Back to Week 2

DOM | Introduction & Selectors

LESSON

Learning Goals

After this lesson, you will be able to:

- Understand what the Document Object Model (DOM) is
- Understand the tree representation of the DOM
- Select elements by using `.getElementById()` or `.getElementsByClassName()`
- Manipulate element's content with `innerHTML`
- Manipulate DOM objects properties
- Create DOM objects and add them to the DOM

Introduction to DOM

What is the DOM?

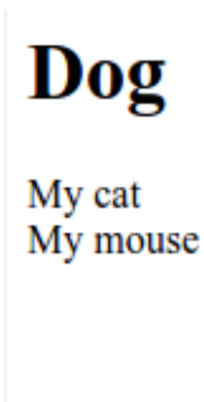
> The Document Object Model (DOM) is an API (Application Programming Interface) for HTML and XML documents. It provides a **structured representation of the document (web page) and defines a way that the structure can be accessed from JavaScript**. This allows us to change the document structure, style or the content from JavaScript!

The DOM provides a representation of the document as a structured group of nodes and objects that have **properties** and **methods**.

Essentially, the DOM connects web pages to scripts or programming languages.

What is a Web page?

A Web page is a document. This document can be either displayed in the browser (window) or as the HTML source. But it is the **same document** in both cases.

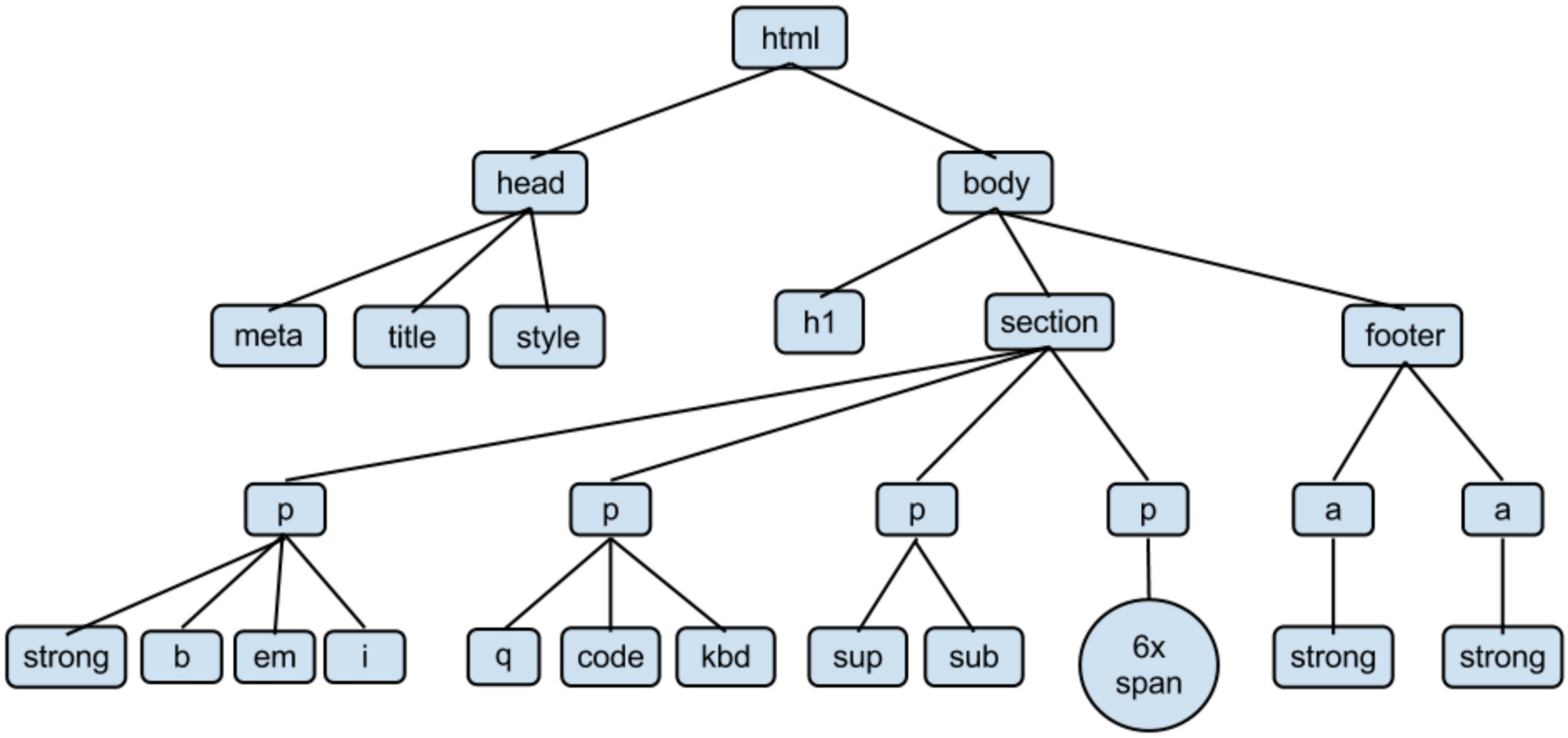
Browser	HTML Source	JavaScript DOM Object
		

The Document Object Model (DOM) provides another way to represent, store and manipulate that same document. The DOM is a fully object-oriented representation of the web page, and it can be modified from JavaScript.

The **W3C DOM standard** forms the basis of the DOM implemented in most modern browsers. Many browsers offer extensions beyond the W3C standard, so you have to be careful when using them on the web where documents may be accessed by various browsers with different DOMs.

A Tree?

The DOM represents a document as a tree. The tree is made up of parent-child relationships. One parent can have one or many children nodes.



Example Setup

Let's create a folder and a couple of files to work in:

```
$ mkdir dom-intro
$ cd dom-intro
$ touch dom.html
$ touch index.js
$ code .
```

Explain this code

Add some basic html boilerplate code:

```
<!DOCTYPE >
<html>
  <head>
    <title>DOM Intro</title>
  </head>
  <body>
    <!-- include js in the end of the body to make sure
    the DOM is already loaded before you want to make updates in it-->
    <script src="index.js"></script>
  </body>
</html>
```

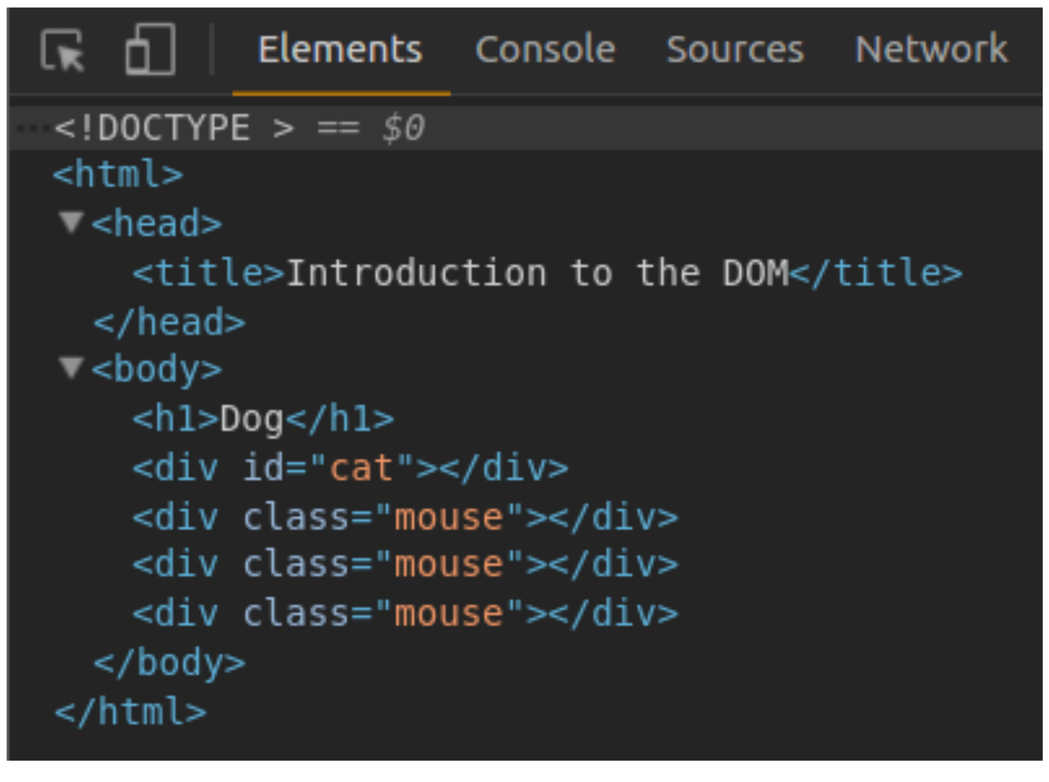
Explain this code

Now add some `div` elements inside `body`:

```
<h1>Dog</h1>
<div id="cat"></div>
<div class="mouse"></div>
<div class="mouse"></div>
<div class="mouse"></div>
<div class="hello"></div>
```

Explain this code

Now open the `dom.html` in the browser. Check the DOM Tree representation of the example:

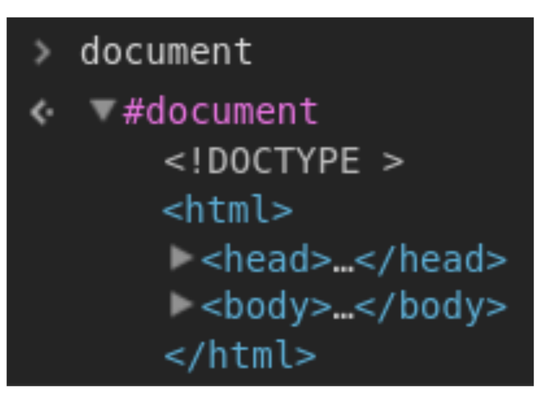


Accessing DOM objects from JavaScript

Now we have the document loaded in our browser and we can open the Chrome Developer tools to access the document from JavaScript. First, we will use the console, but later we can use the same methods and objects to manipulate our webpage dynamically from our JavaScript files.

The document object

Every website can be accessed by the JavaScript DOM using the **`document`** object, which is instantiated automatically when we render the page. The first thing we are going to do is to access it from the browser's console:



As we can see, the `document` will return the source code of our website! The **Document Object** has a very long list of properties and methods available. In this lesson, we are going to learn a few of the most useful ones to manipulate websites from JavaScript.

Searching for Elements by ID

Before we can modify elements in our webpage, we may want to look for them so we can have a reference. We can find any HTML element in the document using JavaScript and the `document` object.

We can access any element that has a particular ID attribute by using the **getElementById** method of the `document` object:

```
let element = document.getElementById('some-id-goes-here');
```

Explain this code

⚠️ Remember that `id` must be a string, and it will return the first occurrence only.

Setting an HTML element content

Once you have an element selected, it is possible to change this object's properties. Any changes we make with JavaScript will be immediately reflected in the HTML.

We are going to learn about two useful properties we may use to change an element's content or style.

To change the content of an element, we can use the **element.innerHTML** property:

```
// index.js

console.log('JS connected'); // <== just a quick check up to make sure js doc is connected prope
rly

// TO GET THE ELEMENT FROM DOM YOU CAN USE "getElementById"
let theCatDiv = document.getElementById('cat');

console.log(theCatDiv); // <== what can you see in browser's console

// TO ADD TEXT TO DOM USE "innerHTML"
theCatDiv.innerHTML = "I'm a cat";

// set the HTML content of "otherElement" to "I'm a cat"
otherElement.innerHTML = theCatDiv.innerHTML;
```

Explain this code

Elements attributes and content

Once we have fetched an element using one of the above methods, we can also modify them. Here we are going to learn a few useful properties that we can manipulate to change the style, set a class name or an ID programmatically.

Another useful element property is **element.style**, which allows us to change the styles from JavaScript.

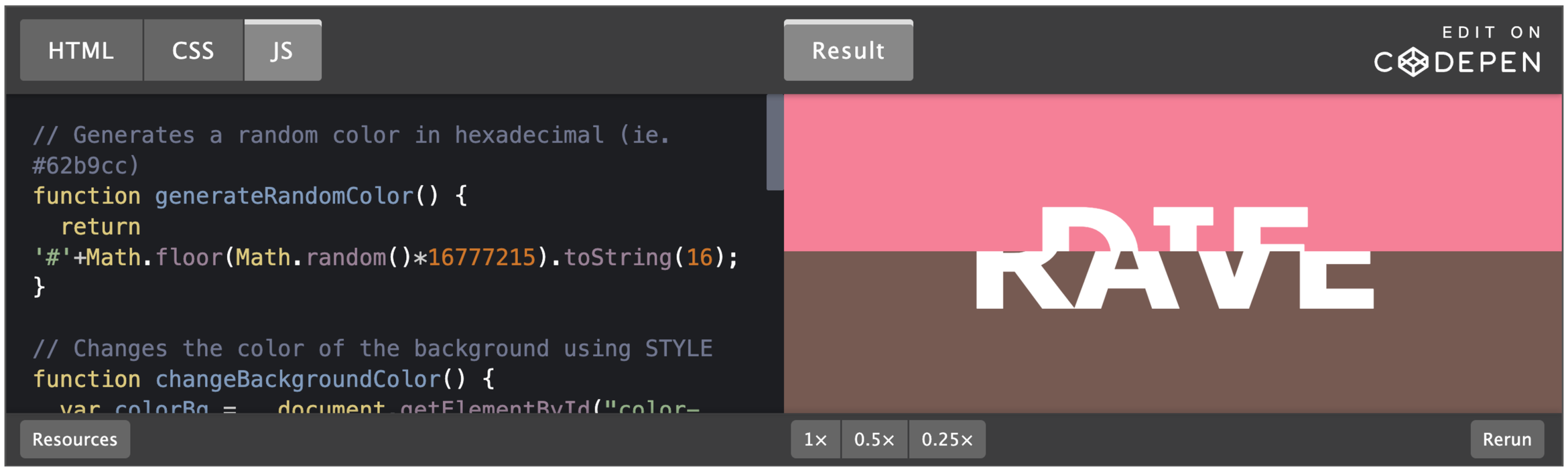
```
// index.js
// ...

theCatDiv.style.backgroundColor = 'red';
theCatDiv.style.border = '2px green solid';
theCatDiv.style.fontSize = '50px';
theCatDiv.style.marginTop = '30px';
theCatDiv.style.paddingBottom = '30px';
```

Explain this code

⚠️ Note that the property names may differ from CSS; (i.e. `backgroundColor` instead of `background-color`).

Example

Check the following example on how we can change the background color of an element and the text of an `h1`. Go ahead into CodePen and play around with the example.



Selecting Elements

So far we saw we can access elements using the **id** attribute. Another, amongst other ways, is using the **class** attribute.

Accessing Elements by Class Name

It is super useful to know how to access elements using the class that is attached to them.

.getElementsByClassName() returns an array of all child elements which have all of the given class names.

```
let elements = document.getElementsByClassName(names);
```

Explain this code

- `elements` is a **HTMLCollection** of found elements
- `names` is a string representing the list of class names to match; class names are separated by whitespace, not commas
- `getElementsByClassName` can be called on any element, not only on the document. The element on which it is called will be used as the root of the search.

An example:

```js
// index.js
// ....

let mice = document.getElementsByClassName('mouse');
console.log(mice); // <== HTMLCollection(3) [div.mouse, div.mouse, div.mouse]
```

Explain this code

HTML Collections

The HTMLCollection represents a generic collection (**array-like object**) of elements.

In the `.getElementsByClass()` case, it will return the collection with all the elements that have the `className` we are looking for. It is important to notice, that the collection will be ordered in the same index it appears on the DOM.

An HTMLCollection in the HTML DOM is live; it is automatically updated when the document is changed.

Iterate over an HTML Collection

The HTMLCollection is an array-like object but is not an array. So we can't use the array methods like `forEach`, `map`, `push`, etc.

To iterate over an HTMLCollection, we should use a `for` loop.

✔️ Another option we have is to **turn our HTML collection into an array**, using `spread operator [...]` or any other approach to copy the array (we learned 4-5 of them, remember 😉).

```js
// index.js
...
let miceArray = [...mice];

console.log(miceArray); // <== [div.mouse, div.mouse, div.mouse]
```

Explain this code

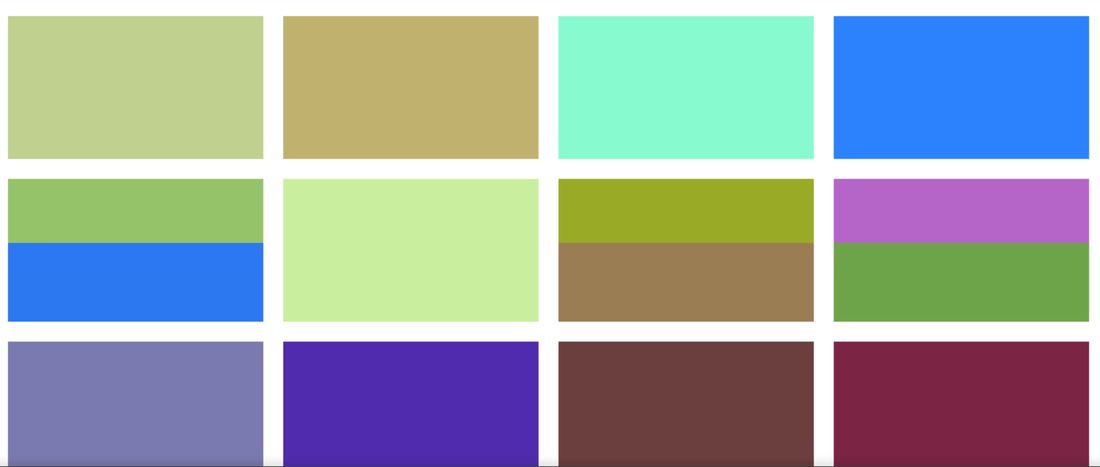
After this, we can use any array method to manipulate with it.

Example

Check how we can make a psychedelic wall change color from our the boxes that use the same class. Don´t be afraid, click on the CodePen and play around a bit! 😉

HTML CSS JS Result EDIT ON CODEPEN

```
// Generates a random color in hexadecimal (ie.
#62b9cc)
function generateRandomColor() {
  return
'#'+Math.floor(Math.random()*16777215).toString(16);
}

function changeColor() {
  var discoBoxes =
document.getElementsByClassName("disco-box");
```



Resources 1× 0.5× 0.25× Rerun

Accessing by Tag Name

The method `.getElementsByTagName()` returns an HTMLCollection of elements with the given HTML tag name. The complete document is searched, including the root node.

```
let elements = document.getElementsByTagName(name);
```

Explain this code

- `elements` is a live HTMLCollection of found elements in the order they appear in the tree.
- `name` is a string representing the name of the element. The special string "*" represents all elements.

An example:

```
// index.js
// ...

let divs = document.getElementsByTagName('div');
console.log(divs); // <== [div#cat, div.mouse, div.mouse, div.mouse]
```

Explain this code

`querySelector` Accessing First Found Selector

The querySelector() returns the **first element** within the document (using depth-first pre-order traversal of the document's nodes) that matches the specified group of selectors. Supported by IE8 and up.

```
let theFirstFoundElem = document.querySelector(selectors);
```

Explain this code

- `theFirstFoundElem` is an element object
- `selectors` is a string containing one or more selectors separated by commas.

In this example, the first element in the document with the class "mouse" is returned:

```
let firstMouse = document.querySelector('.mouse');
let firstDiv = document.querySelector('div');

console.log(firstMouse); // <== <div class="mouse"></div>
console.log(firstDiv);
// <== <div id="cat" style="background-color: red; border: 2px solid green; font-size: 50px; mar
gin-top: 30px; padding-bottom: 30px;">I'm a cat</div>
```

Explain this code

`querySelectorAll` Accessing an Array of Selectors

The method `.querySelectorAll()` returns a list of the elements within the document that match the specified group of selectors. It is very similar to `.querySelector()`, except it doesn't stop on the first element.

The object returned is a **NodeList**. Supported by IE8 and up.

```
// index.js
// ...

let elementList = document.querySelectorAll(selectors);
```

Explain this code

- `elementsList` is a non-live NodeList of element objects.
- `selectors` is a string containing one or more CSS selectors separated by commas.

This example returns a list of all `div` elements within the document with either a `mouse class` or a `cat id`:

```
let allDivs = document.querySelectorAll('.mouse, #cat');

console.log(allDivs); // <== NodeList(4) [div#cat, div.mouse, div.mouse, div.mouse]
```

Explain this code

`.className` Getting and Setting the Class Name

The `className` property of the element gets and sets the value of the class attribute of the specified element.

```
// get the class name of "element"
let cName = element.className;

// set the class name of "otherElement"
otherElement.className = cName;
```

Explain this code

In our example:

```
// index.js
// ...
let mouse1 = document.querySelector('.mouse');
console.log(mouse1.className); // <== mouse
```

Explain this code

Later we can use this knowledge to manipulate, perhaps, with *active* and *inactive* DOM elements:

```
let el = document.getElementById('item');

if (el.className === 'active') {
  el.className = 'inactive';
} else {
  el.className = 'active';
}
```

Explain this code

The code snippet and more about using `className` property, you can dind on the official MDN Element.className page.

Getting and Setting the ID

Just like we did with a class name, here you can find the same with **id** elements' attribute.

`.id` - gets or sets the element's identifier (attribute **id**).

```
// get the id of "element"
let idStr = element.id;

// set the id of "otherElement"
otherElement.id = 'some-value';
```

Explain this code

A List of DOM Properties and Methods

Here is a list of some more DOM Properties and DOM Methods for you to refer to

Property / Method	Description
`element.attributes`	Returns a NamedNodeMap of an element's attributes
`element.childNodes`	Returns a collection of an element's child nodes (including text and comment nodes)
`element.children`	Returns a collection of an element's child element (excluding text and comment nodes)
`element.classList` ❤️	Returns an array with the class name(s) of an element
`element.className`	Sets or returns the value of the class attribute of an element
`element.clientHeight` ❤️	Returns the height of an element, including padding
`element.clientLeft`	Returns the width of the left border of an element
`element.clientTop`	Returns the width of the top border of an element
`element.clientWidth` ❤️	Returns the width of an element, including padding
`element.contains()`	Returns true if a node is a descendant of a node, otherwise false
`element.contentEditable`	Sets or returns whether the content of an element is editable or not
`element.firstChild`	Returns the first child node of an element
`element.firstElementChild`	Returns the first child element of an element
`element.focus()`	Gives focus to an element
`element.getAttribute()` ❤️	Returns the specified attribute value of an element node
`element.getAttributeNode()`	Returns the specified attribute node
`element.getElementsByClassName()`	Returns a collection of all child elements with the specified class name
`element.getElementsByTagName()`	Returns a collection of all child elements with the specified tag name

`element.getFeature()`	Returns an object which implements the APIs of a specified feature
`element.hasAttribute()`	Returns true if an element has the specified attribute, otherwise false
`element.hasAttributes()`	Returns true if an element has any attributes, otherwise false
`element.hasChildNodes()`	Returns true if an element has any child nodes, otherwise false
`element.id`	Sets or returns the value of the id attribute of an element
`element.isEqualNode()`	Checks if two elements are equal
`element.isSameNode()`	Checks if two elements are the same node
`element.lastChild`	Returns the last child node of an element
`element.lastElementChild`	Returns the last child element of an element
`element.nextSibling`	Returns the next node at the same node tree level
`element.nextElementSibling`	Returns the next element at the same node tree level
`element.nodeName`	Returns the name of a node
`element.nodeValue`	Sets or returns the value of a node
`element.offsetHeight`	Returns the height of an element, including padding, border and scrollbar
`element.offsetWidth`	Returns the width of an element, including padding, border and scrollbar
`element.offsetLeft`	Returns the horizontal offset position of an element
`element.offsetParent`	Returns the offset container of an element
`element.offsetTop`	Returns the vertical offset position of an element
`element.parentNode` ❤️	Returns the parent node of an element
`element.parentElement`	Returns the parent element node of an element
`element.previousSibling`	Returns the previous node at the same node tree level
`element.previousElementSibling`	Returns the previous element at the same node tree level
`element.querySelector()` ❤️	Returns the first child element that matches a specified CSS selector(s) of an element
`element.querySelectorAll()` ❤️	Returns all child elements that matches a specified CSS selector(s) of an element
`element.removeAttribute()`	Removes a specified attribute from an element
`element.removeAttributeNode()`	Removes a specified attribute node, and returns the removed node
`element.removeChild()`	Removes a child node from an element
`element.replaceChild()`	Replaces a child node in an element
`element.removeEventListener()`	Removes an event handler that has been attached with the addEventListener() method
`element.scrollHeight`	Returns the entire height of an element, including padding
`element.scrollLeft`	Sets or returns the number of pixels an element's content is scrolled horizontally
`element.scrollTop`	Sets or returns the number of pixels an element's content is scrolled vertically

`element.scrollWidth`	Returns the entire width of an element, including padding
`element.setAttribute()`	Sets or changes the specified attribute, to the specified value
`element.setAttributeNode()`	Sets or changes the specified attribute node
`element.style` ❤️	Sets or returns the value of the style attribute of an element
`element.tabIndex`	Sets or returns the value of the tab index attribute of an element
`element.tagName`	Returns the tag name of an element
`element.textContent`	Sets or returns the textual content of a node and its descendants
`element.title`	Sets or returns the value of the title attribute of an element
`element.toString()`	Converts an element to a string
`nodelist.item()`	Returns the node at the specified index in a NodeList
`nodelist.length`	Returns the number of nodes in a NodeList

Summary

In this lesson, you have learned:

- What is the DOM and the tree structure of HTML documents
- How to select elements based on their id - `getElementById()`
- How to modify an element using `innerHTML` and `style`
- How to select elements based on their class name - `getElementsByClassName()`
- How to select elements based on their tag name - `getElementsByTagName()`
- How to select the first element based on a query
    - First occurrence - `querySelector()`
    - All occurrences - `querySelectorAll()`
- How to get and use element's class name - `Element.className`
- How to get and use element's id - `Element.id`

Extra Resources

- Document properties - MDN
- Document methods - MDN

Mark as completed

PREVIOUS

Bonus: JS | Context & Function invocation

NEXT

DOM | Manipulation →