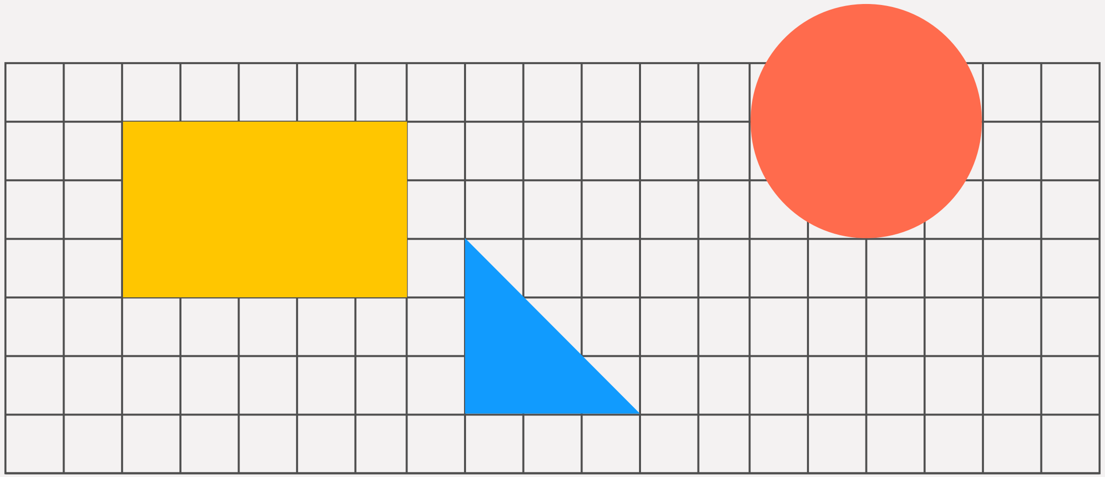
Structure Meets Behaviour!

Ali Abrishami



In This Lecture You Will...

a

Learn What DOM is

b

Figure out a way to make your pages interactive

c

Understand async programming usages in web

REMINDER: The Three Layers of a Website

Structure

What the **content** is

Style

How it **looks**

Behaviour

How it **reacts**

REMINDER: Objects

- Objects group related data and behavior.
- Made of key-value pairs.
- Values can be data (properties) or functions (methods).
- JSON (JavaScript Object Notation) is a common way to store & share object data.

Technically speaking, in JavaScript, almost everything is an object.

REMINDER: `this` Keyword

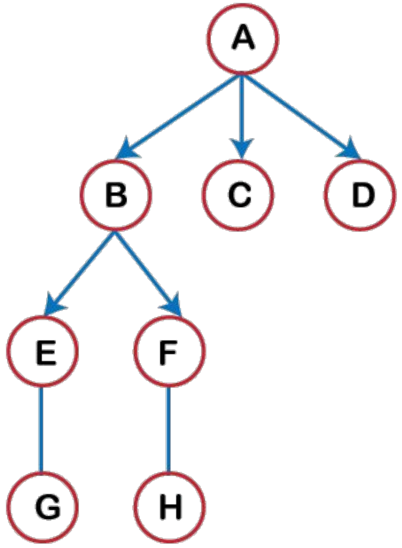
- Refers to the object that is calling the function.
- In objects, `this` points to the object itself.
- In classes, `this` points to the created instance.
- In arrow functions, `this` is inherited from the outer scope.
- The value of this depends on how a function is called, not where it’s defined.

REMINDER: Async JS

- Callbacks: Pass a function to run later.
- Promises: Represent a future value with states.
- `async`/`await`: Built on Promises, but cleaner syntax.
- Callbacks: can cause nested “callback hell.”
- Promises: chain with `.then()` / `.catch()`.
- `async`/`await`: looks synchronous, easier to read.
- Error handling: callbacks → manual, promises → `.catch()`, `async` → `try...catch`.

A Deeper View on Markup Languages

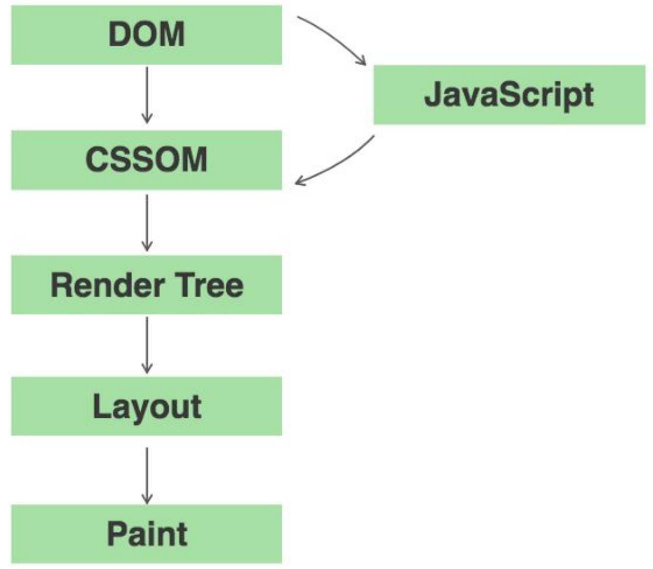
- Markup languages describe structure using tags
- Examples: HTML, XML, Markdown
- Documents are arranged in a **hierarchical tree**
- Parent and child elements show how content nests
- This tree model is the foundation of the **DOM**



Source: Board Infinity

How To Load A Webpage

- Load: Fetch all necessary files
- Parse: Read and interpret HTML and CSS
- Construct: Make DOM tree
- Make Render Tree: Combine DOM and CSS
- Layout (Reflow): Calculate sizes and positions.
- Paint: Draw pixels of each element.
- Composite: Combine all painted layers.

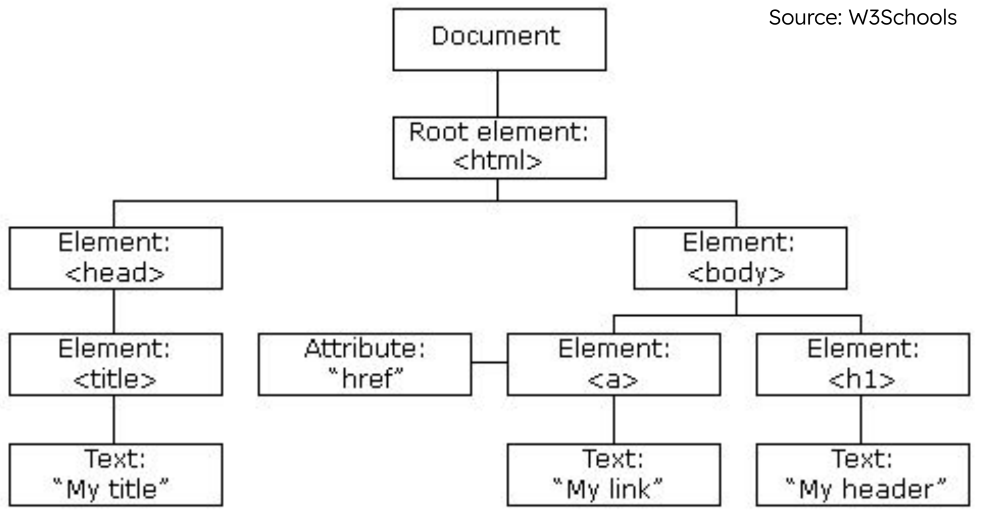


Source: freeCodeCamp

From Markup To DOM

- Browser parses HTML into a tree of nodes
- Each tag becomes an element node in the tree
- Text inside tags becomes text nodes
- Attributes become node properties
- This DOM tree is what JavaScript can **access & modify**

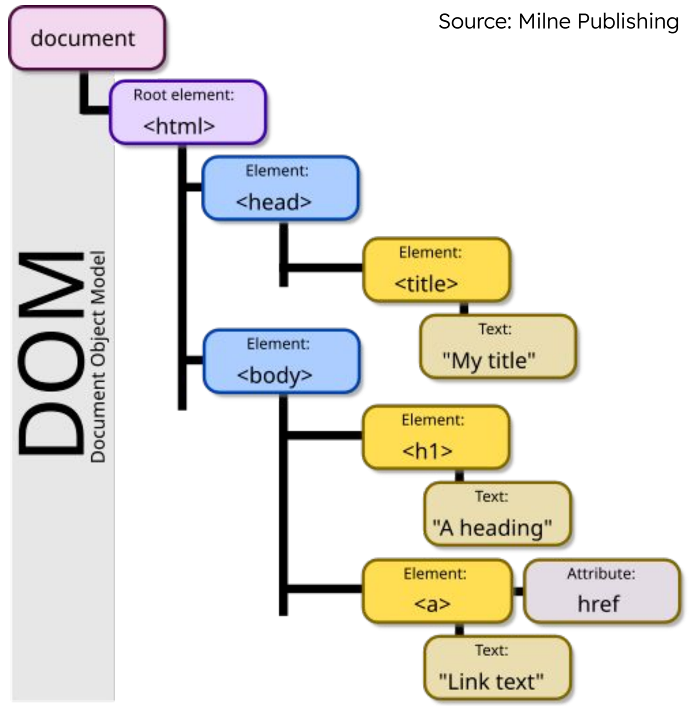
Source: W3Schools



Look how the DOM tree organizes the elements.

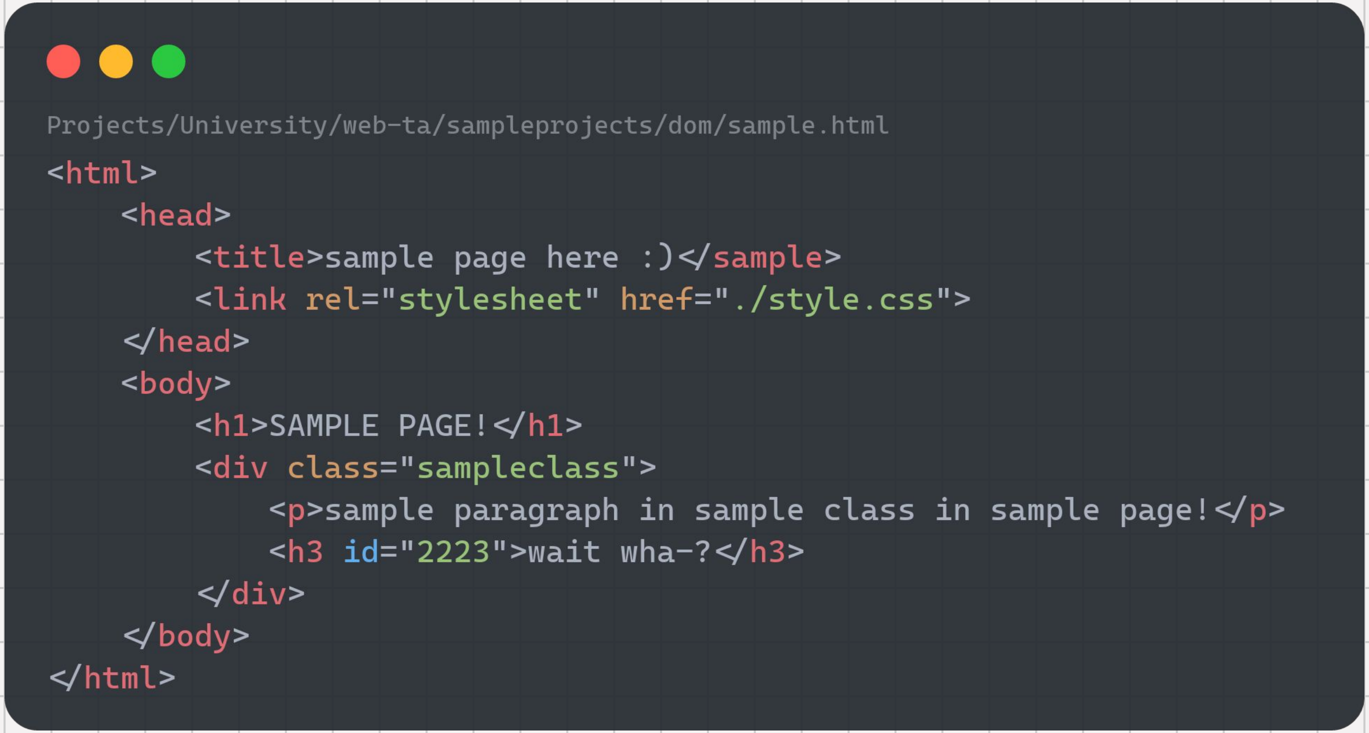
What is the DOM?

- DOM is short for **Document Object Model**
- A tree-based representation of a webpage
- Connects HTML structure with JavaScript
- Lets programs read, modify, add, or remove elements
- Acts as a **bridge** between content and interactivity



Class Activity Time!

Draw the DOM tree for this HTML code

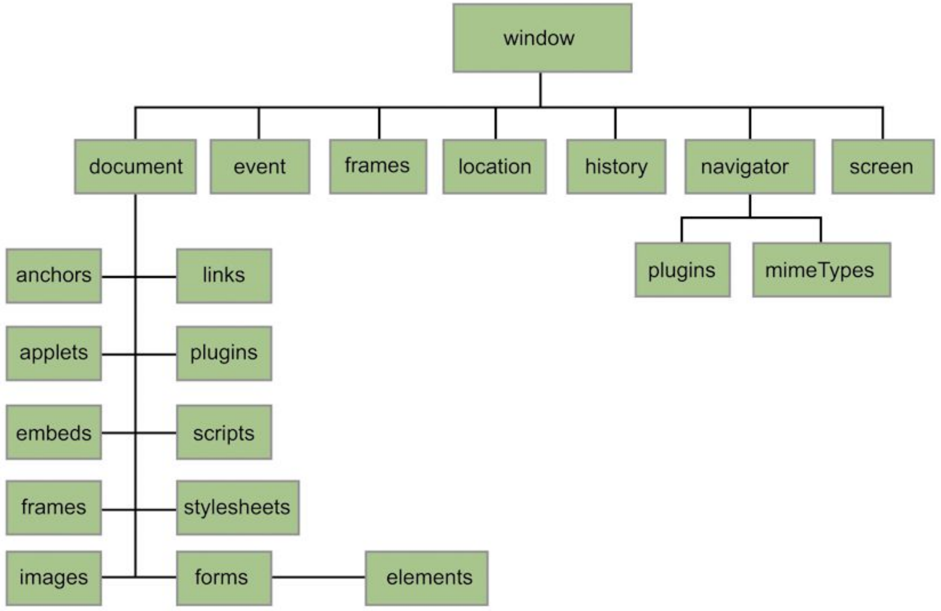


Projects/University/web-ta/sampleprojects/dom/sample.html

```html
<html>
    <head>
        <title>sample page here :)</sample>
        <link rel="stylesheet" href="./style.css">
    </head>
    <body>
        <h1>SAMPLE PAGE!</h1>
        <div class="sampleclass">
            <p>sample paragraph in sample class in sample page!</p>
            <h3 id="2223">wait wha-?</h3>
        </div>
    </body>
</html>
```

DOM Hierarchy In JS

- `window`: the top-level browser object (global context, remember?)
- `document`: represents the web page inside the window
- **Elements:** tags (`<div>`, `<p>`, `<h1>`) as objects in the DOM tree
- **Attributes & Text:** properties of element nodes
- **Events:** methods & listeners that connect JS to user actions



Source: CIS

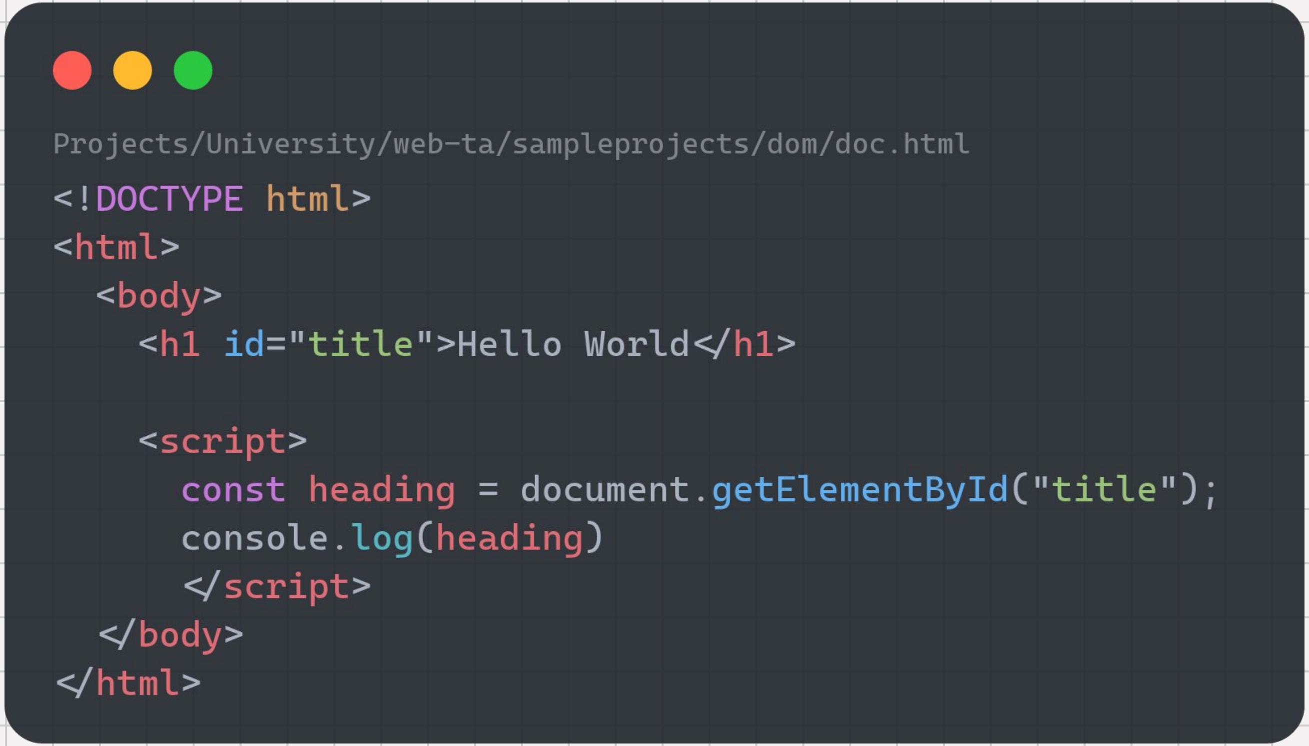
The `document` Object

- Represents the entire web page in the browser
- Acts as the entry point to the DOM tree
- Provides methods to find elements (`getElementById`, `querySelector`)
- Lets JS create, remove, or update nodes

Projects/University/web-ta/sampleprojects/dom/doc.html

```html
<!DOCTYPE html>
<html>
  <body>
    <h1 id="title">Hello World</h1>
    <button id="btn">Change Text</button>

    <script>
      // Access an element
      const heading = document.getElementById("title");

      // Add event to button
      document.getElementById("btn").addEventListener("click", () => {
        heading.textContent = "Text Changed via document!";
      });
    </script>
  </body>
</html>
```

The document Object (cont.)



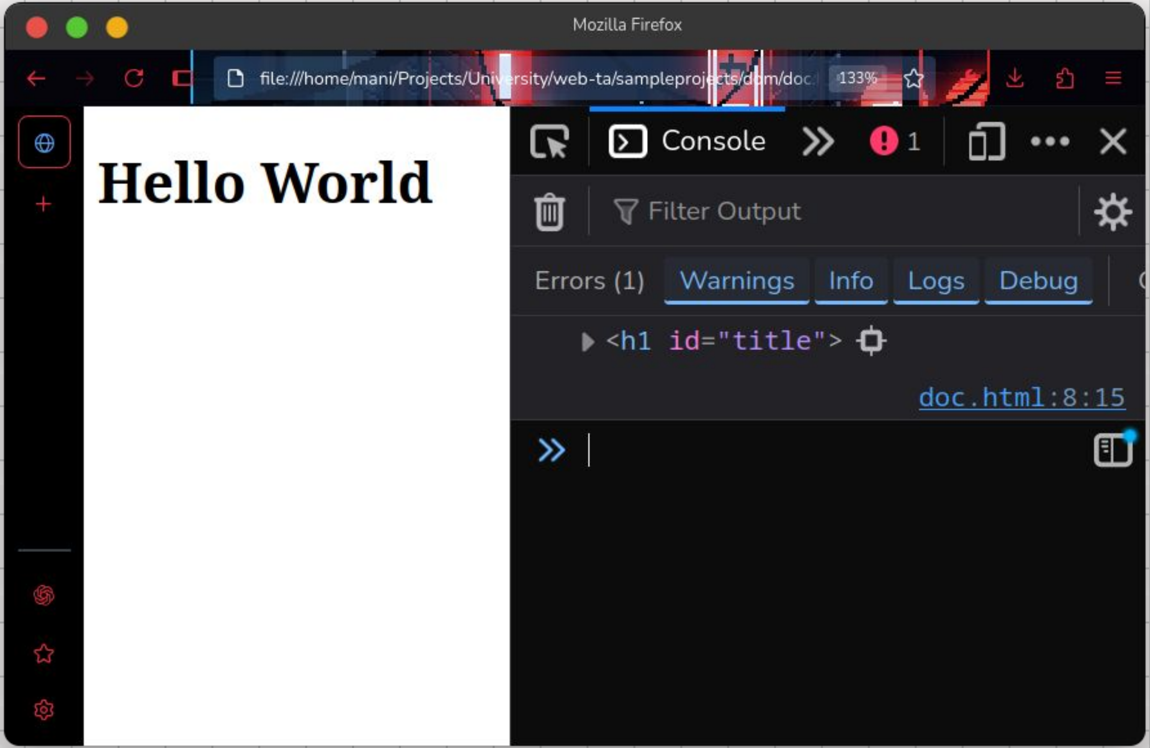
```
Projects/University/web-ta/sampleprojects/dom/doc.html
<!DOCTYPE html>
<html>
  <body>
    <h1 id="title">Hello World</h1>

    <script>
      const heading = document.getElementById("title");
      console.log(heading)
      </script>
  </body>
</html>
```

Let’s run this code

It has JavaScript embedded inside HTML (remember?)

Can you predict what’s going to be printed in the console?

The `document` Object (cont.)



Hello World

```
<h1 id="title">
doc.html:8:15
```

The element itself has been printed in the console, which makes sense, since we logged the element to the console.

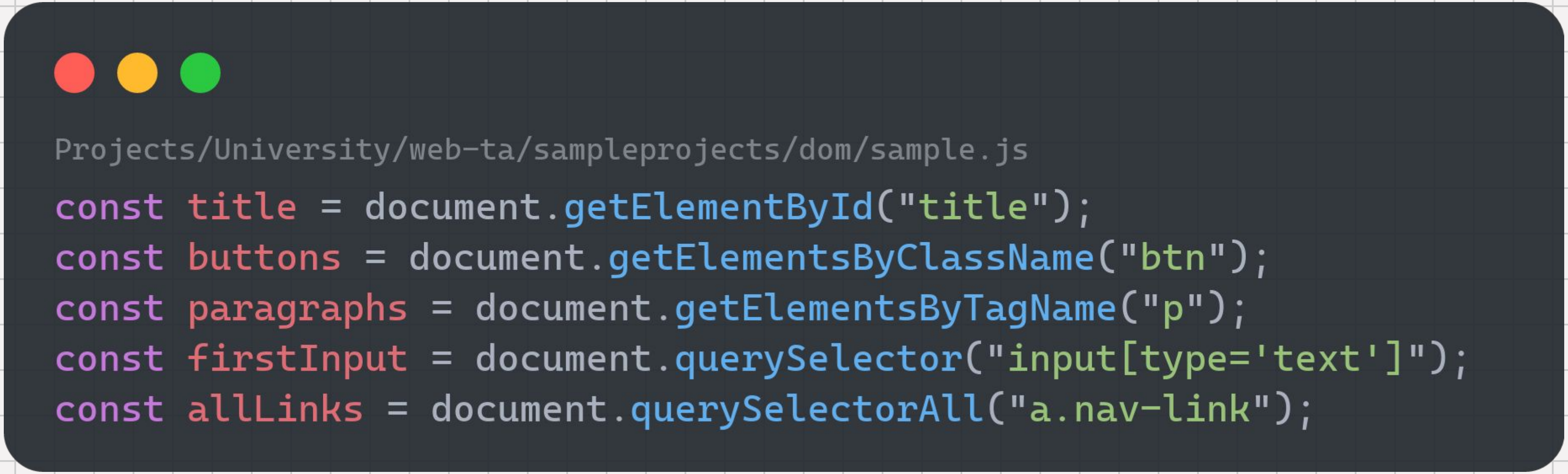
Now, let’s ask ourselves, what other cool things can `document` do?!

Querying DOM Elements

- Use JavaScript to find elements on the page
- `getElementById("id")`: finds one element by its ID
- `getElementsByClassName("class")`: finds all elements with that class
- `getElementsByTagName("tag")`: finds all elements with that tag
- `querySelector("selector")`: finds the first match using CSS selectors
- `querySelectorAll("selector")`: finds all matches using CSS selectors

Class Activity Time!

What does each line of this JavaScript code snippet do?

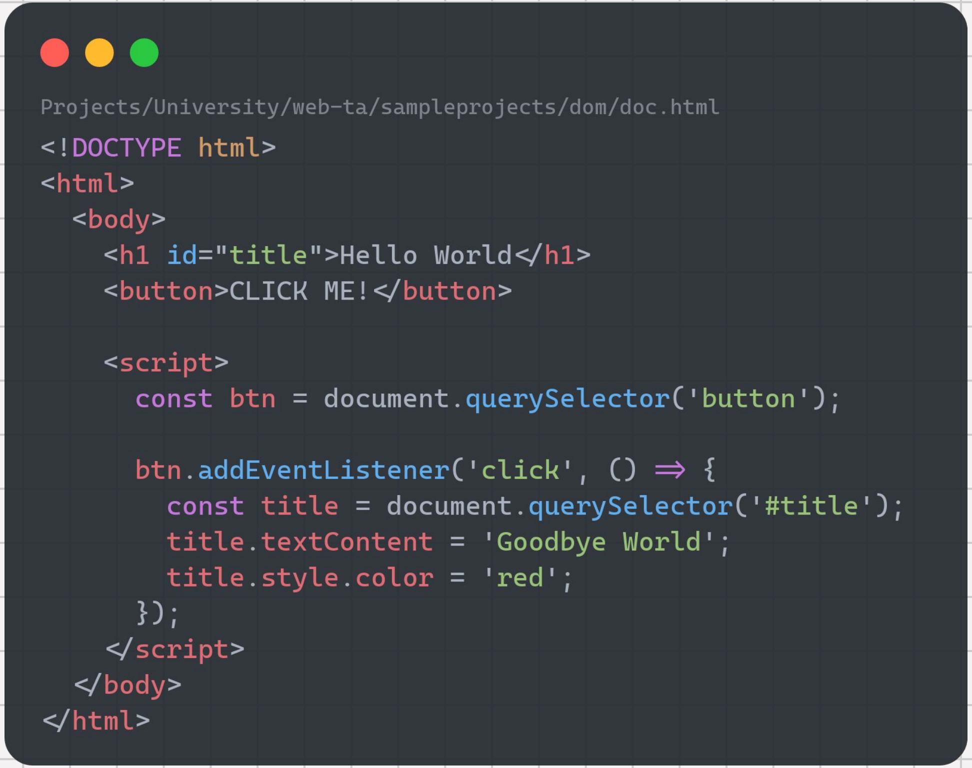


```
Projects/University/web-ta/sampleprojects/dom/sample.js
const title = document.getElementById("title");
const buttons = document.getElementsByClassName("btn");
const paragraphs = document.getElementsByTagName("p");
const firstInput = document.querySelector("input[type='text']");
const allLinks = document.querySelectorAll("a.nav-link");
```

Events

- Events let webpages **react** to user actions
- Common events: `click`, `input`, `submit`, `keydown`, `load`, etc.
- Handlers can be set using HTML attributes or JavaScript
- `addEventListener` is the modern way to attach events
- Multiple handlers can be added to the same event
- Events enable interactive and dynamic user experiences

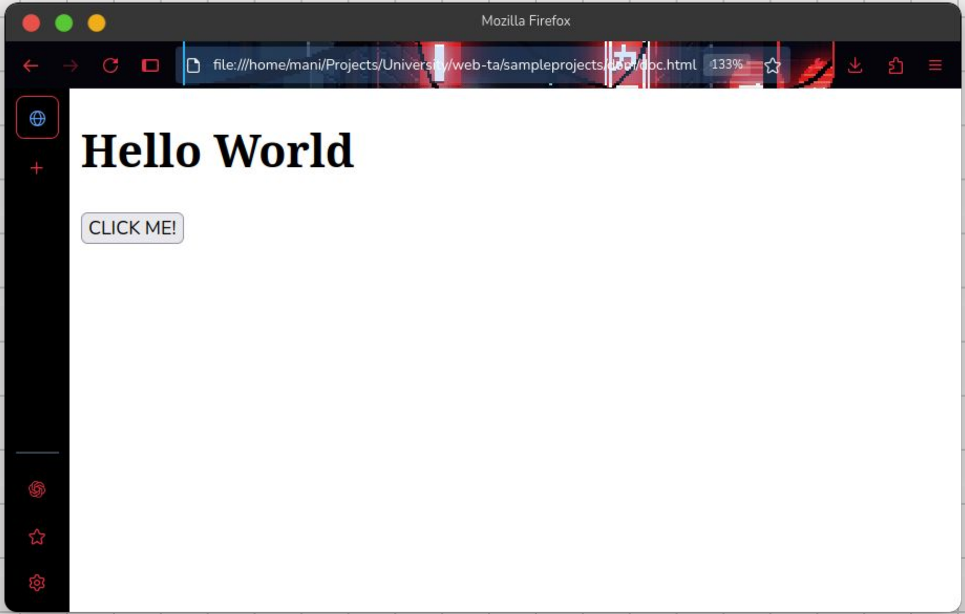
Example: Button Action



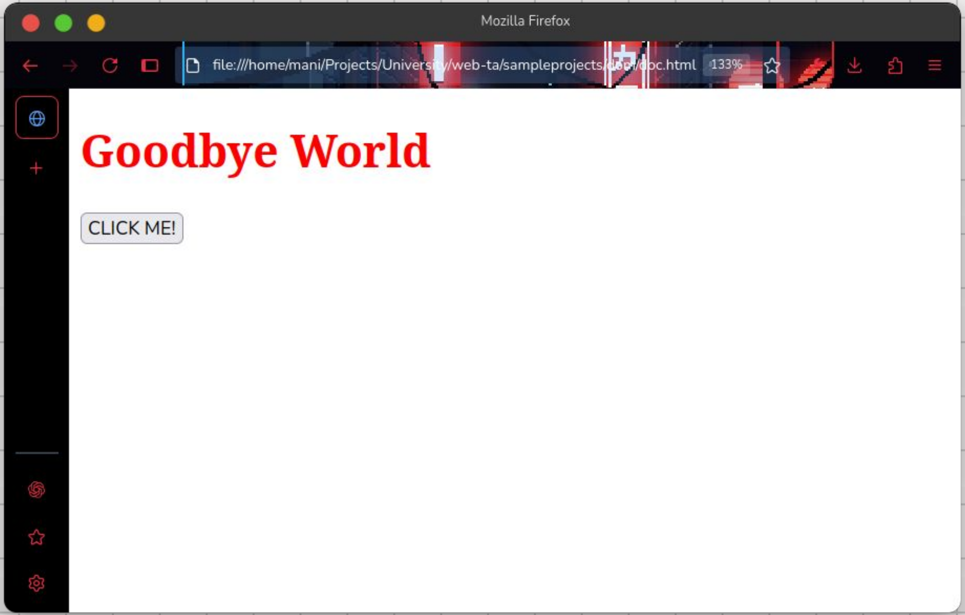
Projects/University/web-ta/sampleprojects/dom/doc.html

```html
<!DOCTYPE html>
<html>
  <body>
    <h1 id="title">Hello World</h1>
    <button>CLICK ME!</button>

    <script>
      const btn = document.querySelector('button');

      btn.addEventListener('click', () => {
        const title = document.querySelector('#title');
        title.textContent = 'Goodbye World';
        title.style.color = 'red';
      });
    </script>
  </body>
</html>
```

Example: Button Action
(cont.)


Mozilla Firefox
file:///home/mani/Projects/University/web-ta/sampleprojects/doc/doc.html
133%
Hello World
CLICK ME!

Example: Button Action (cont.)


Mozilla Firefox
file:///home/mani/Projects/University/web-ta/sampleprojects/doc/doc.html
133%
Goodbye World
CLICK ME!

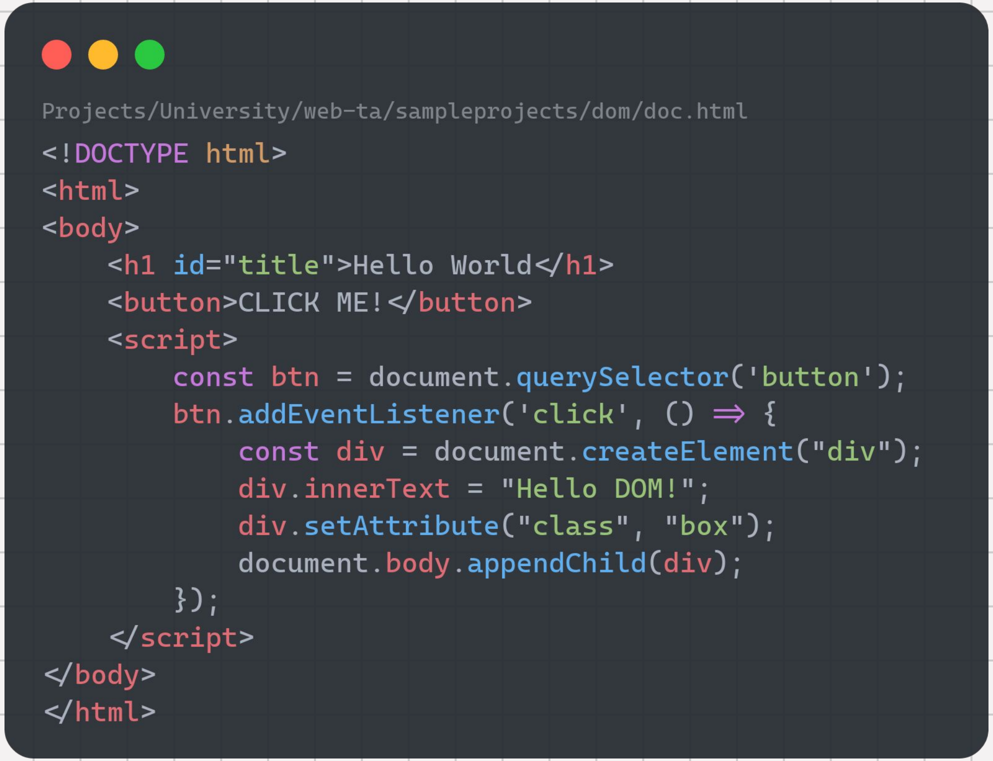
Manipulating The DOM

- The DOM represents the page as a tree of objects
- JavaScript can change text, HTML, and styles dynamically
- You can create, insert, and remove elements on the fly
- Attributes like `id`, `class`, and `src` can be modified easily
- Methods like `getElementById` and `querySelector` help find elements
- DOM manipulation powers interactive, dynamic web pages

Adding New Elements

- Use `document.createElement("tag")` to create a **new** element
- Set text using `.innerText` or `.innerHTML`
- Add attributes with `.setAttribute("name", "value")`
- Append the element using `.appendChild()` or `.append()`
- Insert elements before others with `.insertBefore()`
- Combine with events for dynamic UI changes

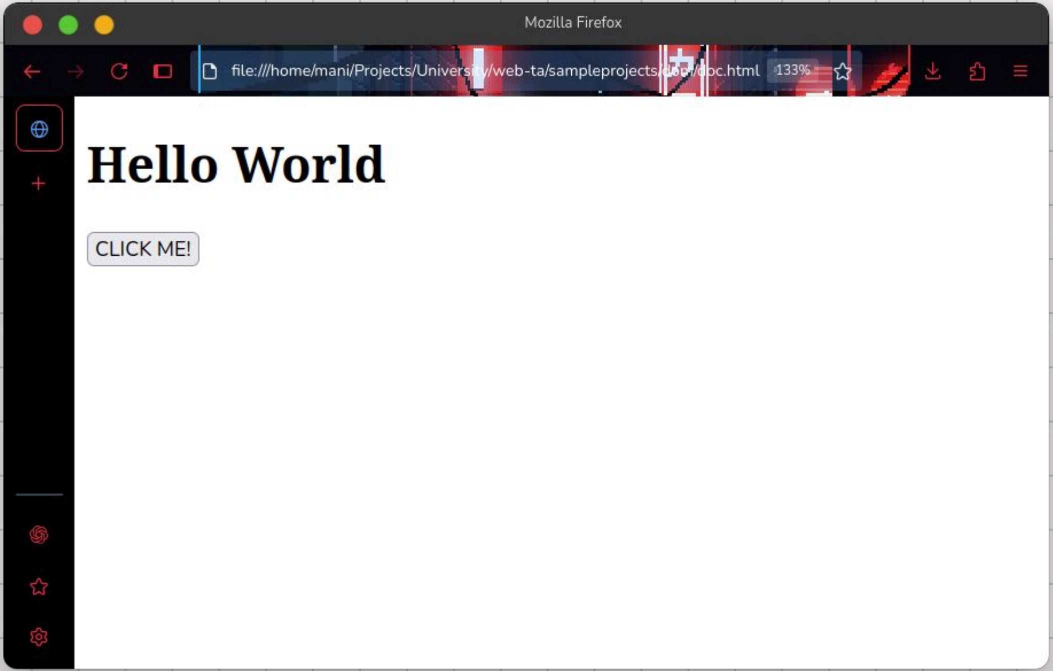
Example: New Div!



Projects/University/web-ta/sampleprojects/dom/doc.html

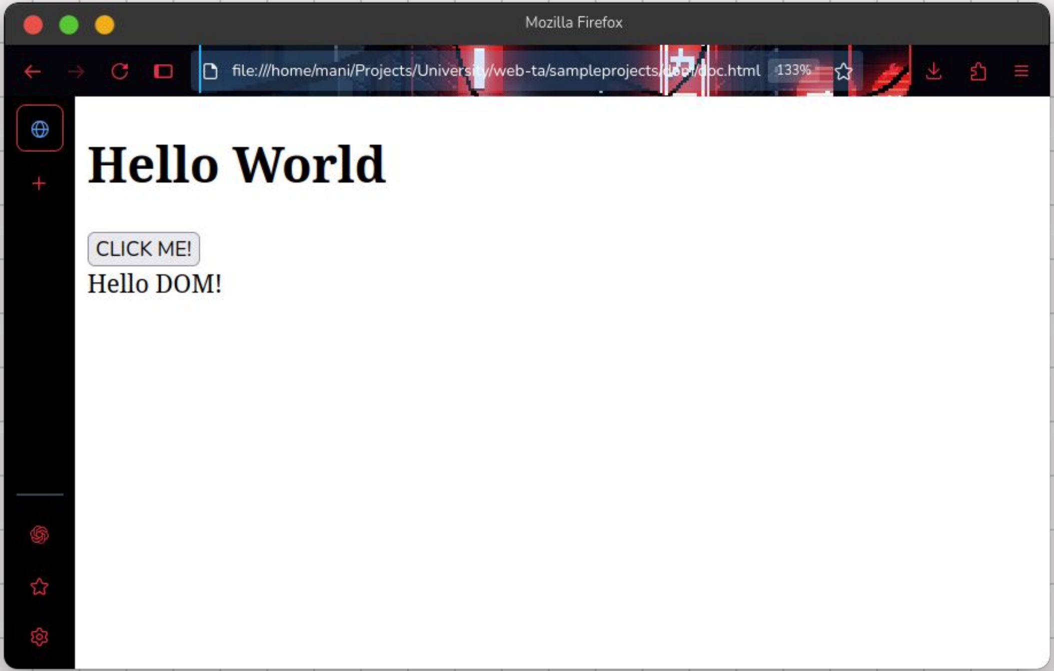
```html
<!DOCTYPE html>
<html>
<body>
    <h1 id="title">Hello World</h1>
    <button>CLICK ME!</button>
    <script>
        const btn = document.querySelector('button');
        btn.addEventListener('click', () => {
            const div = document.createElement("div");
            div.innerText = "Hello DOM!";
            div.setAttribute("class", "box");
            document.body.appendChild(div);
        });
    </script>
</body>
</html>
```

Example: New Div! (cont.)


Mozilla Firefox
file:///home/mani/Projects/University/web-ta/sampleprojects/dom/doc.html
133%

Hello World

Example: New Div! (cont.)


Mozilla Firefox
file:///home/mani/Projects/University/web-ta/sampleprojects/dom/doc.html
133%

Hello World

CLICK ME!

Hello DOM!

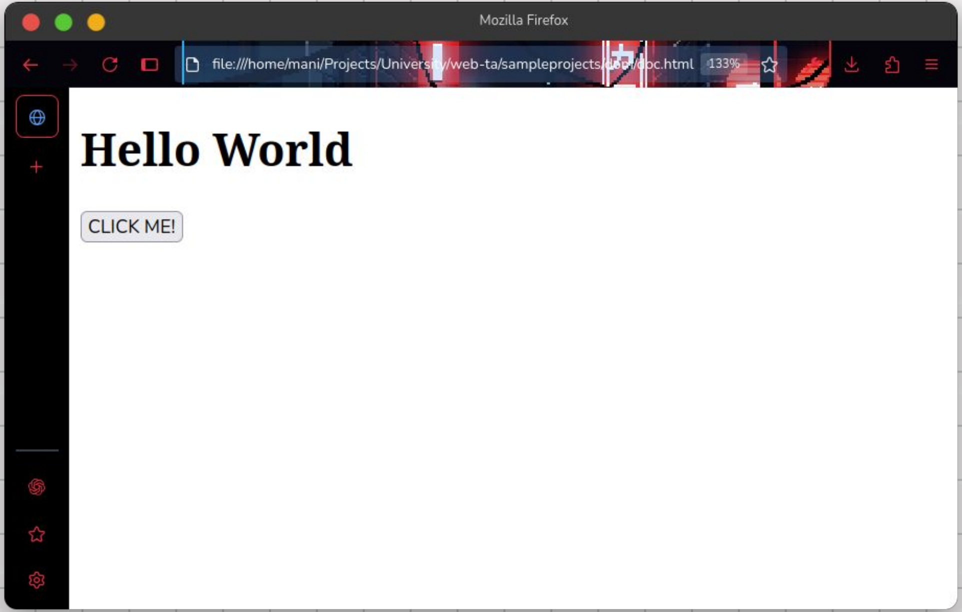
Removing Elements

- Use `.remove()` to delete an element directly
- Or call `.removeChild()` on the parent element
- `replaceChild()` can swap one element for another
- Always query the element first before removing
- Useful for dynamic UIs like todo lists or notifications
- Combine with events for interactive removals

Example: Title Goes Away!

```
Projects/University/web-ta/sampleprojects/dom/doc.html
<!DOCTYPE html>
<html>
<body>
    <h1 id="title">Hello World</h1>
    <button>CLICK ME!</button>
    <script>
        const btn = document.querySelector('button');
        btn.addEventListener('click', () => {
            const element = document.getElementById("title");
            element.remove();
        });
    </script>
</body>
</html>
```

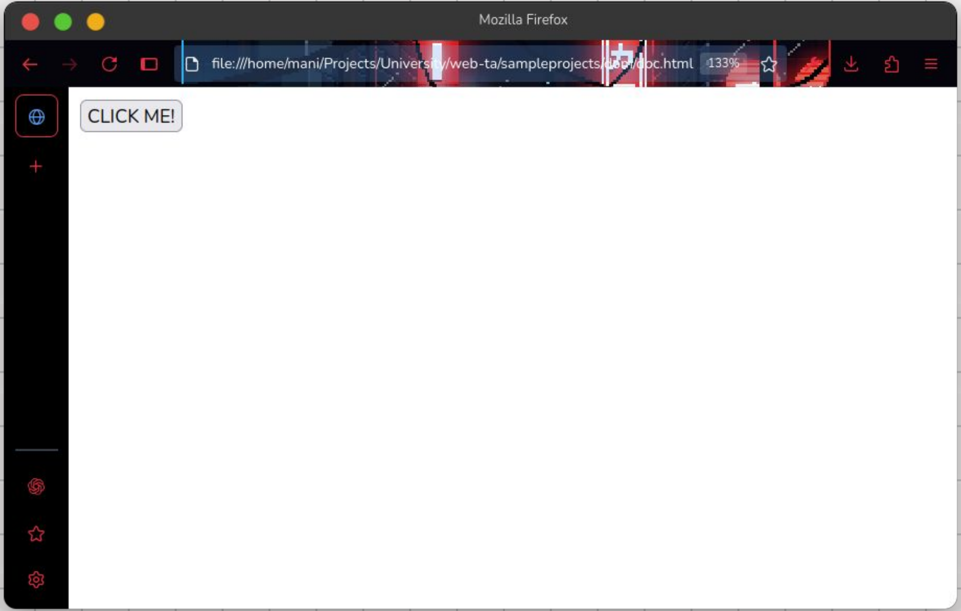
Example: Title Goes Away! (cont.)


Mozilla Firefox
file:///home/mani/Projects/University/web-ta/sampleprojects/dom/doc.html
133%

Hello World

CLICK ME!

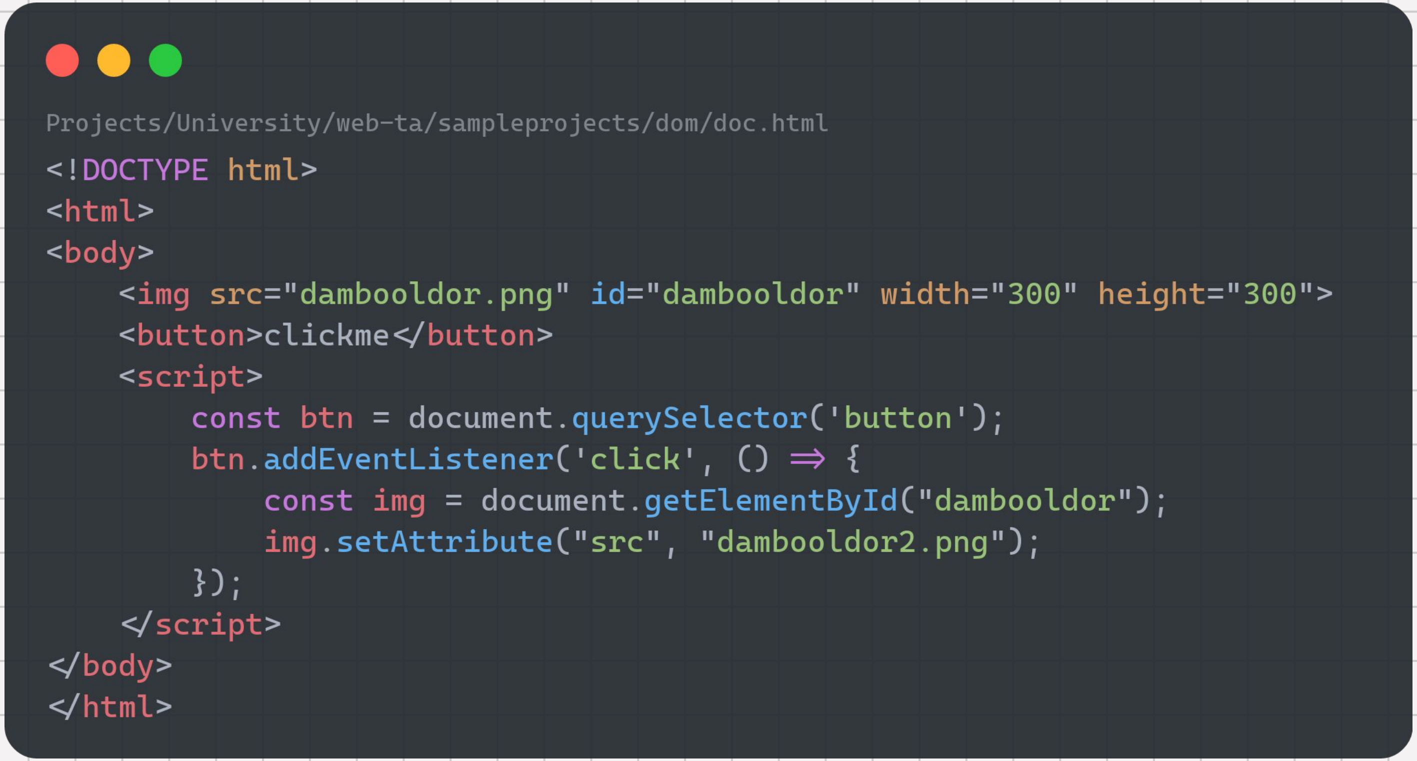
Example: Title Goes Away! (cont.)


Mozilla Firefox
file:///home/mani/Projects/University/web-ta/sampleprojects/dom/doc.html
133%
CLICK ME!

Modifying… Everything!

- Use `.setAttribute("name", "value")` to add or change an attribute
- `.getAttribute("name")` reads an attribute's value
- `.removeAttribute("name")` deletes an attribute
- Use `element.classList.add("class")` to add classes dynamically
- `classList.remove("class")` and `classList.toggle("class")` for flexibility
- Useful for dynamic styling and state changes

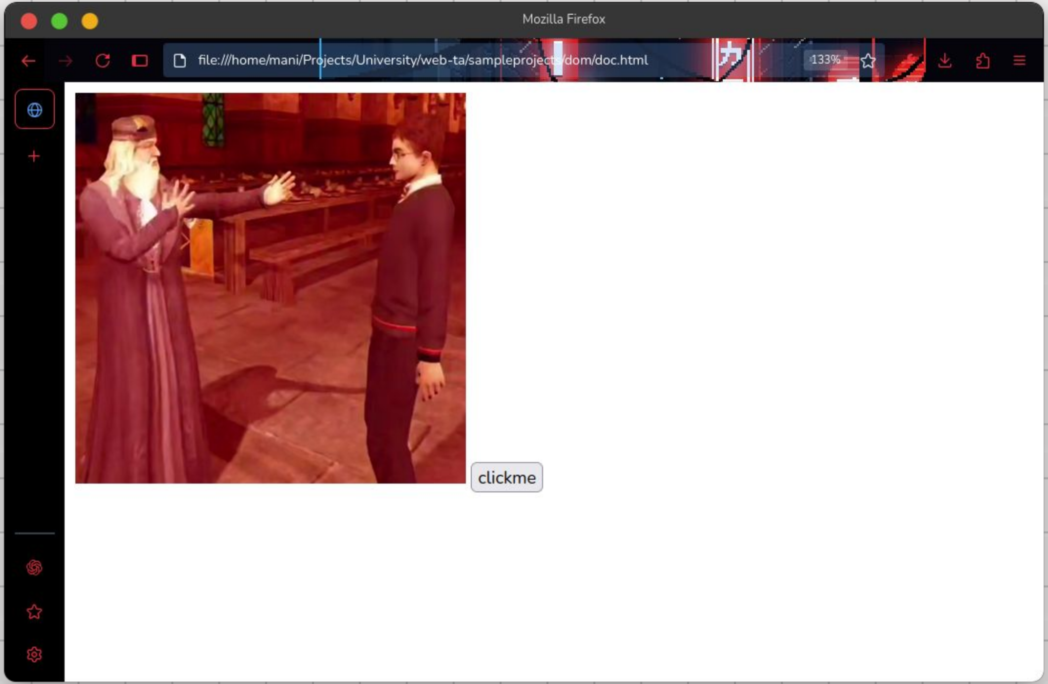
Example: Change Image



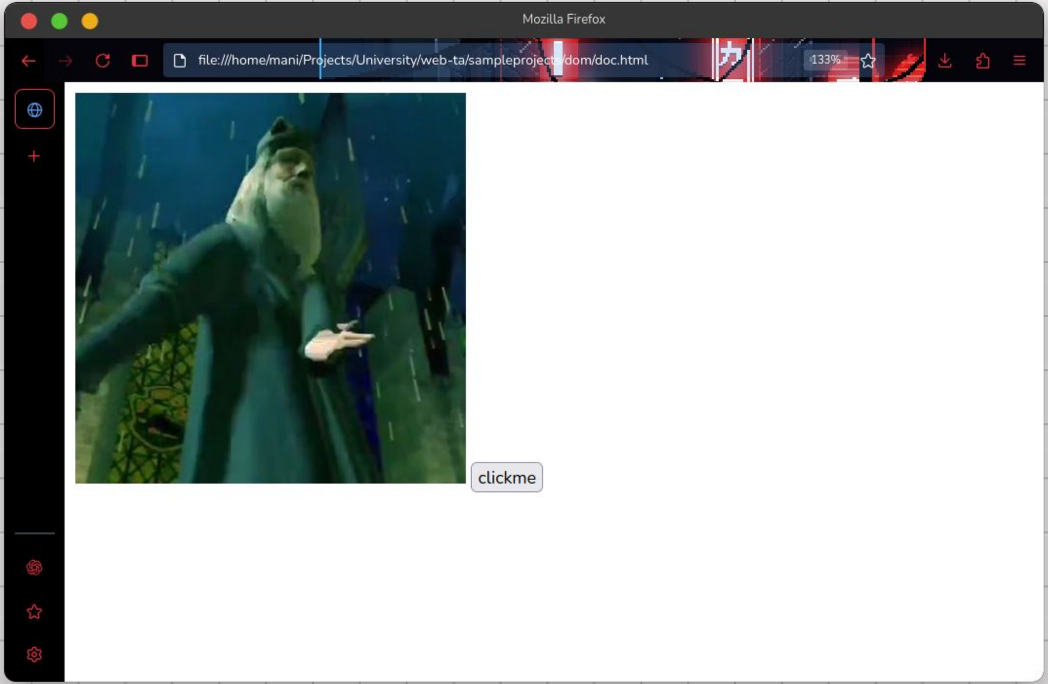
Projects/University/web-ta/sampleprojects/dom/doc.html

```html
<!DOCTYPE html>
<html>
<body>
    <img src="dambooldor.png" id="dambooldor" width="300" height="300">
    <button>clickme</button>
    <script>
        const btn = document.querySelector('button');
        btn.addEventListener('click', () => {
            const img = document.getElementById("dambooldor");
            img.setAttribute("src", "dambooldor2.png");
        });
    </script>
</body>
</html>
```

Example: Change Image (cont.)


Mozilla Firefox
file:///home/mani/Projects/University/web-ta/sampleprojects/dom/doc.html
133%
clickme

Example: Change Image (cont.)


Mozilla Firefox
file:///home/mani/Projects/University/web-ta/sampleprojects/dom/doc.html
133%
clickme

Forms

- Access form elements with `getElementById` or `querySelector`
- Read user input using `.value` or `.checked` for checkboxes/radios
- Attach submit or input events for real-time interaction
- Prevent page reload on submit with `event.preventDefault()`
- Validate user input **before** processing
- Combine with DOM methods for dynamic feedback

The `window` Object

- Represents the **browser window** containing the web page
- All **global** JavaScript objects, functions, and variables belong to it
- Provides methods like `alert()`, `confirm()`, and `prompt()`
- Gives access to `location`, `history`, `navigator`, and more
- Offers timers: `setTimeout()` and `setInterval()`
- Accessible properties: `innerWidth`, `innerHeight`, `scrollX`, `scrollY`

Class Activity Time!

What does `alert` do? How is it different from `prompt` and `confirm`?

What does `location` represent?

How would you use `window` to make your pages responsive?

Does `window` have any other children than `document`?

The `localStorage` Object

- Local storage lets websites store data **persistently** in the browser.
- Data is saved as **key–value pairs** using `localStorage.setItem()` and `getItem()`.
- The data stays even after closing the tab or browser.
- Browsers usually allow about 5–10 MB of space per site.
  `
- It follows the same-origin policy, so data is **isolated per domain**.
- Only strings are stored, it’s synchronous, and not safe for sensitive info.

The `sessionStorage`

- sessionStorage stores data only for the **current browser tab**.
- Data is saved as **key–value pairs** using `sessionStorage.setItem()` and `getItem()`.
- The data is cleared when the tab or window is closed.
- It usually has the same storage limit as localStorage (around 5–10 MB).
- It also follows the same-origin policy for security isolation.
- Stores only strings, is synchronous, and not meant for sensitive data.

Class Activity Time!

What is the key difference between local storage and session storage?

Provide an example usage of local storage.

Provide an example usage of session storage.