LECTURE 13

DOM Manipulation and Event Handling

How JavaScript reads, changes, and reacts to the structure of a web page — the engine behind every interactive site.

CSC336 Web Technologies

OVERVIEW

In This Lecture

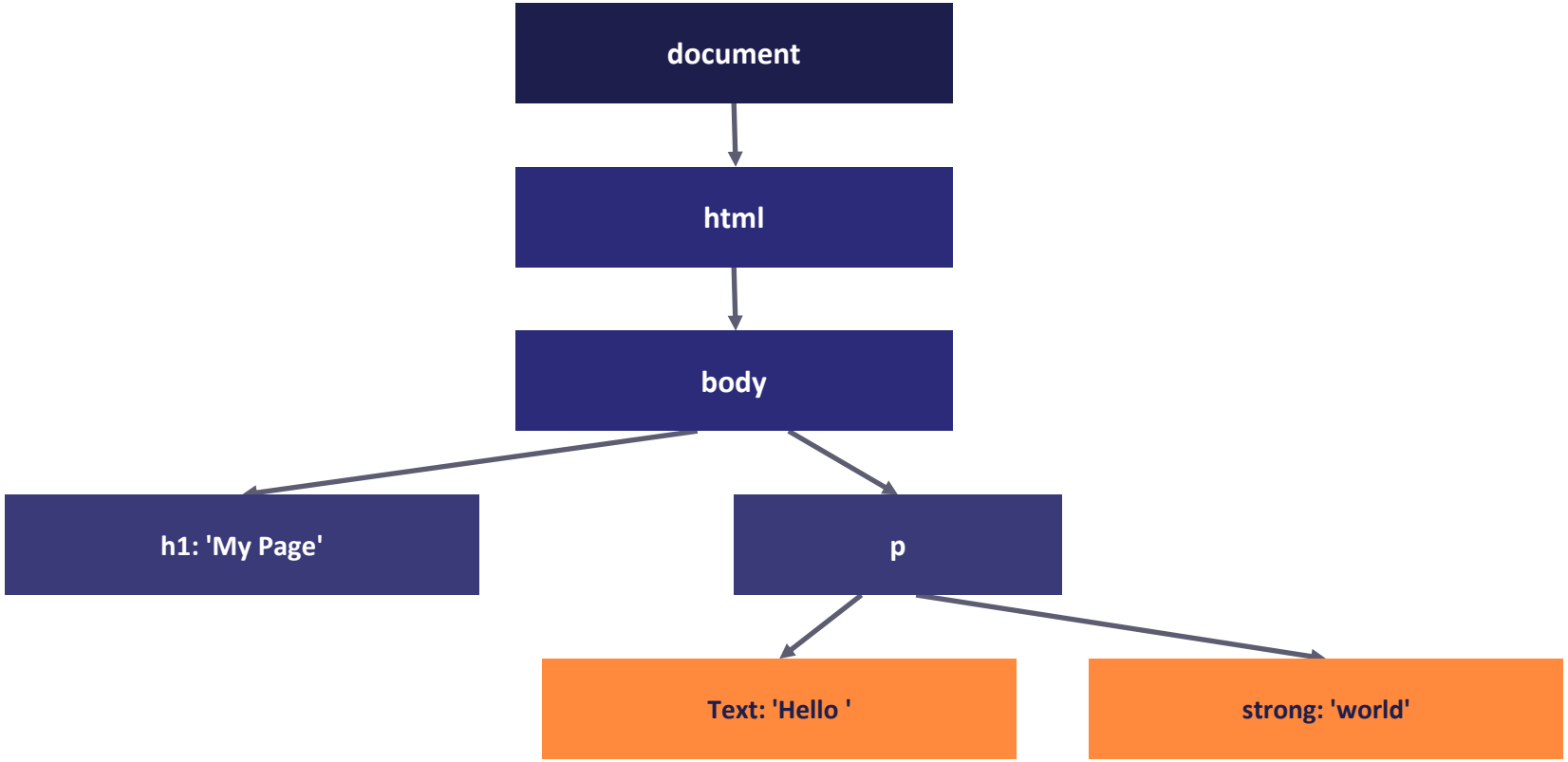
1. What the DOM tree, nodes, and the document object actually are
2. Selecting elements with getElementById, querySelector, and querySelectorAll
3. Creating, updating, and removing elements
4. Working with classList and inline styles
5. Handling events with addEventListener and the event object
6. Event bubbling, capturing, delegation, preventDefault, and stopPropagation

A Page, Before It Becomes a Tree

```
<!DOCTYPE html>
<html>
  <body>
    <h1>My Page</h1>
    <p>Hello <strong>world</strong></p>
  </body>
</html>
```

The DOM (Document Object Model) is a live, in-memory, tree-shaped representation of this HTML that the browser builds after parsing it.

Every Tag Becomes a Node



Node: any single item in the tree. Element: a node representing an HTML tag. body is the PARENT of h1 and p; h1 and p are SIBLINGS.

The DOM Is Not the Same as Your HTML File

NOTE

The DOM is what the browser builds from your HTML -- and JavaScript can change it after the fact. "View Page Source" shows the original file; "Inspect Element" (DevTools) shows the CURRENT DOM, which may look very different after scripts have run.

getElementById and querySelector

```
// By id -- a single element, or null if not found
const title = document.getElementById("main-title");

// By CSS selector -- the FIRST matching element, or null
const firstButton = document.querySelector(".btn");
const firstInput = document.querySelector("input[type='email']");

// By CSS selector -- ALL matching elements, as a NodeList
const allButtons = document.querySelectorAll(".btn");
```

querySelector/querySelectorAll accept ANY valid CSS selector -- by class, tag, attribute, or combinations -- far more flexible than getElementById.

NodeList Supports forEach

```
allButtons.forEach(btn => console.log(btn.textContent));
```

A NodeList Is Not an Array

TIP

A NodeList supports forEach, but not map or filter directly. Convert it first if you need those: Array.from(allButtons) or [...allButtons].

Creating and Inserting Elements

```
const newItem =
  document.createElement("li");
newItem.textContent = "New task";

const list =
  document.querySelector("#task-list");
list.appendChild(newItem);
// list.prepend(newItem);
// list.insertBefore(newItem,
//   list.children[1]);
```

#task-list AFTER appendChild(newItem)

- Buy groceries
- Walk the dog
- New task

Updating Content and Attributes

```
const heading = document.querySelector("h1");
heading.textContent = "Updated Title";      // sets plain text (safe)
heading.innerHTML = "<em>Updated</em> Title"; // parses the string AS HTML

const link = document.querySelector("a");
link.setAttribute("href", "https://example.com");
console.log(link.getAttribute("href"));
link.removeAttribute("target");
```

innerHTML and Security (XSS)

WARNING

innerHTML parses whatever string you give it as real HTML. If that string comes from user input and you insert it with innerHTML, a malicious user could inject a <script> tag -- this is Cross-Site Scripting (XSS). Prefer textContent unless you specifically need to insert markup.

Removing an Element

```
const item = document.querySelector("#task-3");
item.remove(); // modern, simplest way

// Older way, still seen in existing code:
item.parentNode.removeChild(item);
```

classList vs. Raw Style Strings

```
const box =
  document.querySelector(".box");

box.classList.add("highlighted");
box.classList.remove("hidden");
box.classList.toggle("active");
box.classList.contains("active");
// true or false
```

Setting Individual CSS Properties

```
box.style.backgroundColor = "yellow";
box.style.display = "none";
box.style.fontSize = "18px";
```

The style property writes INLINE styles -- CSS applied directly on the element, with the highest priority.

Prefer classList Over style

TIP

Toggling a class keeps "what it should look like" (CSS) separate from "when it should look that way" (JavaScript), and it's easier to change the look later without touching your script.

addEventListener

```
const button = document.querySelector("#save-btn");

button.addEventListener("click", function () {
  console.log("Button was clicked!");
});

// Arrow function version
button.addEventListener("click", () => console.log("Clicked!"));

function handleClick() { console.log("Handled once"); }
button.addEventListener("click", handleClick);
button.removeEventListener("click", handleClick); // same function reference
```

addEventListener is preferred over onclick=... because it allows MULTIPLE listeners on the same event, and a specific one can be removed later.

The Event Object

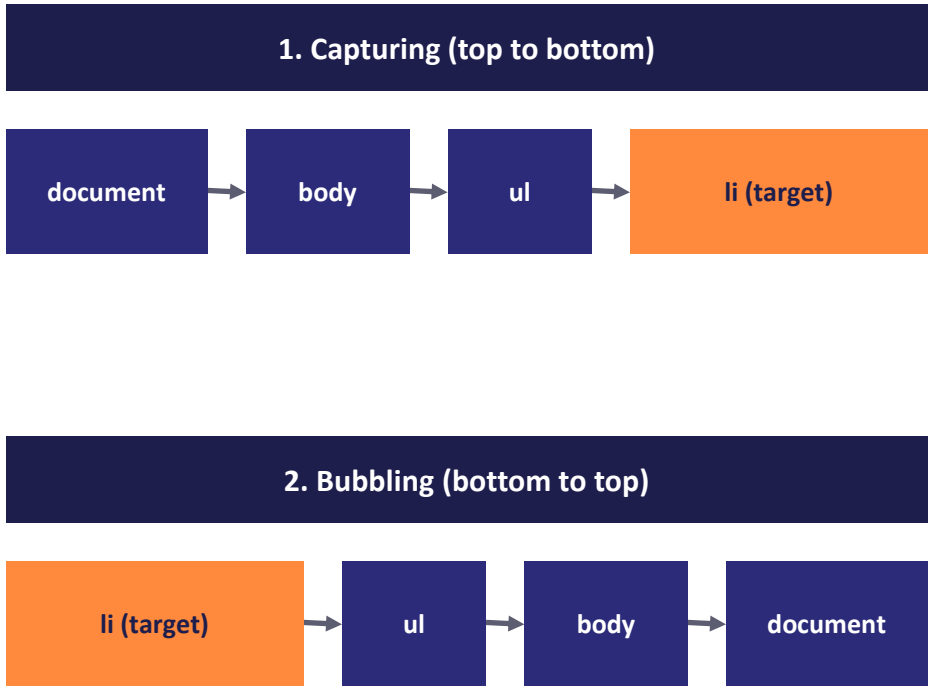
```
document.querySelector("input").addEventListener("keydown", (event) => {
  console.log(event.key);    // which key was pressed, e.g. "Enter"
  console.log(event.target); // the exact element the event happened on
  console.log(event.type);   // "keydown"
});

document.querySelector("form").addEventListener("submit", (event) => {
  event.preventDefault(); // stop the page reload
  console.log("Form data captured without reloading the page");
});
```

Common Events You Will Use Constantly

Category	Events
Mouse / pointer	click, mouseover, mouseout
Keyboard	keydown, keyup
Forms	submit, input, change
Page lifecycle	load, DOMContentLoaded

Two Phases of Every Event



By default, addEventListener listens during the BUBBLING phase. Pass true as a third argument to listen during capturing instead.

One Listener, Many Children

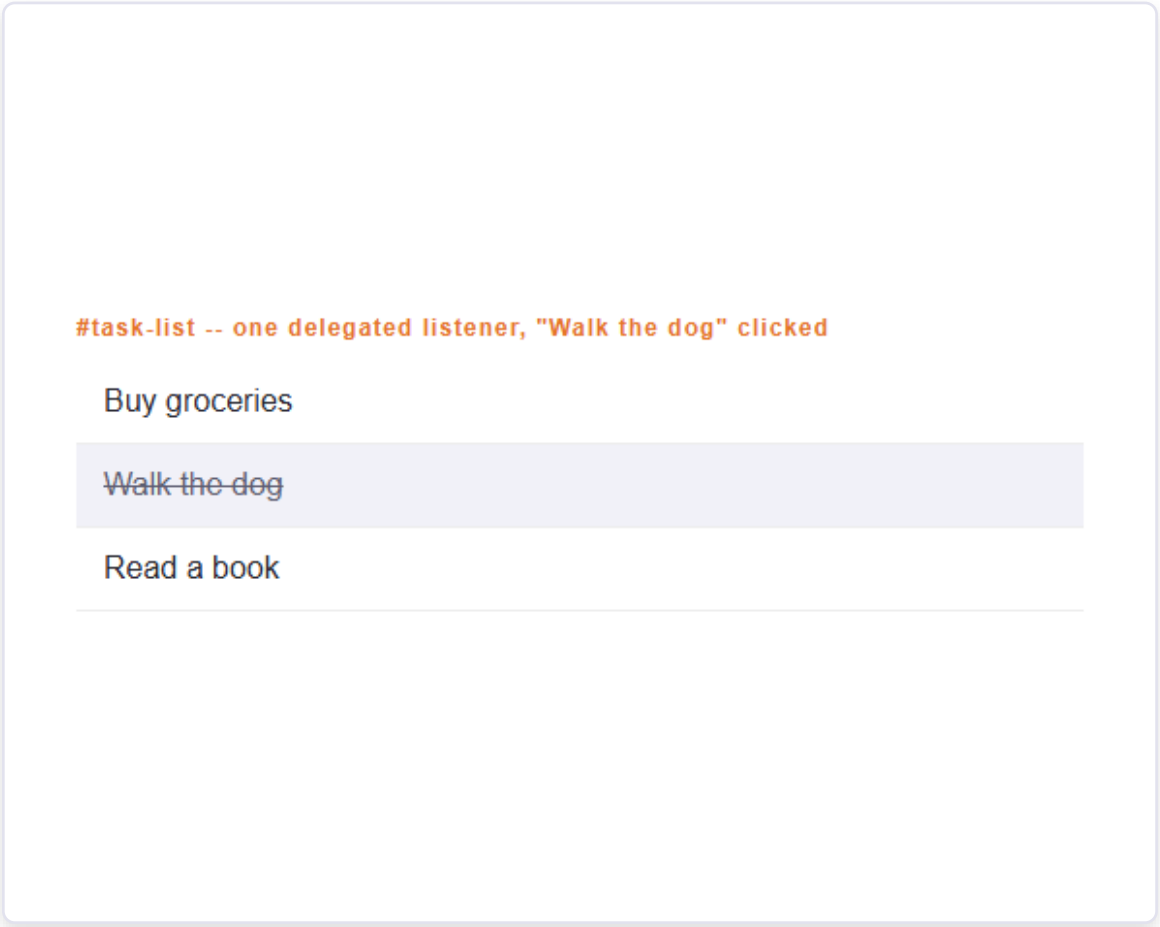
```
const list = document.querySelector("#task-list");

// One listener on the parent handles clicks on ANY current or future <li>
list.addEventListener("click", (event) => {
  if (event.target.tagName === "LI") {
    event.target.classList.toggle("done");
  }
});
```

event.target is the specific element clicked; event.currentTarget is the element the listener is attached to (list itself).

Event Delegation, Rendered

```
list.addEventListener("click",
  (event) => {
  if (event.target.tagName ===
      "LI") {
    event.target.classList
      .toggle("done");
  }
});
```



Why Event Delegation Matters

TIP

If you add 100 new <li> items after the page loads, a delegated listener on the parent <ul> automatically handles clicks on all of them -- no need to attach or re-attach 100 separate listeners.

Two Very Different Methods

```
document.querySelector("a.disabled-link").addEventListener("click", (event) => {
  event.preventDefault(); // the link will NOT navigate anywhere
});

document.querySelector(".dropdown-toggle").addEventListener("click", (event) => {
  event.stopPropagation(); // won't also trigger a parent's listener
});
```

preventDefault() stops the browser's default behavior. stopPropagation() stops the event from continuing to bubble/capture.

Don't Confuse the Two

WARNING

preventDefault() cancels what the browser would have done. stopPropagation() cancels the event from reaching other listeners further up (or down) the tree. A single handler can call both when needed.

KEY TAKEAWAYS

Lecture 13 in Seven Points

1. The DOM is a live, tree-shaped, in-memory representation of the page; document is your entry point into it.
2. getElementById selects by id; querySelector/querySelectorAll select by any CSS selector and are more flexible.
3. Prefer textContent over innerHTML unless you specifically need to insert HTML, to avoid XSS security risks.
4. classList.add/remove/toggle/contains is the preferred way to change appearance; reserve direct style for one-off values.
5. addEventListener registers handlers and supports multiple listeners per event; every handler receives an event object.
6. Events bubble up (and can capture down) the DOM tree -- this enables event delegation, one listener for many children.
7. preventDefault() stops the browser's default action; stopPropagation() stops the event reaching other listeners.