

🌟 JavaScript DOM & Event Handling — Explained with Fun 🚀

📌 What is Event Handling?

💬 Event handling in JavaScript refers to executing code in response to events (like a button click or mouse hover) on a webpage using the **DOM (Document Object Model)**.

🧠 Common Event Types

Event Type	Description
click	User clicks an element
mouseover	Mouse hovers over an element
submit	Form submission
keydown	Key press on the keyboard
load	Page has fully loaded

📄 Basic Click Event Example

```
<!-- HTML -->
<button id="myButton">Click Me</button>
```

```
// JavaScript
let button = document.getElementById("myButton");
button.addEventListener("click", function () {
  alert("🌟 Button clicked!");
});
```

📖 Explanation:

- We grab the button using `getElementById`.
- We attach a `click` event listener.
- When the button is clicked, an alert pops up saying "Button clicked!".

📄 Output:

🌟 Button clicked!

🔗 Using a Named Function for Event Handling

```
function handleClick() {  
  alert("🚀 Button clicked!");  
}  
button.addEventListener("click", handleClick);
```

☑ This is more reusable than using an anonymous function.

✖ Removing Event Listeners

```
button.removeEventListener("click", handleClick);
```

💡 The function used in `removeEventListener` must be **the same reference** as the one passed to `addEventListener`.

🔧 Console Inspection: `console.log()` vs `console.dir()`

```
console.log(document.getElementsByTagName('span')[0]); // 🌳 DOM Tree  
console.dir(document.getElementsByTagName('span')[0]); // 🔗 Object properties
```

🔗 TagName vs nodeName

```
console.log(document.body.firstChild.nodeName); // Text node or element  
console.log(document.body.firstChild.tagName); // Only element
```

Property	Works For	Description
<code>tagName</code>	Element nodes	Returns tag like <code>DIV</code> , <code>SPAN</code> , etc.
<code>nodeName</code>	Any node	Works for text, comment, etc.

📦 innerHTML vs outerHTML

```
let first = document.getElementById("first");
```

```
console.log(first.innerHTML); // Inside content
console.log(first.outerHTML); // Whole HTML of the element
```

- `innerHTML`: Just the inner content.
- `outerHTML`: Whole element including tags.

```
first.innerHTML = "<i>Hello</i>"; // ✎ Inserts italic text
```

Text Content & Hidden Property

```
console.log(document.body.textContent); // Get text

first.hidden = true; // ✕ Hide element
first.hidden = false; // ☑ Show element
```

Attribute Methods

Method	Purpose
<code>hasAttribute(name)</code>	Check if attribute exists
<code>getAttribute(name)</code>	Get value of an attribute
<code>setAttribute(name, value)</code>	Set attribute
<code>removeAttribute(name)</code>	Remove attribute
<code>attributes</code>	Get all attributes

```
first.setAttribute("class", "highlight");
console.log(first.getAttribute("class")); // "highlight"
first.removeAttribute("class");
```

Data-* Attributes

```
<div id="first" data-game="mario" data-player="luigi"></div>
```

```
console.log(first.dataset.game); // "mario"
console.log(first.dataset.player); // "luigi"
```

💡 Custom attributes starting with **data-** are accessible via **.dataset**.

🧱 Inserting HTML: 3 Ways

1 Using **innerHTML**

```
let div = document.getElementsByTagName('div')[0];
div.innerHTML += "<h1>Hello World!</h1>";
```

2 Using **createElement** & **appendChild**

```
let newDiv = document.createElement("div");
newDiv.innerHTML = "<h1>Hello World!</h1>";
div.appendChild(newDiv);
```

3 Using **createTextNode**

```
let textNode = document.createTextNode("📄 This is plain text");
div.appendChild(textNode);
```

🔧 Node Manipulation Methods

Method	Description
append()	Add at end
prepend()	Add at start
before()	Insert before element
after()	Insert after element
replaceWith()	Replace node
remove()	Remove the element

```
first.insertAdjacentHTML('beforebegin', '<div>👉 beforebegin</div>');
first.insertAdjacentHTML('afterend', '<div>👈 afterend</div>');
```

className vs classList


```
first.className = "red bold";           // Adds multiple classes
first.classList.add("hidden");          // Adds a class
first.classList.remove("red");          // Removes class
first.classList.toggle("bold");         // Toggle class on/off
console.log(first.classList.contains("bold")); // true/false
```

setTimeout & setInterval

 **setTimeout** - Runs once after delay

```
let timeoutId = setTimeout(() => {
  alert("⚡ Timeout Triggered!");
}, 2000);

clearTimeout(timeoutId); // Cancels the timeout
```

 **setInterval** - Runs repeatedly

```
let intervalId = setInterval(() => {
  console.log("🔄 Interval Running...");
}, 3000);

clearInterval(intervalId); // Stops the interval
```

Browser Events

Event Type	Examples
Mouse Events	click, mouseover, mousedown
Keyboard Events	keydown, keyup
Form Events	submit, focus
Document Events	DOMContentLoaded

Event Handler Example

```
<button onclick="alert('👋 Hello')">Click Me</button>
```

JavaScript-Only Way (Preferred)

```
let container = document.querySelector(".container");
container.onclick = () => {
  container.innerHTML = "👋 Hello from JS!";
};
```

⚠️ If both HTML and JS define handlers, JS handler takes precedence.

🧠 Summary

- Use `addEventListener` for multiple handlers and better control.
 - Use `.dataset` for accessing custom data attributes.
 - Prefer `classList` over `className` for toggling classes.
 - Manage timers using `setTimeout`, `setInterval`, `clearTimeout`, and `clearInterval`.
 - DOM manipulation is best done via `createElement`, `append`, or `insertAdjacentHTML`.
-