Welcome to lecture 15!

Agenda

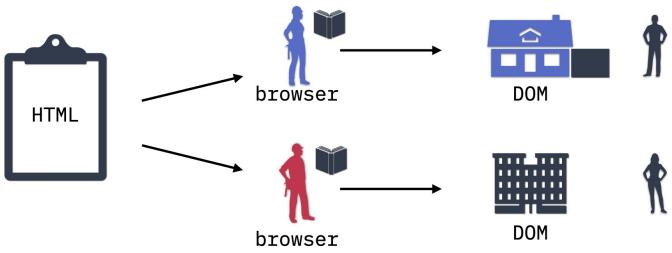
Session Objectives

- Define the DOM and its tree structure
- Explain why DOM manipulation is crucial
- Learn to select elements using various JavaScript methods
- Learn to modify elements (content, attributes, styles)
- Learn to create and add new elements
- Quiz

Introduction to DOM

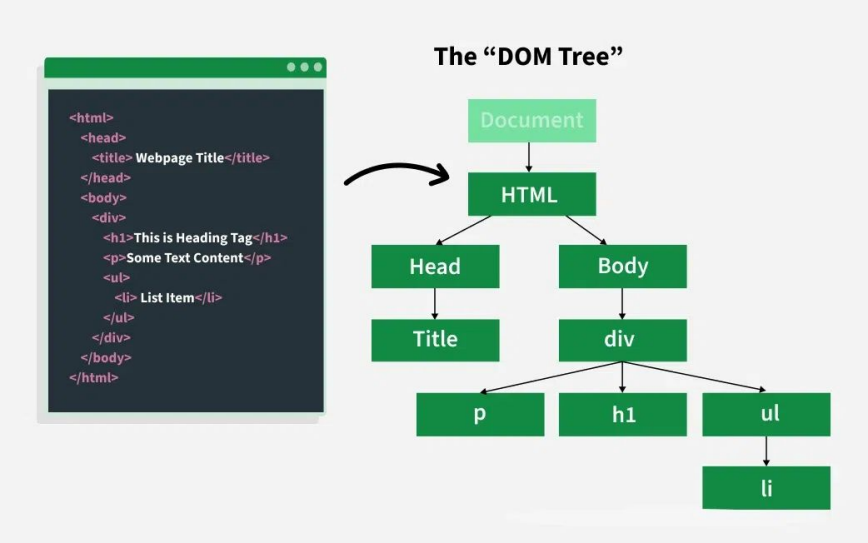
What is the DOM?

- What is DOM?
  - Document Object Model
  - It is a cross-platform, language-independent API that treats a document as a logical tree of nodes
- Think of it as a structured, live representation of your webpage
- Allows JavaScript to "see" and "talk to" your HTML



The DOM as a Tree Structure

- The DOM organizes HTML elements in a hierarchy, like a family tree
- <html> is the root
- <head> and <body> are children of <html>
- Elements like <h1>, <p>, are further descendants
- Each item in the tree is called a Node
    - Nodes form a node tree with root, parent, child, and sibling relationships

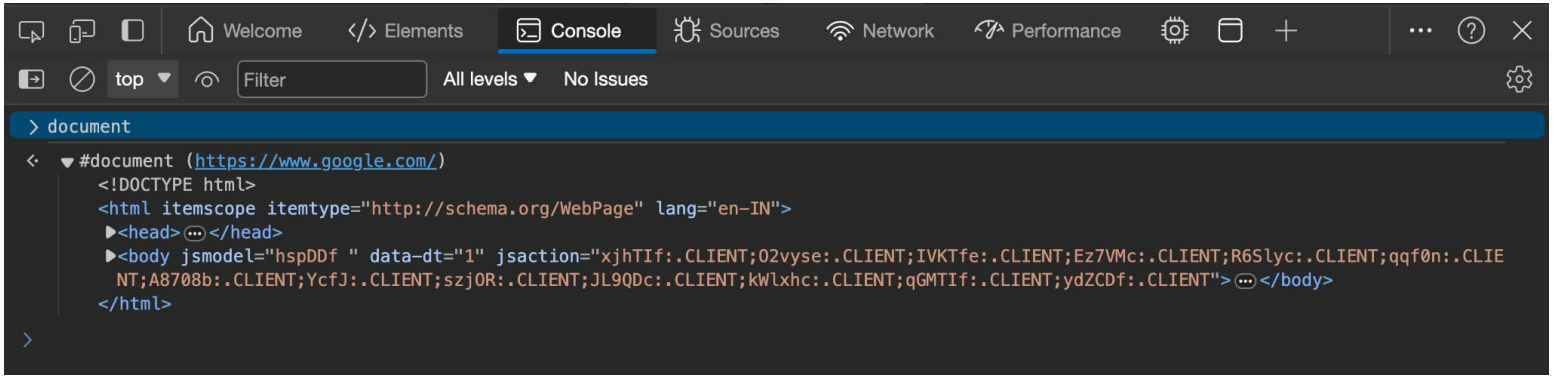


Why is the DOM Tree Important?

- JavaScript uses DOM methods to select and manipulate page elements dynamically
- Access
  - JavaScript can navigate this tree to find any element
- Modification
  - JavaScript can change elements (text, attributes, styles) without needing full page reloads
- Dynamics
  - JavaScript can add new branches (elements) or remove existing ones
- This is what makes web pages dynamic and interactive!

The document Object: Your Entry Point

- JavaScript provides a global object called document
- This document object represents the entire DOM of the current page.
- It's the starting point for almost all DOM interactions
- Example: document.getElementById(...) - a method of the document object



Selecting Elements

Selecting by ID: getElementById()

- Purpose: Finds a single element with a specific, unique id.
- Syntax: document.getElementById('yourElementId')
- Returns: The Element object, or null if not found
  - Element is a JS object representing an HTML element
- Why? IDs are unique – this is fast and precise

```
<!-- HTML -->
<div id="main-title">Hello</div>

// Javascript
const title = document.getElementById('main-title');
```

Selecting by Class Name: getElementsByClassName()

- Purpose: Finds all elements that share a specific class name
- Syntax: document.getElementsByClassName('className')
- Returns: An HTMLCollection (a live, array-like list) of elements
  - Represents a collection of Element nodes
    Updates automatically when elements matching the criteria are added or removed from the document
- Useful for targeting groups of similar items

```
<!-- HTML -->
<p class="highlight">Text 1</p> <span class="highlight">Text 2</span>

// Javascript
const highlights = document.getElementsByClassName('highlight');
```

Selecting by Tag Name: getElementsByTagName()

- Purpose: Finds all elements with a specific HTML tag.
- Syntax: document.getElementsByTagName('tagName') (e.g., 'p', 'li', 'div')
- Returns: An HTMLCollection (also live)
- Why? To apply changes to all instances of a certain element type

```
<!-- HTML -->
<li>Item 1</li> <li>Item 2</li>

// Javascript
const listItems = document.getElementsByTagName(`li`);
```

Modern Selectors: querySelector()

- Purpose: Finds the first element that matches a CSS selector
- Syntax: document.querySelector('yourCssSelector')
- Examples:
  - querySelector('#myId')
  - querySelector('.myClass')
  - querySelector('div p')
- Returns: The first matching Element, or null
- Why? Extremely versatile – uses the power of CSS selectors you already know!

```
<!-- HTML -->
<div id="content">
        <p class="info">First para.</p>
        <p>Second para.</p>
</div>

// Javascript
const listItems = document.querySelector('.info');
```

Modern Selectors: querySelectorAll()

- Purpose: Finds all elements that match a CSS selector.
- Syntax: document.querySelectorAll('yourCssSelector')
- Returns: A static NodeList (array-like list) of elements
- Why? Same versatility as querySelector, but for multiple elements

```
<!-- HTML -->
<p class="note">Note 1</p> <span class="note">Note 2</span>

// Javascript
const allNotes = document.querySelectorAll('.note');
```

Selector Showdown: Key Differences

- getElementById: Fastest for unique IDs. Returns Element or null.
- getElementsByClassName/TagName: Return live HTMLCollection
- querySelector: CSS selectors, returns first Element or null
- querySelectorAll: CSS selectors, returns static NodeList
- Performance
  - getElementById is very fast
  - querySelector(All) are powerful and generally fast enough for most tasks
  - Don't over-optimize early, though

Let’s Code

Manipulating Elements

Manipulating Elements

- Once you've selected an element, you can change it!
- Change content (text, HTML)
- Change attributes (like src of an image, href of a link)
- Change styles (colors, sizes, visibility)

```
<!-- HTML -->
<p class="note">Note 1</p> <span class="note">Note 2</span>

// Javascript
const allNotes = document.querySelectorAll('.note');
allNotes.forEach(p => {
  p.style.color = 'blue';
});
```

Changing Content: textContent vs. innerHTML

- There are two ways to change textual content
- element.textContent
  - Gets or sets the plain text content of an element and its children
  - HTML tags are treated as literal text
  - Safer for setting text
- element.innerHTML
  - Gets or sets the HTML content (markup included) within an element
  - Can be used to insert new HTML structures
  - Caution: Security risk if setting with untrusted user input

Working with Attributes

- element.getAttribute('attributeName'): Reads an attribute's value.
- element.setAttribute('attributeName', 'newValue'): Sets/updates an attribute.
- element.removeAttribute('attributeName'): Removes an attribute.
- Direct properties: Many attributes can also be accessed as properties (e.g., img.src, a.href, input.value).

```
<!-- HTML -->
<img id="myImage" src="old.jpg" alt="Old">

// Javascript
const img = document.getElementById("myImage");
img.setAttribute("src", "new.jpg");
img.alt = "New Image";
```

Styling Elements: element.style

- Directly access and modify an element's inline CSS styles
- Syntax: element.style.propertyName = 'value';
- CSS properties with hyphens become camelCase (e.g., background-color -> backgroundColor).
- Example
  - myElement.style.color = 'blue';
  - myElement.style.fontSize = '20px';

```html
<!-- HTML -->
<p id="message">This is a message.</p>
<button onclick="changeStyle()">Change Style</button>
```

```javascript
// Javascript
const msg = document.getElementById('message');
msg.style.color = 'blue';
msg.style.fontSize = '24px';
msg.style.backgroundColor = '#f0f8ff';
msg.style.padding = '10px';
msg.style.borderRadius = '5px';
```

Styling Elements: element.classList

- Better for managing styles by adding/removing CSS classes.
- Keeps styles in your CSS file – cleaner code!
- Available operations
  - Add: element.classList.add('className')
  - Remove: element.classList.remove('className')
  - Toggle: element.classList.toggle('className')
    - adds if not present, removes if present
  - Check if exists: element.classList.contains('className')

```
/*CSS */
.active { background-color: yellow; font-weight: bold; }

// Javascript
myElement.classList.add('active');
```

Let’s Code

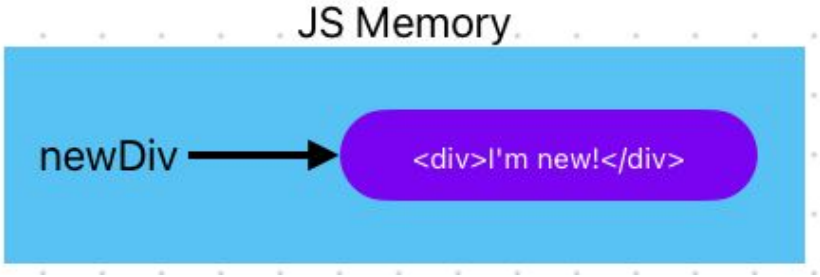
Exercise: Movie Spoiler!

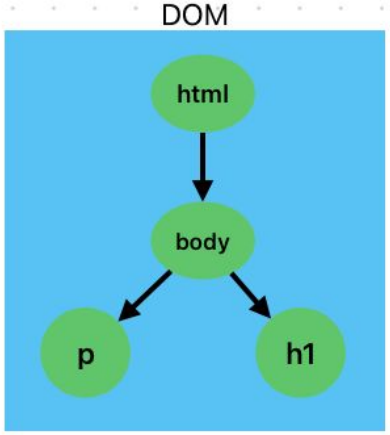
- Create a h1 tag for a movie name
- Create a paragraph tag and make it a spoiler for a movie of your choice!
    - By default: hide the paragraph
- Create two separate buttons to “hide spoiler” and “show spoiler”
- Alternatively, use a single button to hide/show the paragraph tag

Creating and Adding Elements

Creating New Elements: document.createElement()

- Purpose: Creates a new HTML element in memory (not yet on the page).
- Syntax
  - const newElement = document.createElement('tagName');
- Example
  - const newP = document.createElement('p')
- The new element is initially empty and detached from the DOM.





Adding Elements: parentNode.appendChild()

- Purpose: Adds a child node as the last child of parentNode.
- Syntax: parentElement.appendChild(newlyCreatedElement);
- childNode must be a Node object (e.g., usually from createElement).
- Returns the appended node

```
<!-- HTML -->
<div id="container"></div>

// Javascript
const div = document.getElementById('container');
const newP = document.createElement('p');
newP.textContent = 'Hello!';
div.appendChild(newP);
```

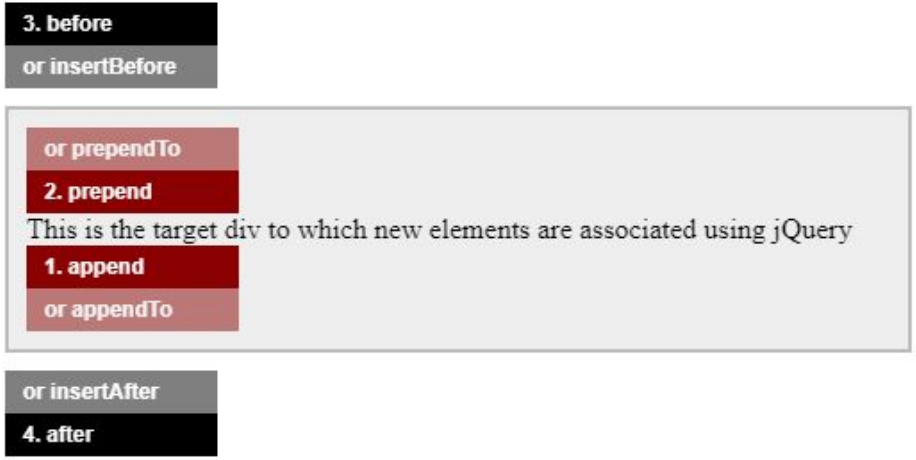
Adding Elements: parentElement.append()

- More flexible way to add content.
- Syntax: parentElement.append(node1, "text string", node2, ...);
- Can append multiple nodes and/or plain text strings at once.
- Text strings are inserted as Text nodes.
- Does not return a value (returns undefined)

```
// Javascript
// Assume “anotherNewElement” has been created
div.append(newP, " Some extra text.", anotherNewElement);
```

Other Ways to Add: insertBefore() & prepend()

- parentNode.insertBefore(newNode, referenceNode):
  - Inserts newNode before an existing referenceNode (which must be a child of parentNode).
- element.prepend(nodeOrString, ...):
  - Similar to append, but adds content before the first child of element.



Removing Elements

- parentNode.removeChild(childNode):
  - Removes childNode from parentNode.
  - Returns the removed node (can be useful).
- element.remove():
  - Call directly on the element you want to remove.
  - No need for parent reference
    Simpler & Modern

```
// Javascript
const item = document.getElementById('item-to-delete');
item.remove();
```

Let’s Code

Winding Down...

Practical Considerations

- Performance
  - DOM manipulation can be "expensive" (slows browser).
  - Try to minimize frequent, small changes
  - Cache selectors: If using an element multiple times, store it in a variable
- Readability
  - Use clear variable names for selected elements
  - Comment your code
- Libraries & Frameworks
  - Tools like React, Vue, Angular are built on top of DOM
  - They abstract away some direct DOM manipulation
  - Understanding these core DOM principles is essential to building large scale web applications

Quiz!

Question 1

- What is the primary purpose of the DOM?
  - A) To store JavaScript functions
  - B) To define the styling of a webpage
  - C) To execute server-side code
  - D) To provide a structured representation of HTML documents that JavaScript can interact with

Correct Answer: D

The DOM is a model of the HTML page, enabling JavaScript to understand and change its structure and content.

Question 2

- Which JavaScript method is used to select a single HTML element by its unique ID?
  - A) document.querySelector('.myId')
  - B) document.getElementsByClassName('myId')
  - C) document.getElementById('myId')
  - D) document.getElementByTag('myId')

Correct Answer: C

Question 3

- If you want to change the text inside a <p> tag to "Hello World!" without adding any HTML tags, which property is best to use?
  - A) element.innerHTML = "Hello World!"
  - B) element.textContent = "Hello World!"
  - C) element.outerHTML = "Hello World!"
  - D) element.style.text = "Hello World!"

Correct Answer: B

Question 4

- Given the HTML & JS snippet, what will be the outcome of items.length?
  - A) 2
  - B) 1
  - C) 0
  - D) undefined

Correct Answer: C

getElementsByClassName returns a live HTMLCollection

When the innerHTML of the parent is cleared, the elements are removed from the DOM

So, the live collections' items update to reflect this, becoming empty.

```
<!-- HTML -->
<div id="parent">
        <p class="child">First</p>
        <p class="child">Second</p>
</div>

// JS
const items = document
                        .getElementById('parent')
                        .getElementsByClassName('child');
document.getElementById('parent').innerHTML = "";
```

Question 5

- Which of the following will successfully add the text "Hello" and then the text "World" as two separate text nodes inside myDiv?
  - A) myDiv.textContent = "Hello";
    myDiv.textContent += "World";
  - B) myDiv.innerHTML = "Hello";
    myDiv.innerHTML += "World";
  - C) myDiv.append("Hello", "World");
  - D) myDiv.appendChild(document.createTextNode("Hello"));
    myDiv.appendChild(document.createTextNode("World"));

```
// JS
const myDiv = document.createElement('div');
```

Correct Answer: C or D

- append() creates separate text nodes for each string argument. Recommend approach
- Option D explicitly creates and appends text nodes. Traditional approach

Question 6

- If myElement.setAttribute('data-info', 'initial'); is executed, and then later myElement.dataset.info = 'updated'; is run, what will myElement.getAttribute('data-info') return?
  - A) initial
  - B) updated
  - C) null
  - D) It will throw an error

Correct Answer: B

The dataset property provides a convenient way to access data-* attributes. Modifying myElement.dataset.info *directly* reflects the change in the underlying data-info attribute

Question 7

- You want to add a new New Item at the very beginning of an existing <ul> element stored in the variable myList. Which of the following is the most direct way to achieve this?
  - A) myList.appendChild(newLi); (where newLi is the created list item)
  - B) myList.append(newLi);
  - C) myList.insertAfter(newLi, myList.firstChild);
  - D) myList.prepend(newLi);

Answer: D)
appendChild and append - adds to the end
insertAfter - not a method

Question 8

- What is a key difference between an HTMLCollection (e.g., returned by getElementsByClassName) and a NodeList (e.g., returned by querySelectorAll) regarding DOM updates?
  - A) NodeList is always live, reflecting real-time DOM changes; HTMLCollection is static
  - B) HTMLCollection is live, reflecting real-time DOM changes; NodeList (from querySelectorAll) is typically static.
  - C) Both are always live.
  - D) Both are always static.

Answer: B
HTMLCollection is live, reflecting real-time DOM changes; NodeList (from querySelectorAll) is typically static

Homework: Treasure Hunt!

- For the base treasure hunt code provided in the lecture-17 Github repository, you need to write the JS functions to complete the game
- Functions to complete:
  - findGoldCoin
  - findAllGems
  - findAncientScroll
  - findAllSecretNotes
  - findHiddenCave
- Submission Link