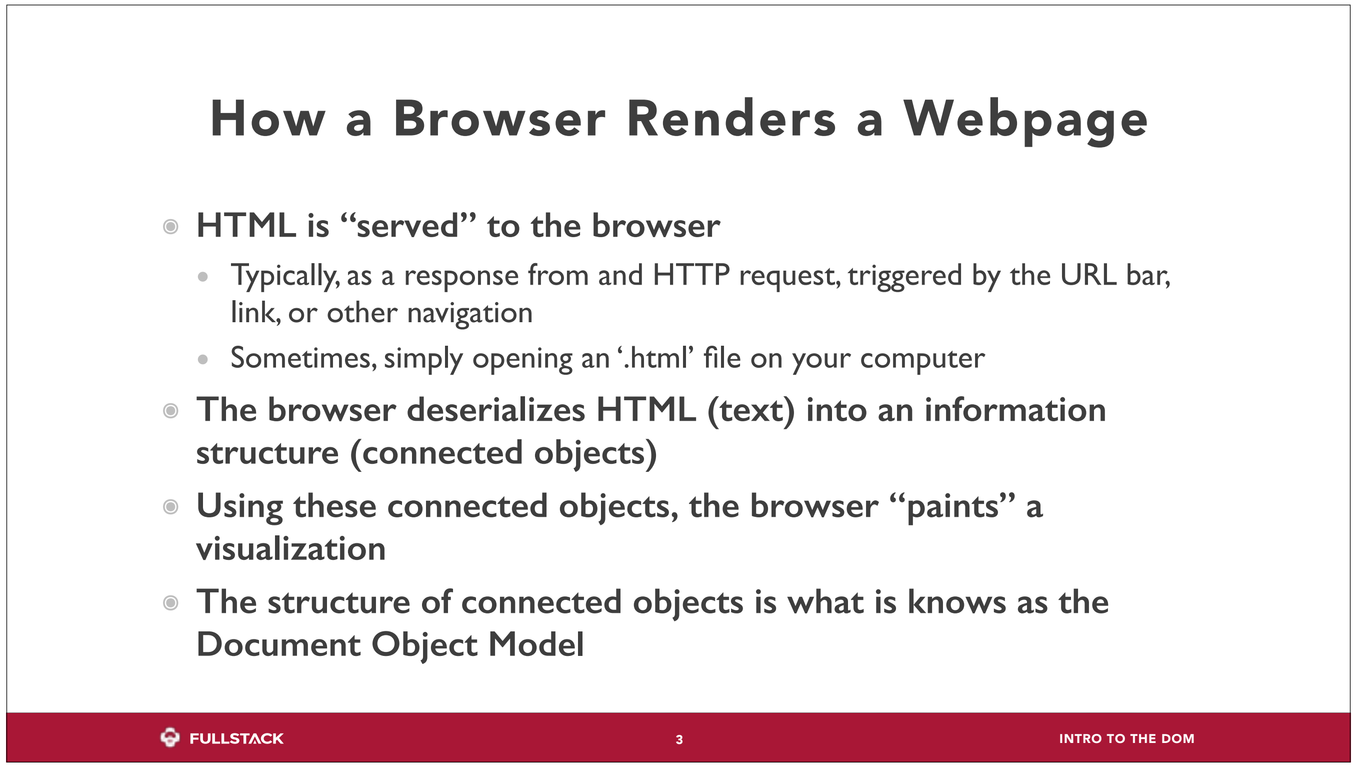
Introduction to the Document Object Model

A technically correct definition of the DOM

The Document Object Model (DOM) is a **cross-platform** and **language-independent** convention for representing and interacting with objects in **HTML, XHTML, and XML documents**.

The **nodes** of every **document** are organized in a **tree structure**, called the **DOM tree**. **Objects** in the DOM tree may be addressed and manipulated by using methods on the objects. The public interface of a DOM is specified in its application programming interface (API).



How a Browser Renders a Webpage

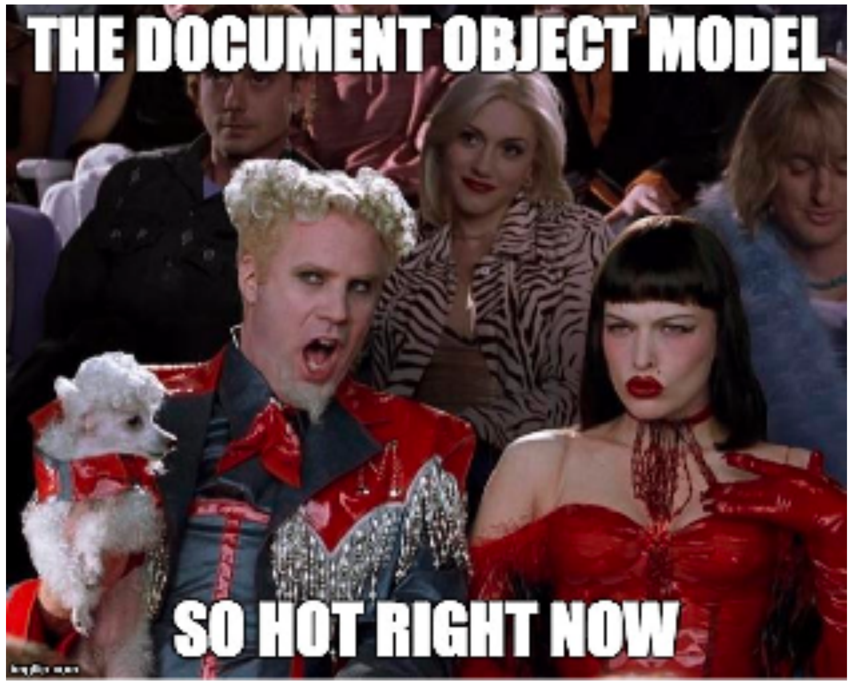
- **HTML is "served" to the browser**
  - Typically, as a response from and HTTP request, triggered by the URL bar, link, or other navigation
  - Sometimes, simply opening an '.html' file on your computer
- **The browser deserializes HTML (text) into an information structure (connected objects)**
- **Using these connected objects, the browser "paints" a visualization**
- **The structure of connected objects is what is knows as the Document Object Model**

Serialization: turning an abstract idea into a stream of bytes in order to store the object in memory.

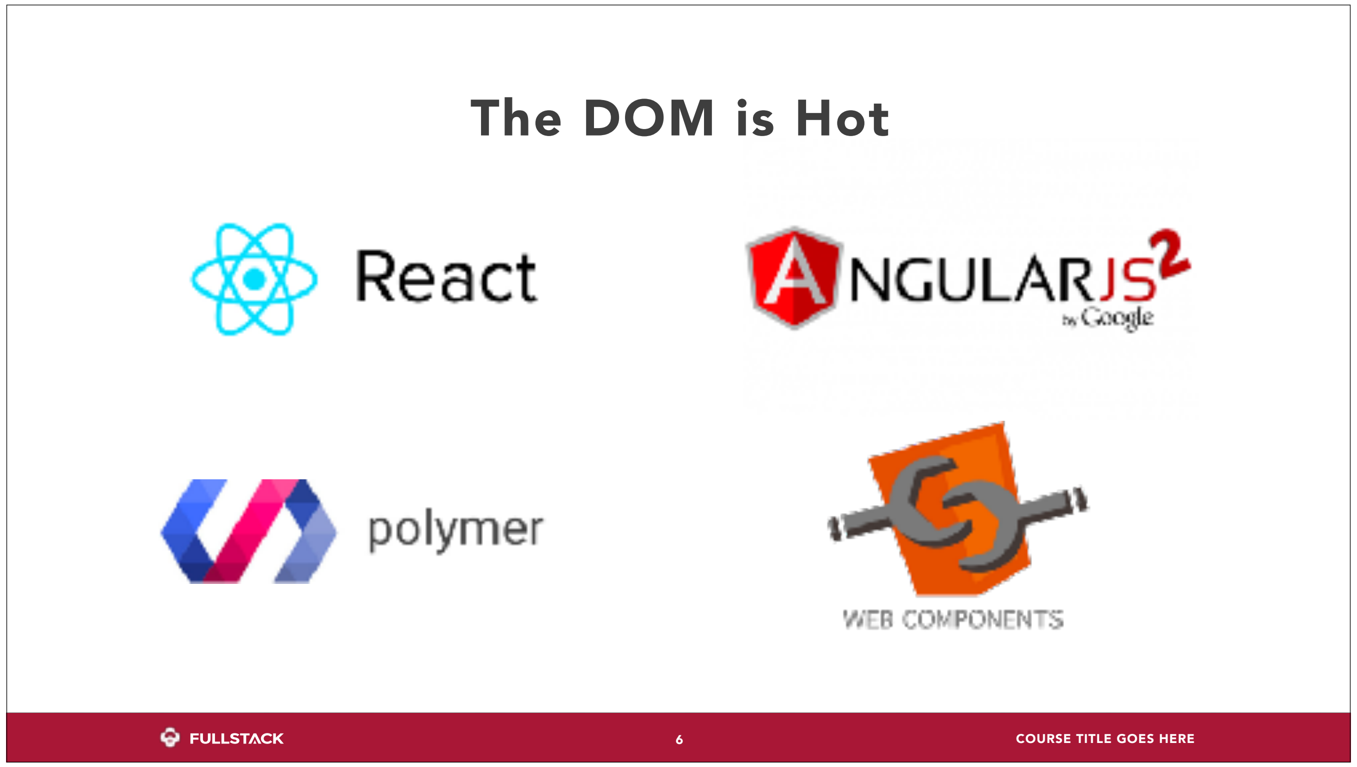
Human Example: Thoughts into words
HTML: the serialized version of the DOM
The DOM is the "deserialized" version of HTML
Deserialization: Reading a book and visualizing the ideas in your mind

Why study the DOM?

- **The Document Object Model is:**
  - The most ubiquitous publishing platform ever created
  - What allows web pages to render, respond to user events and change
  - Connects JavaScript to HTML



Not only that, but we're seeing a resurgence in interest in the DOM because we're moving more work towards the frontend. That means that we have applications that are creating and managing thousands of DOM elements which makes the frontend less of a visual tool and more of an engineering and app platform.



The DOM is Hot

React

ANGULARJS 2 by Google

polymer

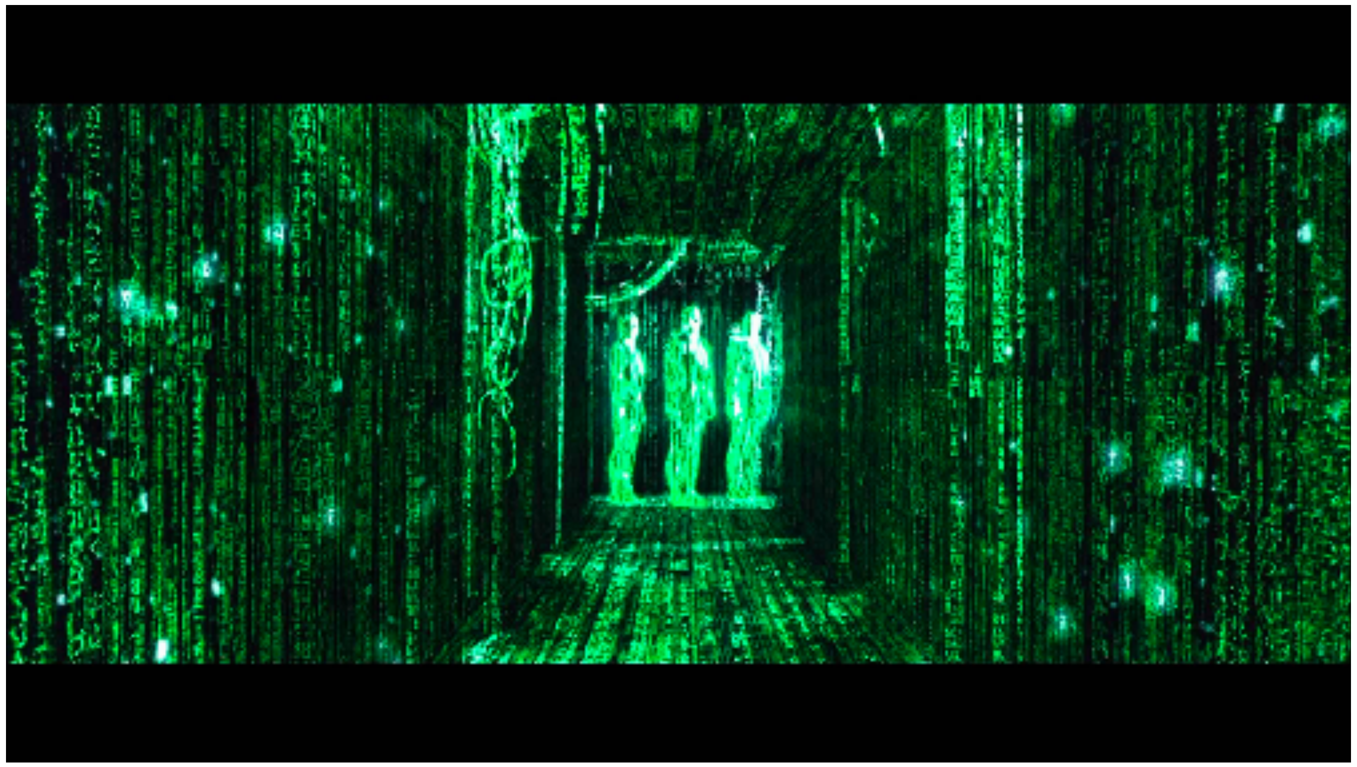
WEB COMPONENTS

PROMPT: Who's heard of these technologies?

If you haven't been under a cave for the last few years, these are the hottest techs in web dev right now.

All of these frameworks are changing our conception of how best to generate, manipulate and maintain web pages and they all are intimately connected with how we think about the DOM.

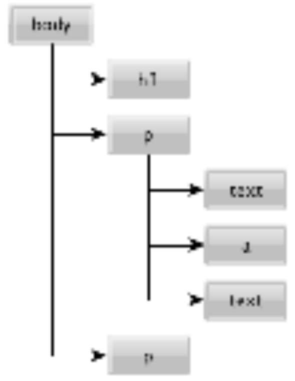
This is how most of us see the web. We experience it as users. But underneath every web page there is a complicated life of objects interacting. Just like in the Matrix...



This is the DOM. It's not what we see, it's how the things under the hood are behaving.

The DOM is a Tree

- Trees are a data structure from computer science
- The main idea here: There is a Node that branches into other Nodes (its children Nodes)
  - Each Node can have 0 to many children Nodes
  - Nodes can have 0 or 1 parent
  - Nodes can have 0 to many Sibling Nodes

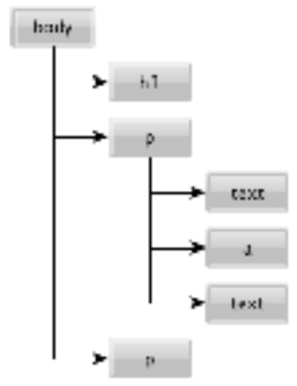


- whats the parent, whats the child, what's the sibling.
- similar to folder structure (show sublime text)

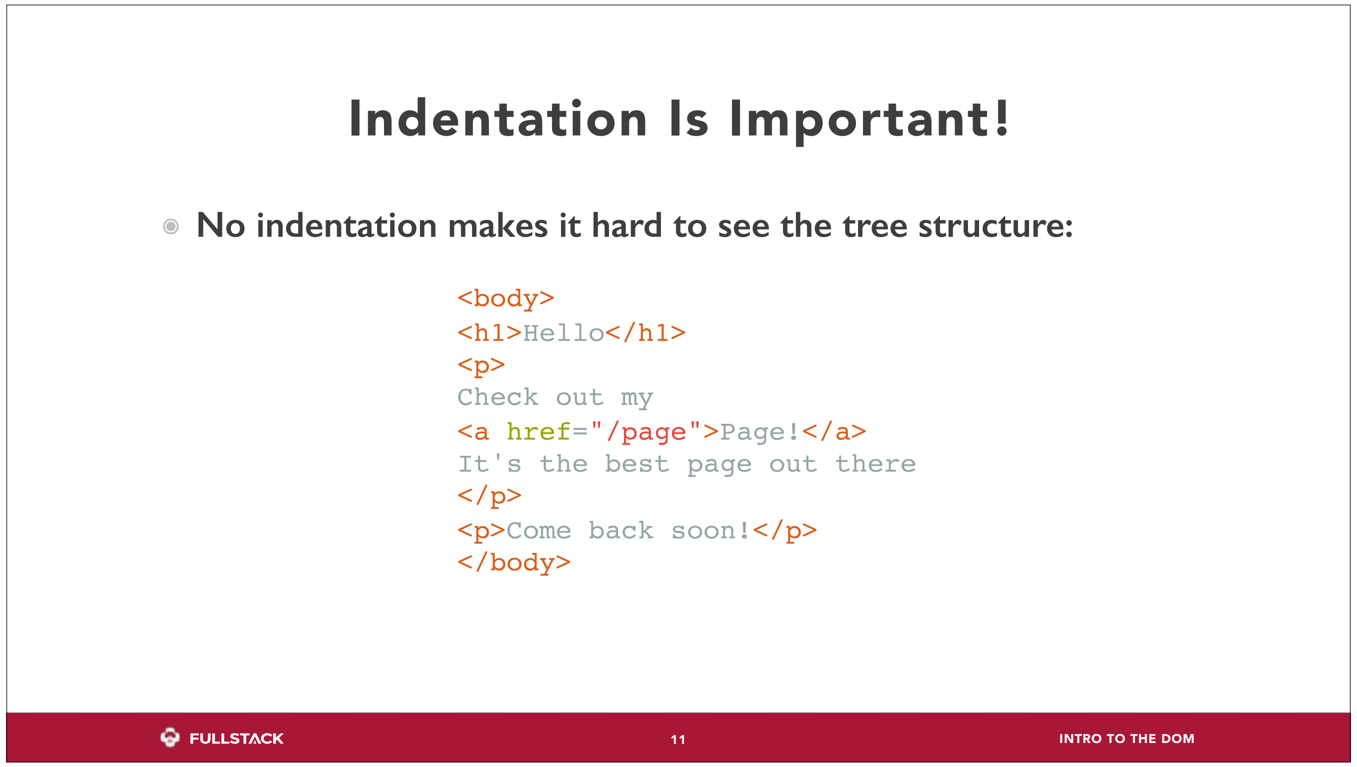
The DOM is a Tree

```
<body>
  <h1>Hello</h1>
  <p>
    Check out my
    <a href="/page">Page!</a>
    It's the best page out there
  </p>

  <p>Come back soon!</p>
</body>
```



http://software.hixie.ch/utilities/js/live-dom-viewer/?%3C!DOCTYPE%20html%3E%0A%3Chtml%3E%0A%09%3Chead%3E%0A%09%20%20%20%20%3Ctitle%3EMy%20first%20web%20page%3C%2Ftitle%3E%0A%09%3C%2Fhead%3E%0A%09%3Cbody%3E%0A%09%20%20%20%20%3Ch1%3EHello%20world!%3C%2Fh1%3E%0A%09%20%20%20%20%3Cp%3E%3Cb%3EI%27m%20very%20excited%3C%2Fb%3E%20to%20be%20exploring%20the%20Document%20Object%20Model.%20Here%27s%20the%20%3Ca%20href%3D%22wikipedia.org%2FDOM%22%3EWikipedia%3C%2Fa%3E%20page%20on%20the%20topic.%3C%2Fp%3E%0A%09%3C%2Fbody%3E%0A%3C%2Fhtml%3E



Indentation Is Important!

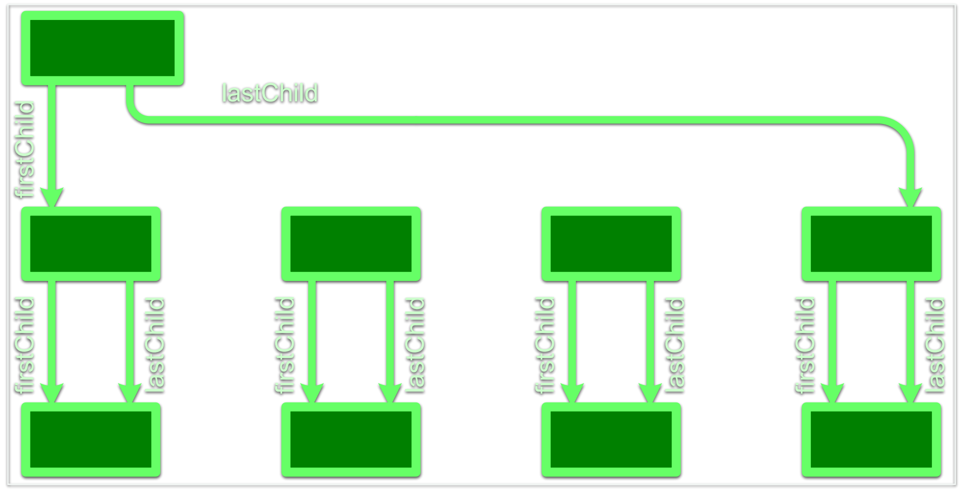
- No indentation makes it hard to see the tree structure:

```html
<body>
<h1>Hello</h1>
<p>
Check out my
<a href="/page">Page!</a>
It's the best page out there
</p>
<p>Come back soon!</p>
</body>
```

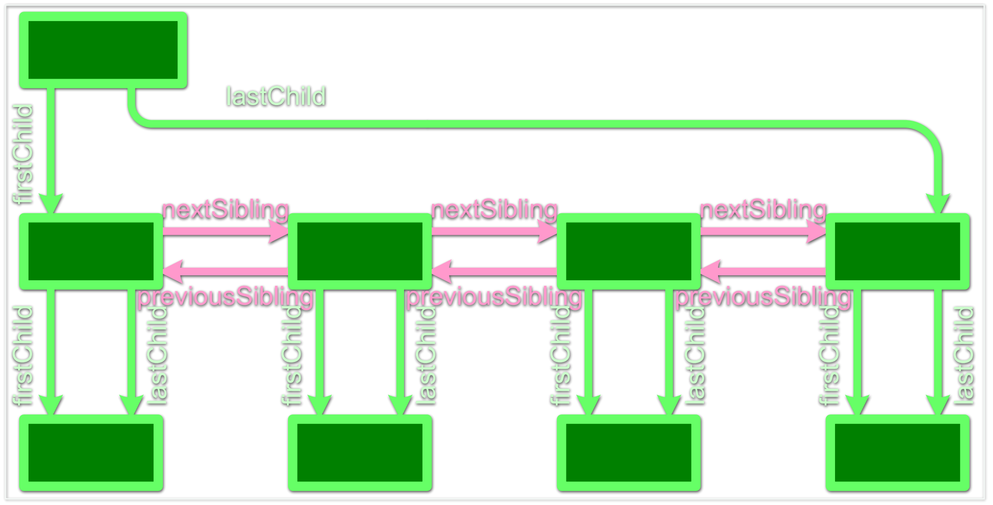
- That's why indenting properly is important. The level of indentation IS the tree structure.

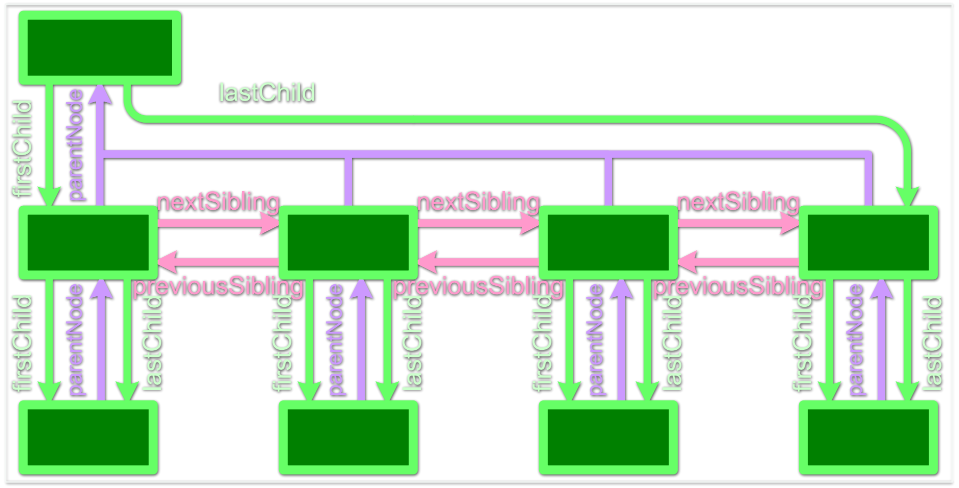
Tree Structures are easy to navigate

- At any point in the DOM you are at a Node
- No matter where you go, you're still at a Node
  - Child
  - Parent
  - Sibling
- All Nodes share similar DOM navigation methods



where have we seen firstChild and lastChild? (answer = css)


lastChild
firstChild
nextSibling
nextSibling
nextSibling
previousSibling
previousSibling
previousSibling
firstChild
lastChild
firstChild
lastChild
firstChild
lastChild
firstChild
lastChild


lastChild
firstChild
parentNode
nextSibling
nextSibling
nextSibling
previousSibling
previousSibling
previousSibling
firstChild
parentNode
lastChild
firstChild
parentNode
lastChild
firstChild
parentNode
lastChild
firstChild
parentNode
lastChild

Nodes have lots of Attributes

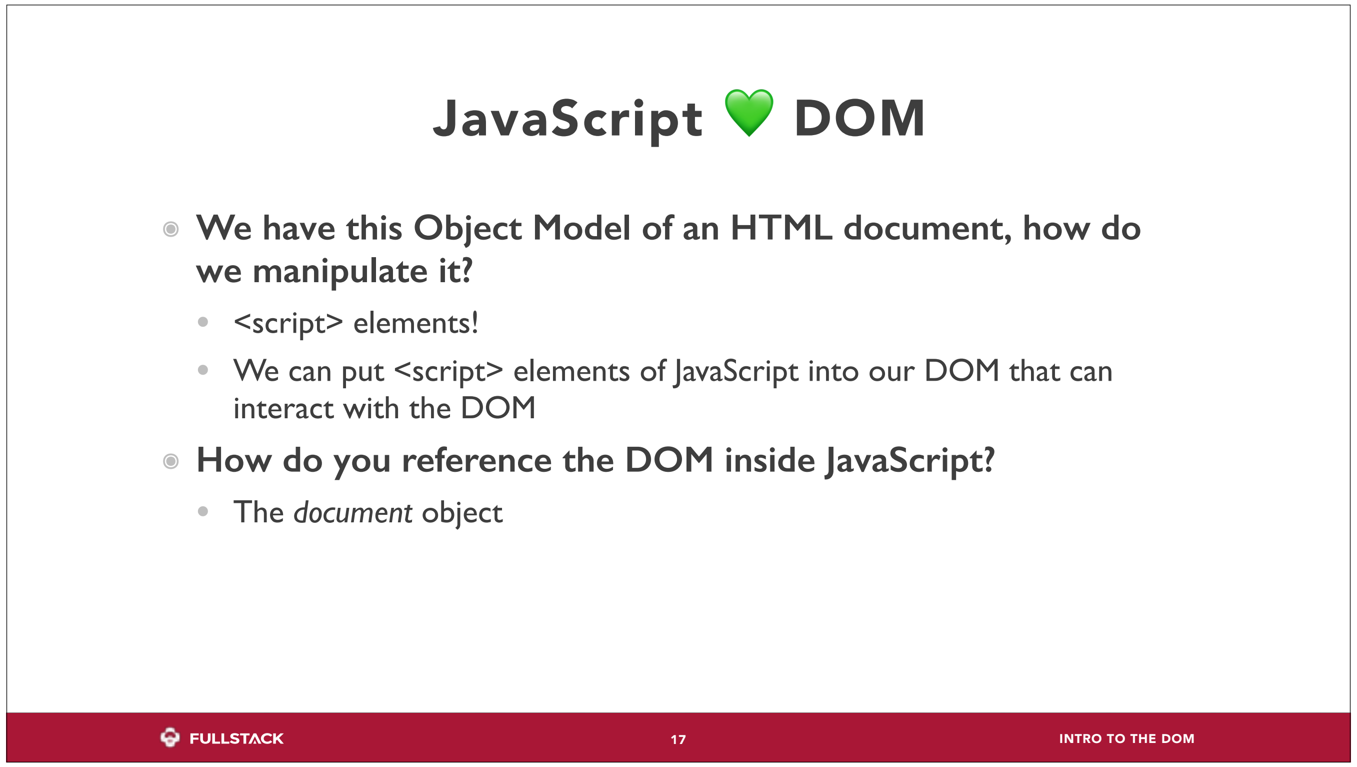
- Nodes are JavaScript Objects
- Nodes have Attributes that are JavaScript properties
- Attributes define how the Node looks and responds to User activity

one node

hundreds of properties!

Node are just JavaScript ObjectsL they're bags of attributes.
These objects contain 100s of attributes that let you change the node in two main ways:
- how it looks and is drawn by the browser
- how it responds to user input

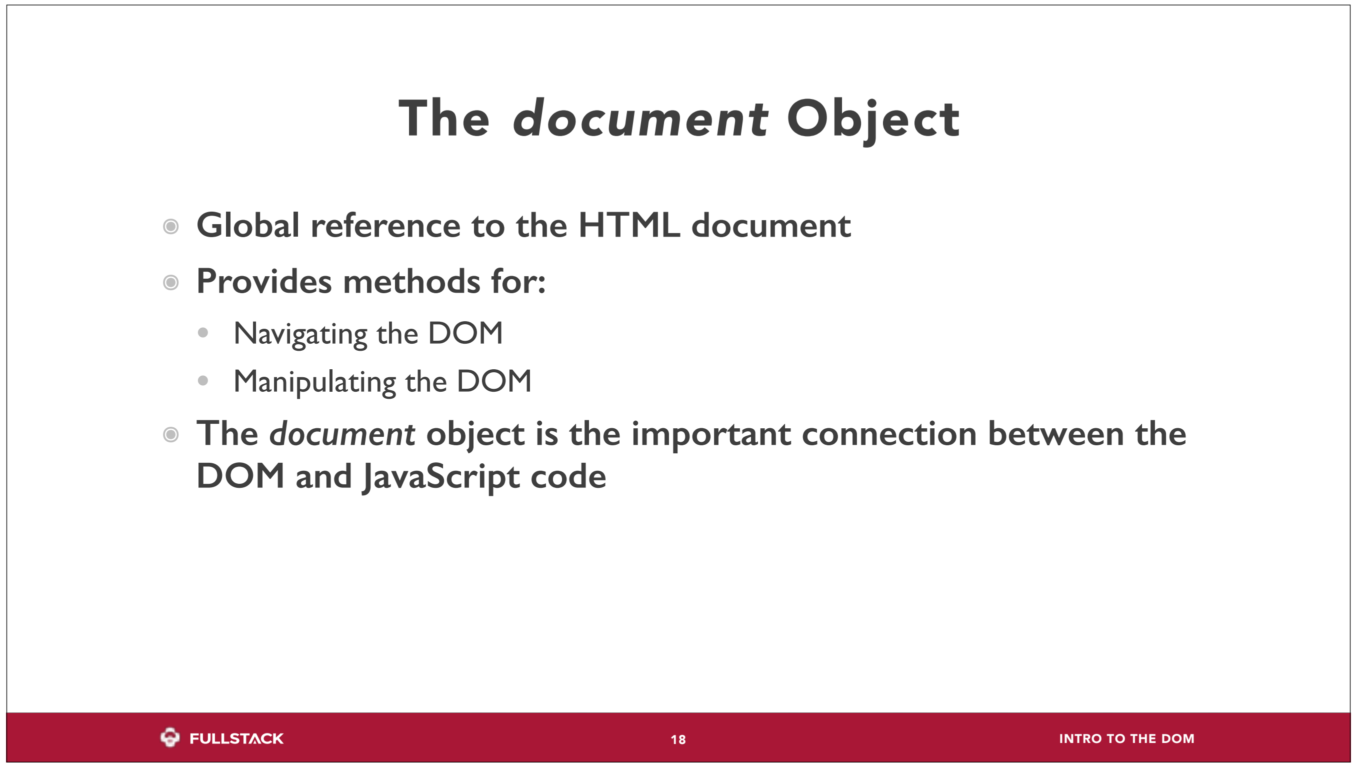
Now we'll get into how to use JS to manipulate these Nodes.



JavaScript 💚 DOM

- **We have this Object Model of an HTML document, how do we manipulate it?**
  - <script> elements!
  - We can put <script> elements of JavaScript into our DOM that can interact with the DOM
- **How do you reference the DOM inside JavaScript?**
  - The *document* object

browser “gives” the DOM to our JS environment, that exists for every page in a web browser

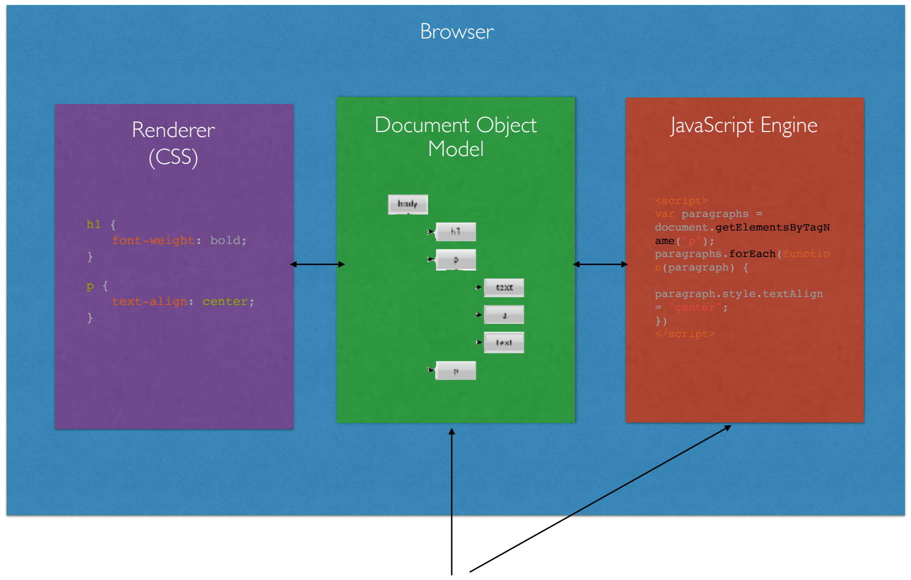


The *document* Object

- Global reference to the HTML document
- Provides methods for:
  - Navigating the DOM
  - Manipulating the DOM
- The *document* object is the important connection between the DOM and JavaScript code

Without the document object, we could write JavaScript but we wouldn't be able to manipulate the DOM. The browser gives us access to the object so that we can do these things and that's the power behind HTML and JS.

CSS of course is involved as well.



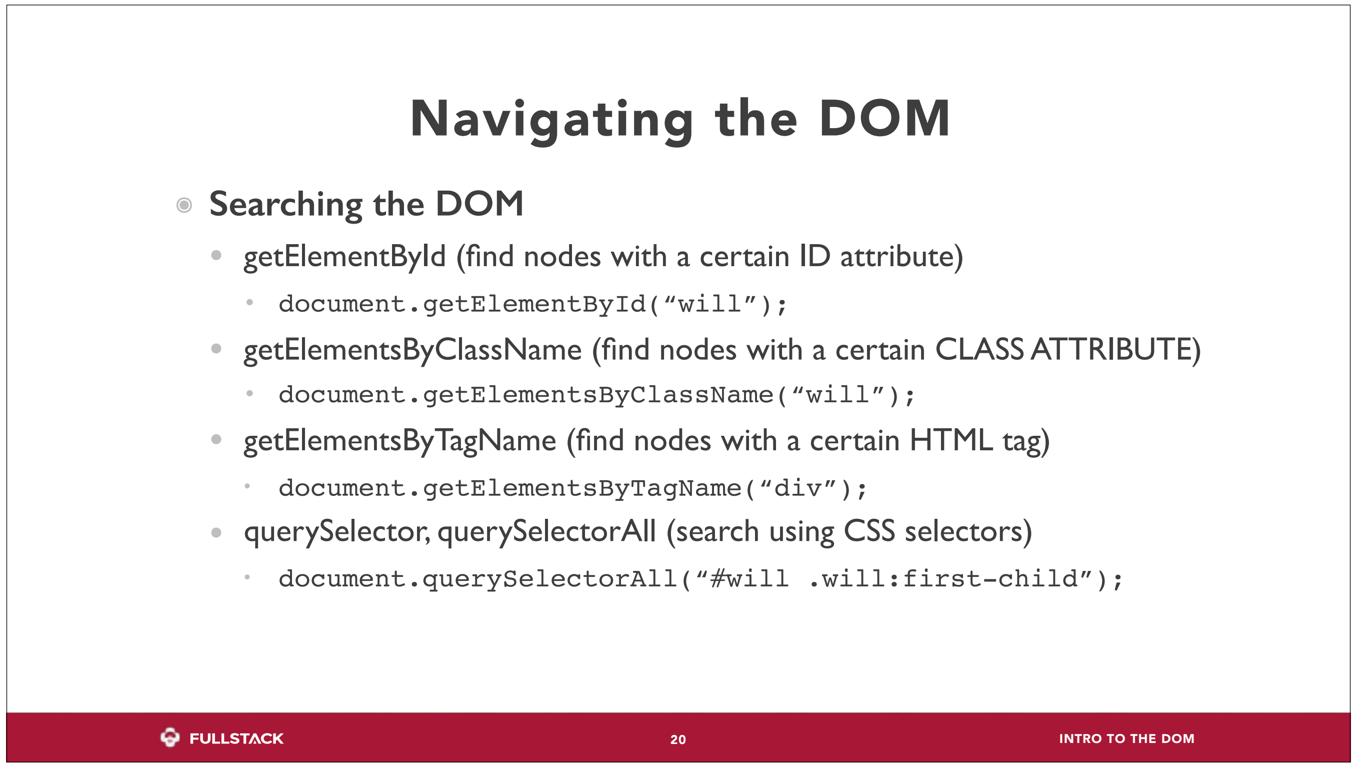
```css
h1 {
    font-weight: bold;
}

p {
    text-align: center;
}
```

```html
<script>
var paragraphs =
document.getElementsByTagN
ame("p");
paragraphs.forEach(functio
n(paragraph) {

paragraph.style.textAlign
= "center";
})
</script>
```

the **document** object is what connects these two things

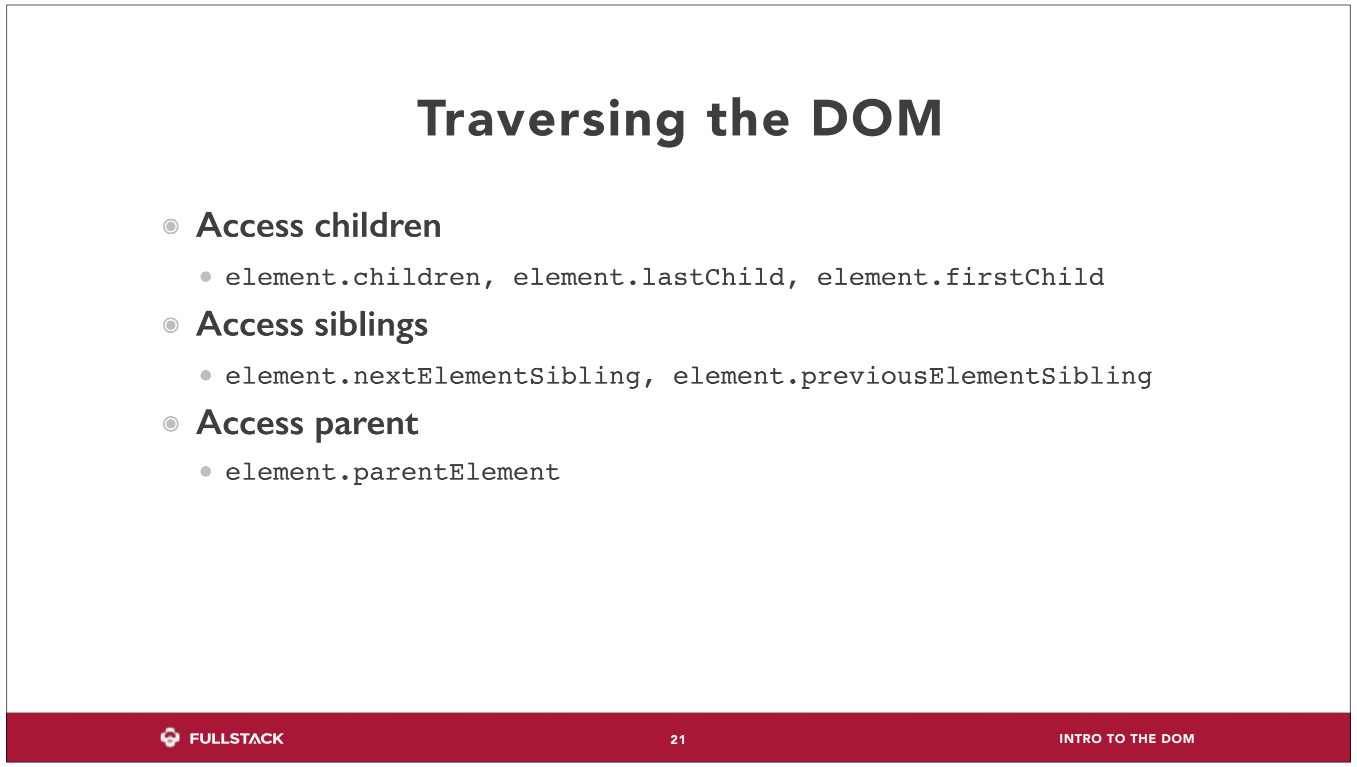


Navigating the DOM

- Searching the DOM
  - getElementById (find nodes with a certain ID attribute)
    - `document.getElementById(“will”);`
  - getElementsByClassName (find nodes with a certain CLASS ATTRIBUTE)
    - `document.getElementsByClassName(“will”);`
  - getElementsByTagName (find nodes with a certain HTML tag)
    - `document.getElementsByTagName(“div”);`
  - querySelector, querySelectorAll (search using CSS selectors)
    - `document.querySelectorAll(“#will .will:first-child”);`

One thing to be careful about is that getElementById returns the first Element it finds. That's why you don't want to have multiple IDs on a page.

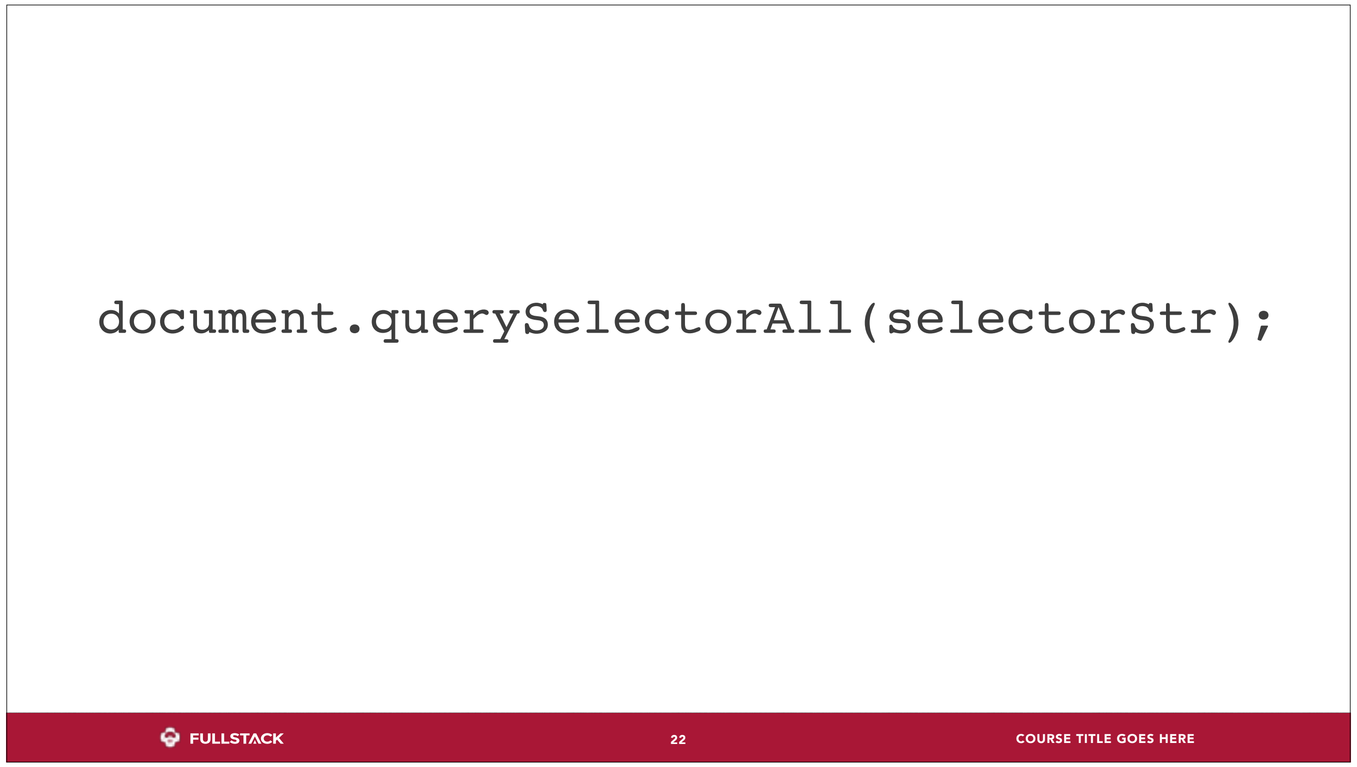
getElements... returns an HTMLCollection (array-like object) of Elements, even if there is only one element with that class or tag. Then you can use your JavaScript to manipulate all of the elements.



Traversing the DOM

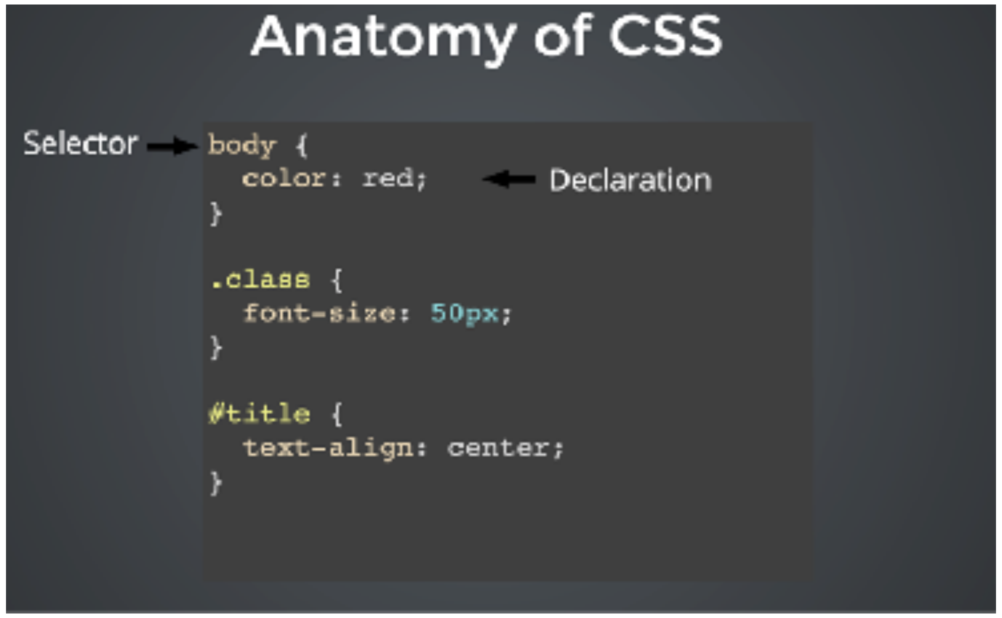
- Access children
  - `element.children, element.lastChild, element.firstChild`
- Access siblings
  - `element.nextElementSibling, element.previousElementSibling`
- Access parent
  - `element.parentElement`

Go to DOM now - show how querySelectorAll is very similar to $



```
document.querySelectorAll(selectorStr);
```

CSS selector – what do I mean by that?



Anatomy of CSS

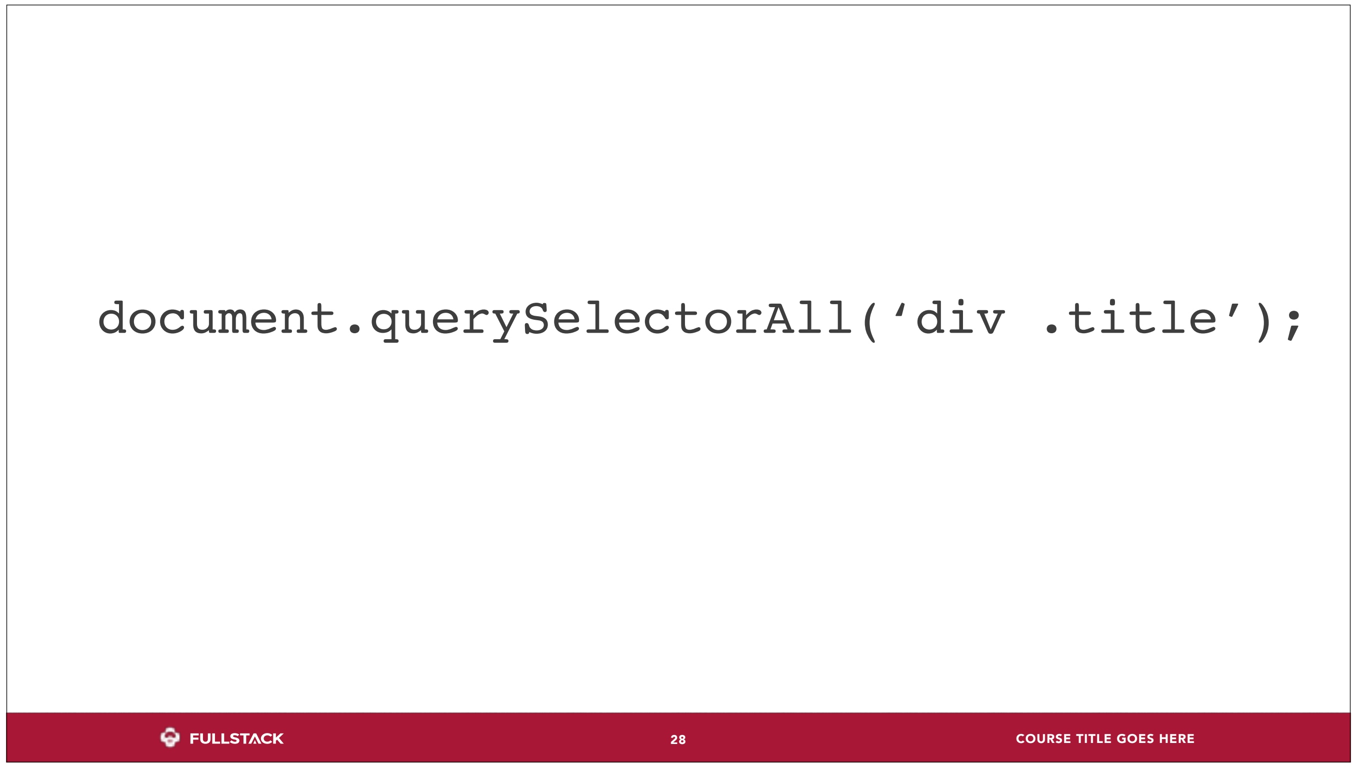
```css
body {
  color: red;
}

.class {
  font-size: 50px;
}

#title {
  text-align: center;
}
```

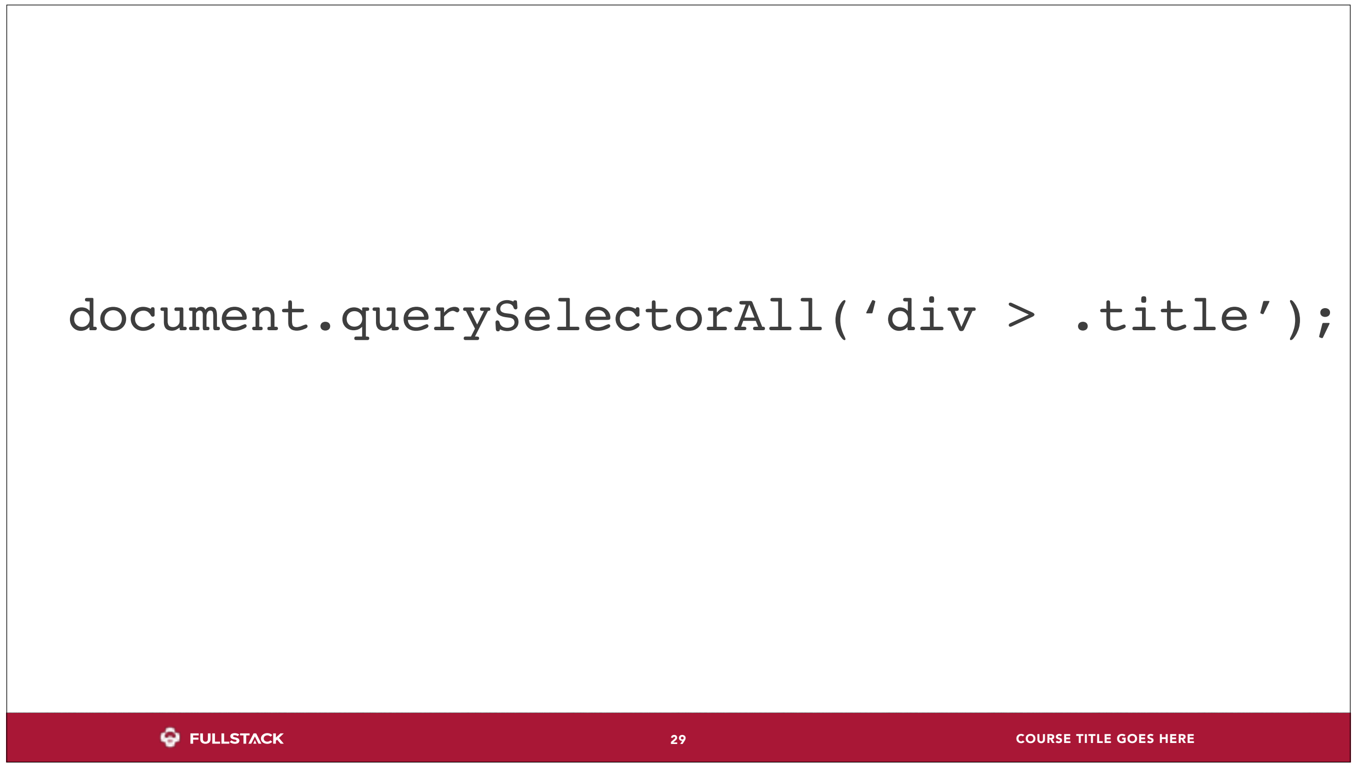
Pop Quiz!!!!

```
document.querySelectorAll(‘#title’);
document.querySelectorAll('.title');
document.querySelectorAll(‘div.title’);
```



```
document.querySelectorAll(‘div .title’);
```

all titles that are within a div



```
document.querySelectorAll('div > .title');
```

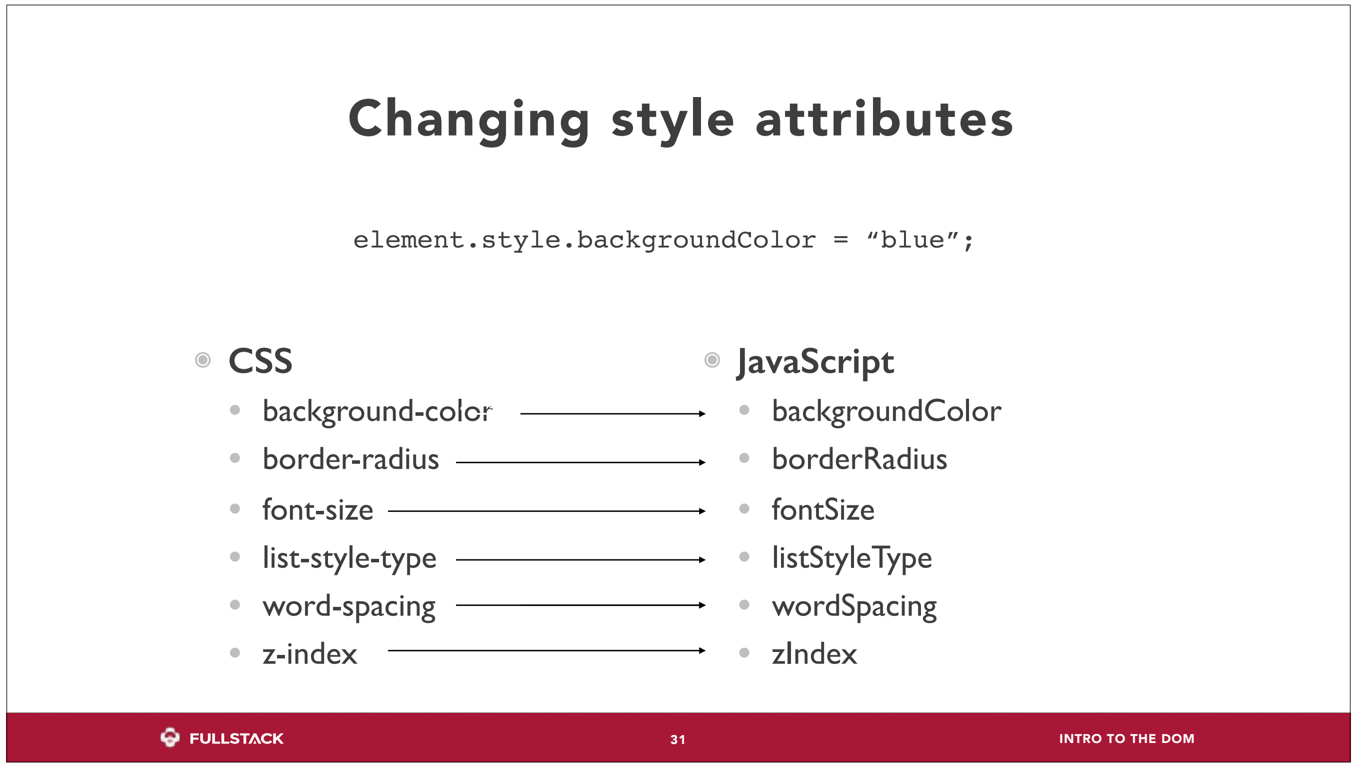
titles who's parent is a div

What does this remind you of? Jquery!

Let's take a look at what we can do in Chrome dev tools!

Manipulating the DOM

- **Changing Attributes for Style**
  - User Agent Stylesheet
  - Paint and Render Cycles
- **Making Elements**
- **Putting them into the DOM**
- **Remove Elements**
- **innerHTML and the DOM HTML Reader**



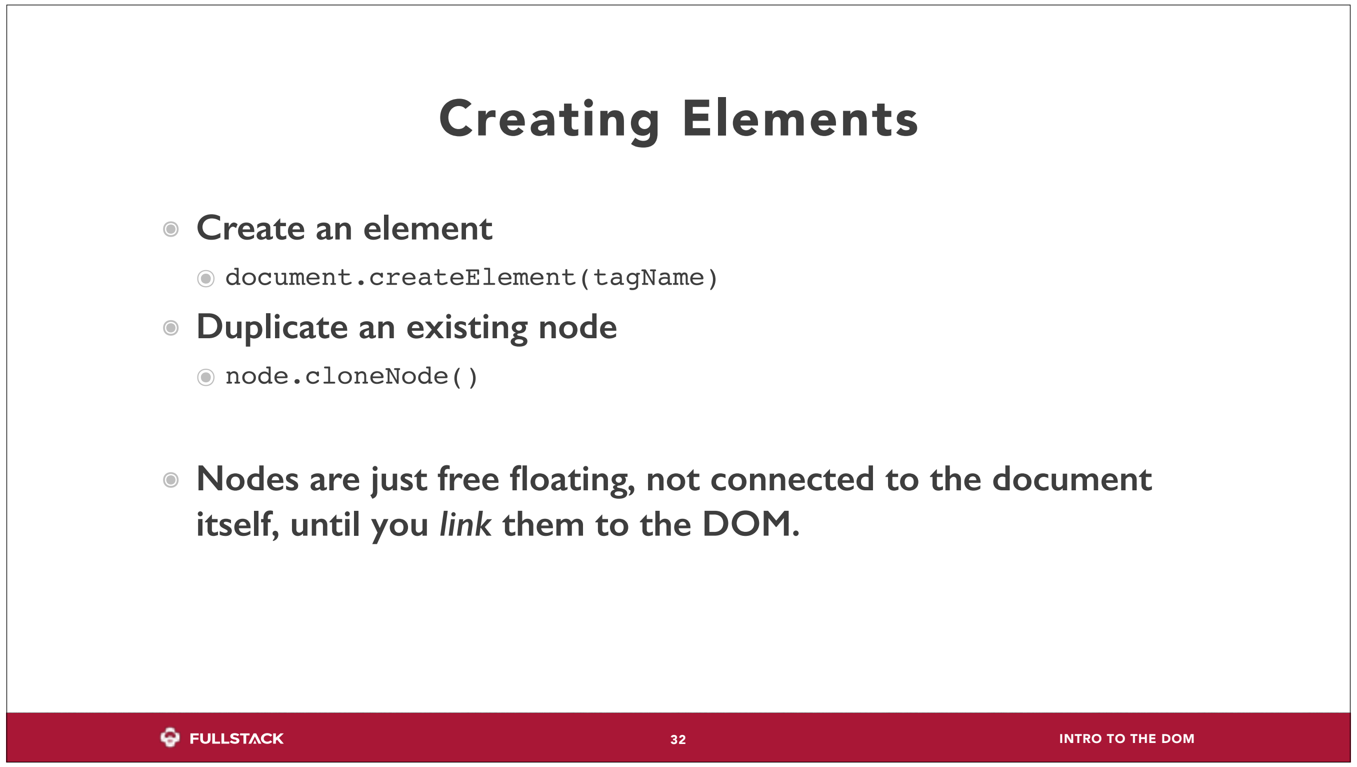
Changing style attributes

```
element.style.backgroundColor = “blue”;
```

CSS		JavaScript
background-color	$\rightarrow$	backgroundColor
border-radius	$\rightarrow$	borderRadius
font-size	$\rightarrow$	fontSize
list-style-type	$\rightarrow$	listStyleType
word-spacing	$\rightarrow$	wordSpacing
z-index	$\rightarrow$	zIndex

We can change the built in attributes either via CSS or JavaScript

Notice that one uses camelCase and the other uses dashed



Creating Elements

- Create an element
  - `document.createElement(tagName)`
- Duplicate an existing node
  - `node.cloneNode()`

- Nodes are just free floating, not connected to the document itself, until you *link* them to the DOM.

Examples:
- create a div
- first create it, then add it.

Adding elements to the DOM

- Insert newNode at end of current node
  - `node.appendChild(newNode);`
- Insert newNode at end of current node
  - `node.prependChild(newNode);`
- Insert newNode before a certain childNode
  - `node.insertBefore(newNode, sibling);`

Removing Elements

- **Removes the oldNode child.**
  - `node.removeChild(oldNode);`

- **Quick hack:**
  - oldNode.parentNode.removeChild(oldNode);