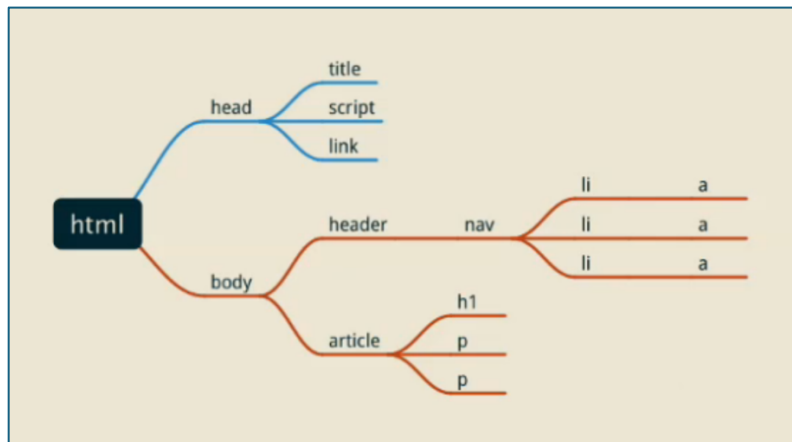


Document Object Model

Because JavaScript works hand in hand with HTML, you need to know how to interact with HTML via JavaScript code. The DOM (Document Object Model) is the interface that allows JavaScript to interact with HTML.



Every element in the HTML document is a node in a tree structure, with parent nodes, child nodes, and sibling nodes. For example, in this figure, the HTML is parent to the head and body areas on the page and then the body is a parent to its children, header and article, which are siblings and so on.

These elements model how the browser presents the page to the user. JavaScript can access, change, add, or delete elements in HTML by interacting with the DOM.

You can reference or select these elements using JavaScript with the following methods. The code shows the elements being selected and then assigned to a constant.

- `querySelector()` selects the *first* element that matches the specified CSS selector.

```
const element = document.querySelector("#myIdName");
```

- `querySelectorAll()` selects *all* elements that match the specified CSS selector and returns a node list instead of a single element.

```
const elements = document.querySelectorAll(".myClassName");
```

A useful aspect of both `querySelector` methods is that they use the same CSS selectors, with dots for classes, hashtags for IDs, and element names.

- `getElementById()` selects the first element with the specified ID attribute.

```
const element = document.getElementById("myElementId");
```

- `getElementsByClassName()` selects all elements with the specified class name and returns multiple elements.

```
const elements = document.getElementsByClassName("myClassName");
```

- `getElementsByTagName()` selects all elements with a specific tag name (like `div`, `p`, etc.) and returns multiple elements.

```
const elements = document.getElementsByTagName("p");
```

In the figure below, the first three statements use `querySelector` to select an element with the ID "container", an image element, and an element with the class "description". The last statement uses `getElementById`. Each of these four elements is then assigned to a constant.

<pre><body> <div id="container"> <h1>Welcome to the Page</h1> <p class="description">This is the description paragraph. </p> <div id="new"></div> </div></pre>	<pre>2 3 const container = document.querySelector("#container"); 4 const title = document.querySelector("h1"); 5 const descrip = document.querySelector(".description"); 6 7 const newDiv = document.getElementById("new"); 8 9</pre>
--	---

Once you selected the elements, you can use different properties to change, add, or delete information about the element.

For example, to change the content of an element in JavaScript, you can use these properties:

- `.innerHTML` sets or gets the HTML content inside an element.

```
element.innerHTML = "<p>New content</p>";
```

- `.innerText` sets or gets the visible text inside an element.

```
element.innerText = "New text content";
```

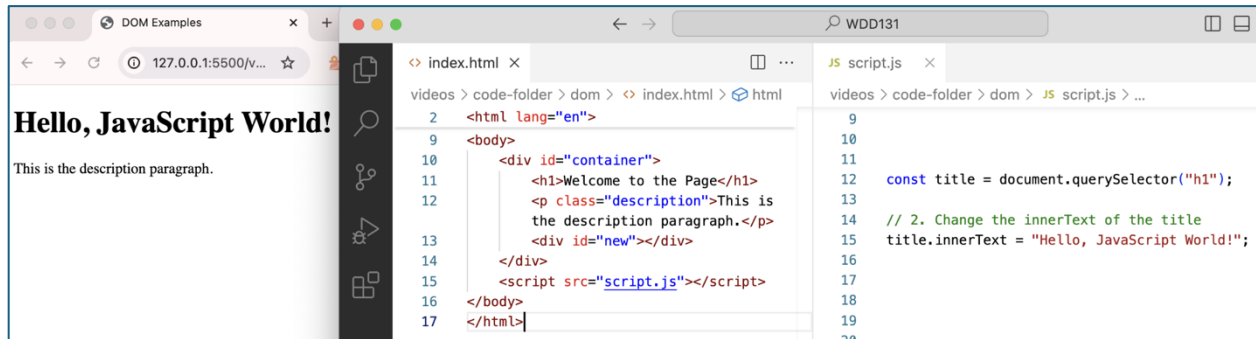
- `.textContent` is similar to `innerText` but includes any hidden elements.

```
element.textContent = "New text content";
```

- `.value` is used with form input or textarea elements to set or get the input values.

```
inputElement.value = "New input value";
```

We can change the text in our title using innerHTML. Where it originally said, "Welcome to the Page" in HTML, it now says "Hello, JavaScript World!".



In addition to changing the content of an element, JavaScript allows you to change how the element is styled.

The style property can be followed by any CSS property in camelCase, which is then assigned a value.

- `.style`

```
element.style.color = "blue";  
element.style.backgroundColor = "yellow";  
element.style.fontSize = "20px";
```

You can also set or get a class name on elements to change how they are styled. `className` will replace any existing classes.

- `.className`

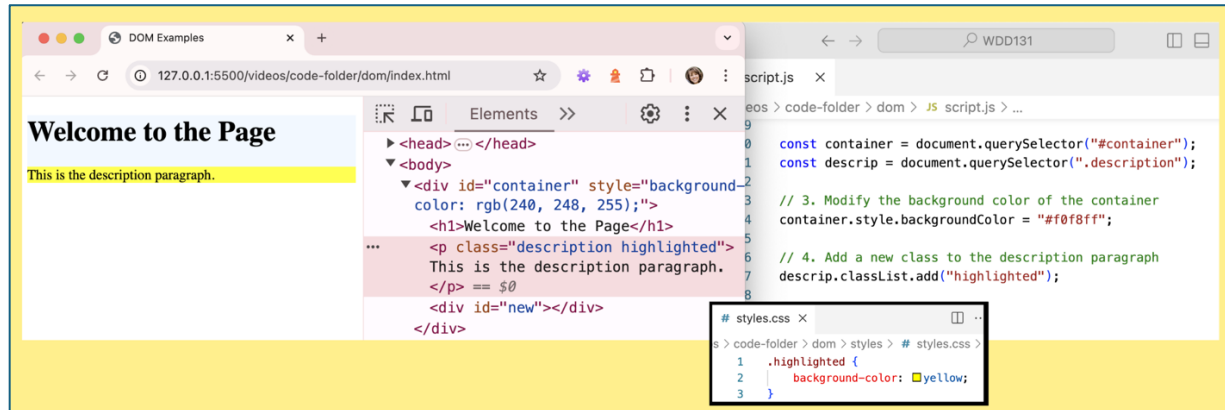
```
element.className = "newClass";
```

A more flexible way to work with classes is by using `classList` which allows adding, removing, toggling, and checking classes with the `add`, `remove`, or `toggle` methods.

- `.classList()`

```
element.classList.add("newClass");  
element.classList.remove("oldClass");  
element.classList.toggle("active");
```

You can use `style.backgroundColor` to assign a light blue to the container. You can also add a class to an element. If you already have CSS that styles any element with the class "highlighted" to have a yellow background color, you can add that class to the description paragraph using `classList.add`.



You can even create elements that don't exist in the HTML using `createElement()` and then give the new element content and add it to the HTML with `appendChild()`. You can even remove an element as well with `remove()`.

- `.createElement()`

```
const newParagraph = document.createElement("p");
```

```
newParagraph.textContent = "This is new paragraph content.";
```

A new paragraph (`<p>`) element has been created using `createElement()`. It then gets content with `textContent` and is added after the other elements in the container div to the page with `appendChild()`.

- `.appendChild()`

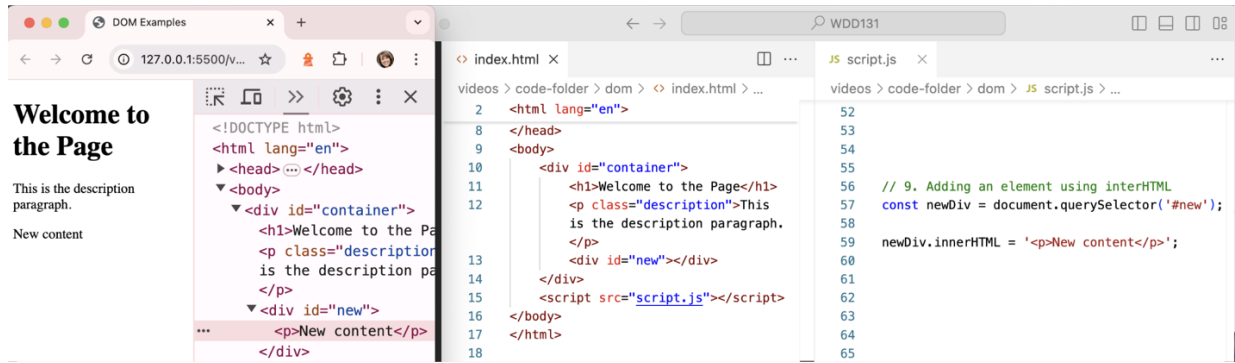
```
const container = document.getElementById("container");
```

```
container.appendChild(newParagraph);
```

- `.remove()`

```
newParagraph.remove();
```

Another way you will see a new element added along with the content is to use `innerHTML`. This creates a new HTML paragraph inside the container element and gives it content at the same time.



The first example using `appendChild` is generally safer and more efficient for adding elements, but it takes more code to implement. While `innerHTML` is quick and simple, it comes with trade-offs in terms of security and performance. `innerHTML` also replaces everything inside the container, whereas `appendChild` only adds the new content to the end of what is already there.

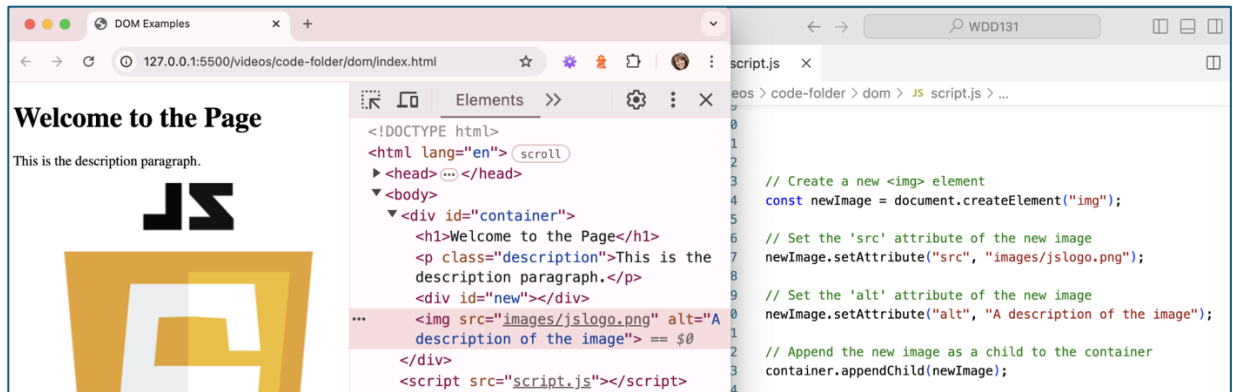
You can add an attribute to an element using the `setAttribute()` method. For example, if you wanted to add a new `src` (source) attribute to an image element, you can assign it this way in JavaScript.

- `.setAttribute()`
- ```
element.setAttribute("src", "new-image.jpg");
```

You can even retrieve and remove attributes as well.

- `.getAttribute()`
- ```
let src = element.getAttribute("src");
```
- `.removeAttribute()`
- ```
element.removeAttribute("disabled");
```

In the figure below, a new image has been created with `createElement` and then we have added two attributes with `setAttribute`: first, the `src` (source) attribute of a JS logo we already have in our images folder, and then an `alt` (alternate text) description. Lastly, we need to add it to the page with `appendChild`.



So, there are just a few of the DOM properties and methods demonstrated that you could use to retrieve or get the output you want to the user, using JavaScript to interact with the DOM of the HTML.