JavaScript DOM & Event Handling

3rd semester @ Erhvervsakademi København

GitHub

Goals for this session

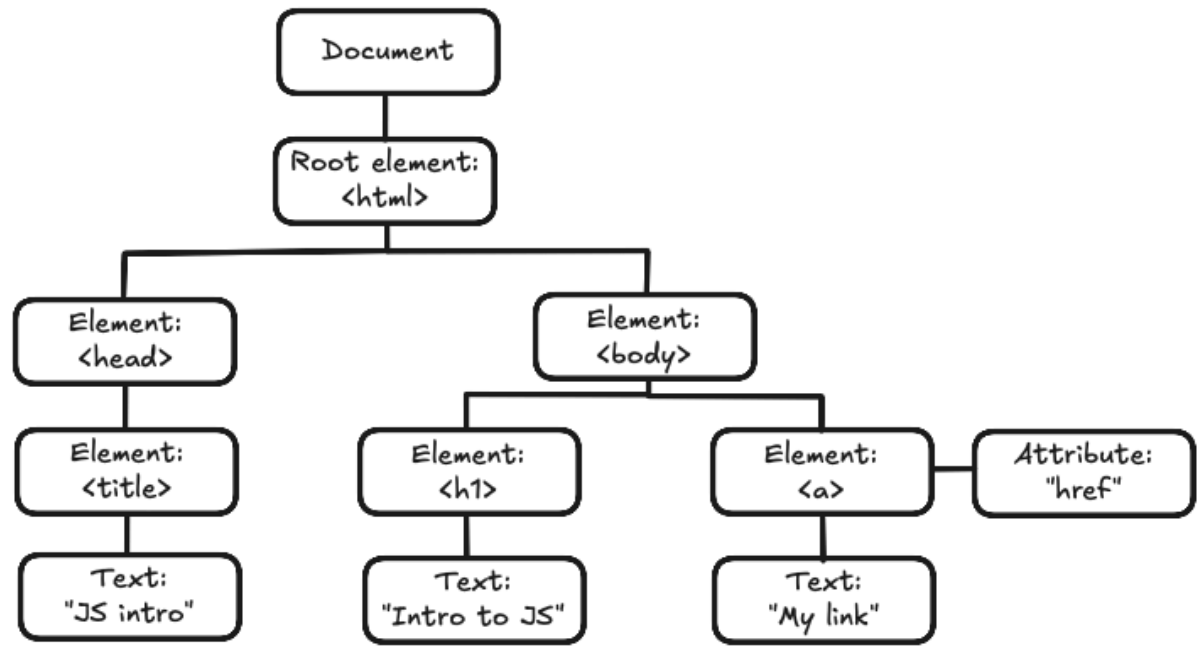
- Understand what the DOM is and how it represents a web page.
- Learn how to access and manipulate DOM elements using JavaScript.
- Handle user events and form submissions to create interactive web pages.

What is the DOM?

> The Document Object Model (DOM) is a programming interface for HTML and XML documents. It represents the structure of a document as a tree of objects, allowing to manipulate the content, structure, and style of web pages dynamically using JavaScript.

```html
<!DOCTYPE html>
<html>
<head>
    <title>JS intro</title>
</head>
<body>
    <h1>Intro to JS</h1>
    <a href="https://www.example.com">My link</a>
</body>
</html>
```

Document Object Model



What can we do with the DOM?

- Access and modify HTML elements and their attributes
- Change the content of elements
- Add or remove elements from the page
- Change the style of elements
- Handle events (e.g., clicks, form submissions)

Traversing the DOM

> The DOM is a tree structure, and you can navigate through it using various properties and methods.
> **Common ways to traverse the DOM elements:**

- `parentElement` : Access the parent element.
- `children` : Access the child elements.
- `nextElementSibling` : Access the next sibling element.
- `previousElementSibling` : Access the previous sibling element.
- `firstElementChild` : Access the first child element.
- `lastElementChild` : Access the last child element.

> These properties returns **HTML element**, ignoring text nodes.

Traversing the DOM (continued)

Example:

```
const child = document.children[0];
const firstChild = child.firstElementChild;
const lastChild = child.lastElementChild;
const nextSibling = firstChild.nextElementSibling;
const previousSibling = lastChild.previousElementSibling;
```

Each is an object representing a DOM element.

Exercise 1 - DOM Traversal

> Traverse the DOM and find the "Target element" paragraph. Log the element to the console.

Accessing Elements

> To access elements in the DOM, we can use various methods provided by the `document` object.

```
// Accessing an element by ID
const element = document.getElementById('myElement');

// Accessing elements by class name
const elements = document.getElementsByClassName('myClass');

// Accessing elements using querySelector
const firstElement = document.querySelector('.myClass');
const allElements = document.querySelectorAll('.myClass');
```

Document object

> The `document` object is the root of the DOM tree and provides methods to access and manipulate elements in the web page.

```
// Accessing the document object
console.log(document);

// Accessing the title of the document
console.log(document.title);

// Changing the title of the document
document.title = "New Title";
```

The `document` object also provides methods for creating new elements, handling events, and more.

Retrieving DOM Elements

> Instead of traversing the DOM manually, you often want to select specific elements directly.

Common ways to select elements:

- `getElementById('id')` : Selects a single element by its unique ID.
- `getElementsByClassName('class')` : Selects all elements with a given class (returns a collection).
- `getElementsByTagName('tag')` : Selects all elements with a given tag name.
- **`querySelector('selector')`** : Selects the first element that matches a CSS selector.
- **`querySelectorAll('selector')`** : Selects all elements that match a CSS selector

Retrieving DOM Elements (continued)

Using `querySelector` and `querySelectorAll` is often the most flexible way to select elements, and uses CSS selectors:

- `#myId` for ID
- `.myClass` for class
- `div` for tag name

Example:

```
const heading = document.querySelector('#someId'); // Select by ID
const items = document.querySelectorAll('.item'); // Select all by class
const allButtons = document.querySelectorAll('button'); // Select all buttons
```

Tip: Check if your selection worked (it may return `null` if not found).

Useful properties of selected elements

- `.textContent` : Get or set the text inside an element.
- `.innerHTML` : Get or set the HTML content inside an element.
- `.classList` : Access the list of classes on an element (useful for adding/removing classes).
- `.value` : Get or set the value of form elements (like input, textarea).
- `.id` : Get or set the ID of an element.
- `.style` : Access or change the inline styles of an element.

Getting and setting content of DOM elements

```
const p = document.querySelector('p');
console.log(p.textContent); // Print text
p.textContent = 'New text content'; // Set text
```

Template used in exercises

```
window.addEventListener('DOMContentLoaded', initApp);

function initApp() {
        // Your code here
}
```

`initApp` runs when the page is fully loaded, and serves as an entrypoint.

Handling Events

> Web pages become interactive by responding to user actions, called **events** (like clicks, submitting forms, or moving the mouse).

How to handle events:

- Use `addEventListener` to tell the browser what code to run when an event happens.

Example:

```
const button = document.querySelector('#myButton');
button.addEventListener('click', () => {
  alert('Button was clicked!');
});
```

Common events: `click`, `submit`, `mouseover`

Exercise 2

> Add a click event to a button. When the button is clicked, take all messages and concatenate them into a single string, then display that string in the output div.

Toggling CSS Classes

> CSS classes control how elements look. You can add, remove, or toggle classes with JavaScript to change styles dynamically.

The `classList` property:

- `element.classList.add('className')` : Add a class
- `element.classList.remove('className')` : Remove a class
- `element.classList.toggle('className')` : Add if missing, remove if present

Example:

```
const box = document.querySelector('#box');
box.classList.toggle('highlight');
```

Exercise 3

> Toggle a CSS class on an element when a button is clicked. Try changing the color or size of the element.

Adding & Removing Elements

You can create new elements and add them to the page, or remove existing ones.

How to add an element:

1. Use `document.createElement('tag')` to make a new element.
2. Set its content or attributes.
3. Use `parent.appendChild(newElement)` to add it to the page.

How to remove an element:

- Call `.remove()` on the element you want to delete.

Example:

```
const ul = document.querySelector('#myList');
const li = document.createElement('li');
li.textContent = 'New item';
ul.appendChild(li); // Add
li.remove(); // Remove
```

Exercise 4

> Create a full DOM structure using only JavaScript.

Exercise 5

> Add and remove list items using buttons.

Data Attributes

Sometimes you need to store extra information on an element that isn't part of the standard HTML attributes. **Data attributes** let you do this.

How to use:

- Add a custom attribute like `data-id="1"` in your HTML.
- Access it in JavaScript with `element.dataset.id` or `element.getAttribute('data-id')`.

Why use data attributes?

- Store state or extra info for scripts without affecting the appearance or meaning of the HTML.

Data Attributes - example

Example (removing a specific item):

```
<ul>
    <li data-id="1">Some text</li>
</ul>
```

```
const li = document.querySelector('li[data-id="1"]');
console.log(li.dataset.id); // "1
li.remove();
```

Exercise 6

> Toggle a data attribute on a button and display its value on the page.

Handling Forms

> Forms let users enter data. You can use JavaScript to process this data and update the page.

How to handle a form:

1. Listen for the `submit` event on the form.
2. Prevent the default page reload with `event.preventDefault()` (what happens by default?)
3. Read the values from the form fields.
4. Do something with the data (like adding it to a list or sending it to a server).

Event object

> When an event happens, the event handler receives an event object that contains information about the event (like which element was clicked, what type of event it is, etc.).

Handling Forms - example

```
<form id="myForm">
  <input type="text" name="name" placeholder="Enter your name">
  <button type="submit">Submit</button>
</form>
<script>
  const form = document.querySelector('#myForm');
  form.addEventListener('submit', handleSubmit);

  function handleSubmit(event) {
    event.preventDefault(); // Prevent page reload
    const target = event.target;
    const form = new FormData(target);
    const name = form.get('name'); // Assuming there's an input with name="name"
    console.log(name); // Do something with the name
  }
</script>
```

Forms

- Use `event.preventDefault()` to prevent the default behaviour of a form submission (page reload).
- Use `const form = new FormData(event.target)` to get form data.
- Access the form data with `form.get('inputName')`.
    - `inputName` must match the `name` attribute of the input field.

Exercise 7

> Handle a form submission and display the entered data in the DOM.

Exercise 8 (extra)

> Implement a simple game using DOM manipulation and event handlers.