node
cloneNode()	Clone element (optionally deep)	Doesn't insert — needs append

```
<div class="container">  
  
<div class="card1 cardMain">c1</div>  
<div class="card2 cardMain">c2</div>  
<div class="card3 cardMain">c3</div>  
  
</div>  
  
const selectCard= document.querySelector(".card1")  
const container = document.querySelector(".container")  
  
container.appendChild(selectCard.cloneNode(true))  
<div class="card1 cardMain">c1</div>
```



Project on use of append ,clone node, innerText , set Attribute

Project1: Using a loop make 100 cards with inner text 1 to 100

html:

```
<div class="container">

    <div class="card1 cardMain">1</div>
    <div class="card2 cardMain">2</div>
    <div class="card3 cardMain">3</div>

</div>
```

Css:

```
1
2  body{
3
4      margin: 0;
5      padding: 0;
6      background-color: #antiquewhite;
7  }
8  .container{
9
10     display: flex;
11     align-items: center;
12     align-content: center;
13     border: 2px solid #black;
14     padding: 20px;
15     justify-content: center;
16     flex-wrap: wrap;
17
18
19
20 }
```

```
2
3  const card= document.querySelector(".card1")
4  const container = document.querySelector(".container")
5
6  for(let i =4;i<=100;i++){
7      const myCard = card.cloneNode();
8      myCard.innerText=i
9      container.appendChild(myCard);
10
11
12 }
13
```

```
const myCard = card.cloneNode();  
याला आपल्याला लूप च्या अंत मध्येच declare करावा लागणार  
कारण बाहेर जर केला तर तो एकदाच कॉपी होणार.  
आपलीला नवीन कॉपी आणि append होणाऱ्या एलिमेंट मध्ये text  
add काराईचे आहे सो आपण innertext use केला
```

Project1: Using a loop make 100 sticker pokemon sprite api

आपल्याला डायनामिकल इलेमेंट क्रिएट करून append करायचा आहे सो डायनामी केली क्रिएट करण्यासाठी आपण लिंक ला manipulate करणार इमेज लिंक ला

```
<h3>Learning appendChild, cloneNode , setAttribute , Create element</h3>
<div class="container">

    

</div>
```

```
const imgUrl= document.querySelector("#img")
const container = document.querySelector(".container")

for(let i =4;i<=100;i++){
    let newImgUrl = imgUrl.cloneNode()
    newImgUrl.setAttribute("src",
`https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/poke
mon/${i}.png`);
    container.appendChild(newImgUrl)
}
```

Concept :

```
setAttribute( src , link)
setAttribute( href , link)
```



7. Element creation using createElement()

[project link](#)

1. Create an Element

```
let element = document.createElement("tagName");
• tagName can be: "div", "p", "h1", "img", etc.
let newDiv = document.createElement("div");
```

2. Add Content (Text)

```
element.textContent = "Hello World!";
newDiv.textContent = "I am a new div!";
```

3. Add Attributes / Classes / IDs

```
element.id = "myId";
element.className = "myClass";
element.setAttribute("title", "tooltip");
newDiv.id = "box1";
newDiv.className = "container";
```

4. Append to the Page

```
document.body.appendChild(element);
document.body.appendChild(newDiv);
```

5. Create and Append Inside Another Element

```
let container = document.getElementById("container");
container.appendChild(newDiv);
```

🔗 Simple Example (HTML + JS)

html:

```
<h3>Learning appendChild, cloneNode , setAttribute , Create element</h3>  
<div class="container">  
</div>
```

Js

```
const newEl = document.createElement("img")  
newEl.src="https://famezip.com/wp-content/uploads/2024/06/arshiya-sharma.png"  
newEl.style.cssText="height:500px;width:400px"  
  
const container=document.querySelector(".container")  
container.append(newEl)
```



Project on use of createElement

Project1: Using a loop make 100 sticker and another p tag which val changes from 1-100

```
<div class="block">
    
        <p>this is 1</p>
    </div>
```

Now create this reference is img in js

```
const block= document.querySelector('.container')

for(let i =1;i<=100;i++){

    const newImg = document.createElement('img')
    newImg.style.cssText="height:100px; width: 100px;"
    const text= document.createElement('p')

    newImg.src=
`https://raw.githubusercontent.com/PokeAPI/sprites/refs/heads/master/sprites/pokemon/${i}.png`
    text.innerText=i
    block.append(newImg, text)

}
```

HTML:

```
<div class="container">

</div>
```

If you want to every time clone or created new el append then always keep this creation or clone in loop

How to Remove Elements in JavaScript

1. Remove HTML Element from the DOM

`element.remove()`

- Most modern way (ES6+)
- Directly removes the element

```
js
CopyEdit
const elem = document.getElementById('myDiv');
elem.remove();
```

Remove all child elements

```
js
CopyEdit
const parent = document.getElementById('container');
parent.innerHTML = ''; // clears all content
```

Good to Know

- Use `remove()` only if you're sure the element exists.
- Always check for `null` before removing:

```
js
CopyEdit
const el = document.getElementById('something');
if (el) el.remove();
```