Modifying the DOM

Download Demo Code <../js-dom-modifying-demo.zip>

Goals

- Modify the text and HTML of elements using innerText and innerHTML
- Change inline styling of an element using the style object
- Modify attributes using getAttribute and setAttribute
- Traverse, create, append and remove elements from the DOM

Recap

So far you've seen how to select elements in the DOM using the following methods

- ***querySelector***
- ***querySelectorAll***
- ***getElementById***
- ***getElementsByClassName***
- ***getElementsByTagName***

Now that you've learned how to select elements in the DOM, let's see how we can modify their values and attributes!

Modifying Elements in the DOM

Accessing text

The easiest way to access the text of an element is to use ***innerText***

```
<section id="main-greeting">
  <article>Hello World!</article>
</section>
```

```
const mainGreeting = document.getElementById("main-greeting")

console.log(mainGreeting.innerText) // "Hello World!"
```

Modifying text

If you need to change any text, you can assign a new value to the innerHTML:

```
<section id="main-greeting">
  <article>Hello World!</article>
</section>
```

```
const mainGreeting = document.getElementById("main-greeting")

mainGreeting.innerText = "It's changed!"
```

Using textContent

Another common way to access and modify text is to use the ***textContent*** property

```
<section id="main-greeting">
  <article>Hello World!</article>
</section>
```

```
const mainGreeting = document.getElementById("main-greeting")

mainGreeting.textContent = "It's changed!"
```

So what's the difference?

There are quite a few small differences:

- innerText is aware of the rendered appearance of text, while textContent is not.
- textContent gets the content of all elements, including <script> and <style> elements. In contrast, innerText only shows "human-readable" elements.
- depending on the complexity of the content inside an element, innerText can be a bit less performant that textContent
- for now, you can use either, but be aware that both exist!

Accessing HTML

If you need to access the HTML of an element, you can use innerHTML. This will include all the elements inside of the one you select.

```
<section id="main-greeting">
  <article>Hello World!</article>
</section>
```

You will find yourself using innerText more commonly, there are some security concerns when using innerHTML if you're not careful.

Modifying HTML

If you need to change any HTML, you can assign a new value to the innerHTML:

```
<section id="main-greeting">
  <article>Hello World!</article>
</section>
```

```
const mainGreeting = document.getElementById("main-greeting")

mainGreeting.innerHTML = "<article>Just Changed!</article>"
```

Another reason it's less common to use innerHTML is that you need the string to be valid HTML for everything to work, which can be quite tedious to build

Modifying styling

Aside from the text or HTML of an element, it's very common that you'll want to change the inline style for an element.

You can access any inline CSS properties on an element using the ***style*** property

```
<h1 style="color: black; background-color: red;">
  Hello everyone!
</h1>
```

```
const mainHeading = document.querySelector("h1")

mainHeading.style.color // "black"
```

To change the style, simply reassign the value of the CSS property

```
const mainHeading = document.querySelector("h1")

mainHeading.style.color = "red"
```

How about background color?

Let's go and change the background-color CSS property:

```
const mainHeading = document.querySelector("h1")

mainHeading.style.background-color // Error!
```

Make sure to camelCase!

```
mainHeading.style.backgroundColor = "green"
```

Modifying Attributes

So far you've seen how to modify HTML and text.

The third most common thing you'll be modifying are attributes for an element

To do that, we can get attributes using getAttribute and modify attributes using setAttribute

Attributes

As a quick refresher, attributes are part of every HTML element that modify an HTML element.

An attribute either modifies the default functionality of an element type or provides functionality.

- ***src***
- ***href***
- ***class***
- ***id***
- ***type***
- ***value***

getAttribute

In order to access an attribute on an element, you can use the ***getAttribute*** method

```
const firstInput = document.querySelector("input")

firstInput.getAttribute("type") // "text"
```

setAttribute

To set an attribute on an element, you can use the ***setAttribute*** method

```
const firstInput = document.querySelector("input")

firstInput.setAttribute("type", "email")
```

Direct attribute access

There are a few attributes that you can directly access and modify as well instead of having to use getAttribute or setAttribute.

One of those is ***id***

```
<input type="text" id="first-name">
```

```
const firstInput = document.querySelector("input")

firstInput.id // "first-name"
```

```
firstInput.id = "full-name" // changes the attribute
```

A more common one you will use is the ***value*** attribute with forms in HTML

```
<input type="text">
```

```
const firstInput = document.querySelector("input")

firstInput.value // ""

firstInput.value = "Just added some value!" // changes the attribute
```

Manipulating classes

We have quite a few ways to manipulate the ***class*** attribute in JavaScript:

- ***setAttribute("class")*** - this will override the class
- ***className*** - this will give you a string representation of the class
- ***classList*** - this will give you an array-like object to add, remove or toggle classes

setAttribute("class")

If you want to access the class attribute you can use getAttribute("class") or the ***className*** property.

```
const mainHeading = document.querySelector("h1")

mainHeading.setAttribute("class", "section-heading");
```

this works, but will overwrites the previous class

className

You can also add a class by reassigning the className property

```
const mainHeading = document.querySelector("h1")

mainHeading.className += " top-heading" // works, but is prone to bugs
```

classList

An easier way to interact with classes on an element is to use the ***.classList*** method

```
const mainHeading = document.querySelector("h1")
```

```
mainHeading.classList // []

mainHeading.classList.add("top-heading") // ["top-heading"]

mainHeading.classList.remove("top-heading") // []

mainHeading.classList.toggle("top-heading") // true

mainHeading.classList.contains("top-heading") // true
```

Changing Multiple Elements

Now that you've seen how to modify styles, attributes, and text. How can we modify multiple elements at once?

```
const listItems = document.querySelectorAll("li");

// let's change them all to green!

listItems.style.color = "green"
//  TypeError: Cannot set property 'color' of undefined
```

How it's done

To do this we need to loop over multiple elements!

```
const listItems = document.querySelectorAll("li");

for(let listItem of listItems){
  listItem.style.color = "red";
}
```

Working with the Dom

Creating Elements

To create an HTML element, we can use the ***createElement*** function and pass in the name of the element

This just makes an empty element, so if we want to add any text, attributes or styling we will have to do that on another line

```
const newButton = document.createElement("button");

const newUnorderedList = document.createElement("ul");
```

```
const newDiv = document.createElement("div");

newDiv.innerText = "a brand new div!"

newDiv.style.color = "tomato"
```

Appending Elements

After you create an element, you need to place it in the DOM to see it. You can do this using the ***append*** method.

append is a method that a parent element calls and you pass in the child element that you would like to place inside of the parent element

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

const newLi = document.createElement("li");

newLi.innerText = "Hello!";

ul.append(newLi);
```

append will place the element as the last child in the parent. If you would like the element to be the first child, you can use the ***prepend*** method.

Removing Elements

If we want to remove elements in the DOM, we can use the handy ***remove*** method.

In order to remove an element, we first need to find it.

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

ul.remove();
```

This function can only be called on a single element, so if you need to remove multiple elements you'll need to call ***remove*** multiple times.

Finding elements near another element

As you start adding and removing elements in the DOM, there are times where you might want to know not only information about an element, but it's parents or children.

You might want to:

- find an element and remove some or all of its children
- find an element and add an element to one of its children

Thankfully there are some very helpful methods for doing just that!

Setting the stage with some HTML

demo/traversal-methods/index.html

```
<!DOCTYPE html>
<html>
<body>
  <section>
    <h1>Here is a main heading!</h1>
    <div>
      <p>
        Here is a paragraph inside a div!
      </p>
      <ul>
        <li>First list item in a div</li>
        <li>Second list item in a div</li>
      </ul>
    </div>
    <div>Here is the second div!</div>
  </section>
  <script src="script.js"></script>
</body>
</html>
```

Take a look at some parent, child, and sibling element relationships.

We will be focusing specifically on the element.

Accessing a parent element

If you want to access the parent element of another element, you can use the ***parentElement*** method.

```
const foundDiv = document.querySelector("div")

foundDiv.parentElement // <section></section>
```

Accessing the children of an element

If you want to access the child elements of another element, you can use the ***children*** method.

```
foundDiv.children // HTMLCollection(2) [p, ul]
foundDiv.firstElementChild // <p>Here is a paragraph inside a div!</p>
foundDiv.lastElementChild // <ul></ul>
```

Accessing the siblings of an element

If you want to access the previous sibling or next sibling element of another element, you can use the ***previousElementSibling*** or ***nextElementSibling*** method.

```
foundDiv.previousElementSibling // <h1>Here is a main heading!</h1>
foundDiv.nextElementSibling // <div>Here is the second div!</div>
```

Text Nodes

You may come across other methods for finding things in the DOM, we have shown you the most common ones, but as you learn more you may come across something called a text node

Nodes Vs. Elements

With some of these finder methods, you will see that you don't always get back an HTML element, you sometimes get back what is called a text node

Everything in the DOM is a node, some nodes are not actually HTML elements, but text or even comments!

With most of the common traversal methods, you will not need to worry about text nodes,

You will be using other methods less frequently, but know that they exist if you need to see elements near/above/below the element you find.

Recap

- The DOM allows us to use JavaScript to find and modify elements and their attributes
- We can modify text, styles and much more using the DOM
- To easily modify multiple elements, we can iterate over a list of elements and change each one