CSET - 105
Lecture 8: Document Object Model (DOM)

Mohammad Ashraful Huq

DOM

- There is one **functionality** JavaScript has, which is the **DOM**.
- The DOM is a vital aspect of JavaScript programming that shouldn’t be neglected.
- JavaScript is widely growing and becoming versatile the use of the DOM becomes more essential and strongly needed.

DOM

- JavaScript can be used to interact with the DOM, allowing developers to create **dynamic web pages** that can update and respond to user input.

- As this is also kind of abstract, I like to refer to the DOM as the "**bridge**" between your **JavaScript and HTML code**.

- The JavaScript DOM allows you to manipulate and access the elements, attributes, and content of an HTML or XML document, changing the structure, layout, and content of a **web page without reloading it**.

DOM

- The **Document Object Model (DOM) {template}** is a **programming interface** for HTML and XML documents.
- It represents the page so that programs can **access and change** the **document structure**, **style**, and **content**.
- The DOM represents the document as **nodes** and **objects (**element**)**. That way, programming languages can **connect** to the page.
- Basically, when a browser loads a page it creates an **object model** of that page and prints it on the screen.

JavaScript-HTML DOM Methods

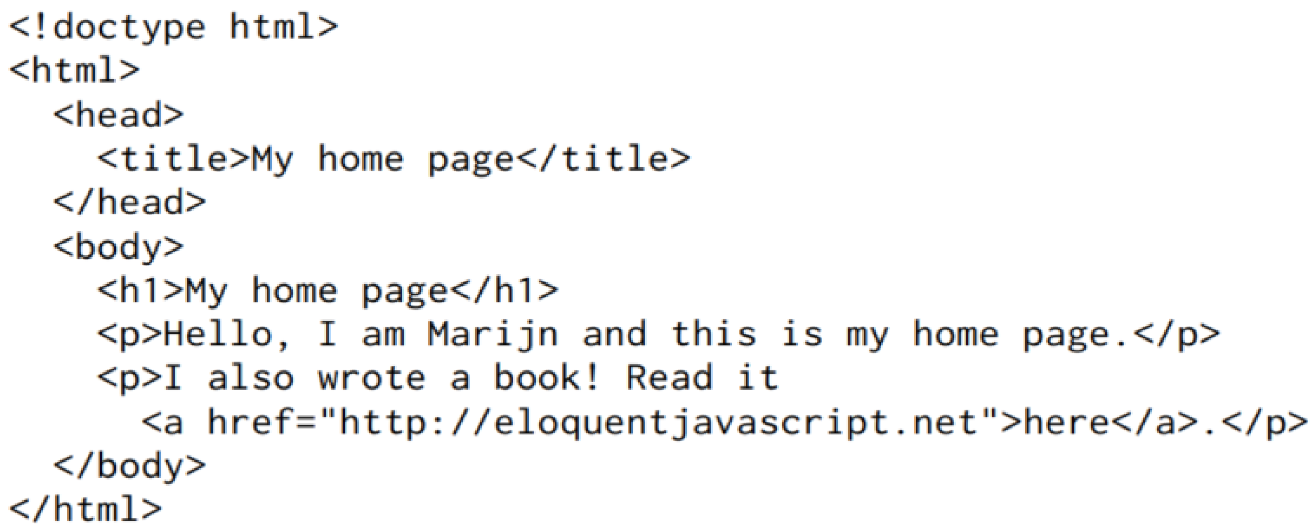
1. HTML DOM methods are actions you can perform (on HTML Elements);

2. HTML DOM properties are values (of HTML Elements) that you can set or change.

Click

DOM

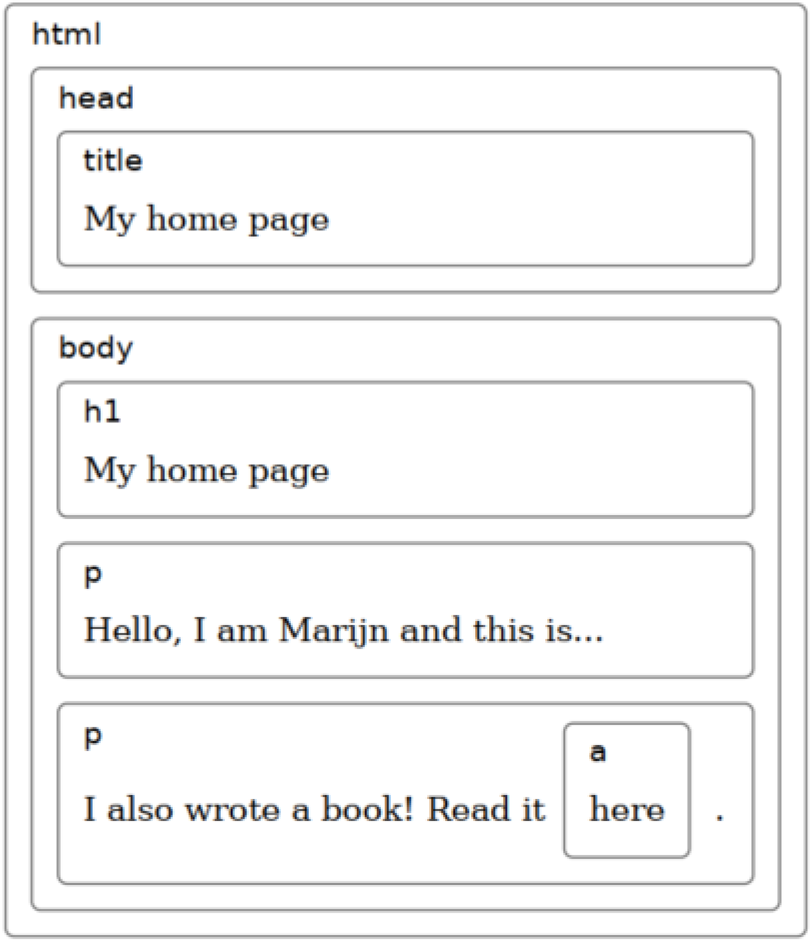
- **The object model is represented in a tree data structure**,
- *each* ***node is an object with properties and methods****, and the topmost node is the document object*. Consider this -



```
<!doctype html>
<html>
  <head>
    <title>My home page</title>
  </head>
  <body>
    <h1>My home page</h1>
    <p>Hello, I am Marijn and this is my home page.</p>
    <p>I also wrote a book! Read it
      <a href="http://eloquentjavascript.net">here</a>.</p>
  </body>
</html>
```

DOM

- The page has the following structure -

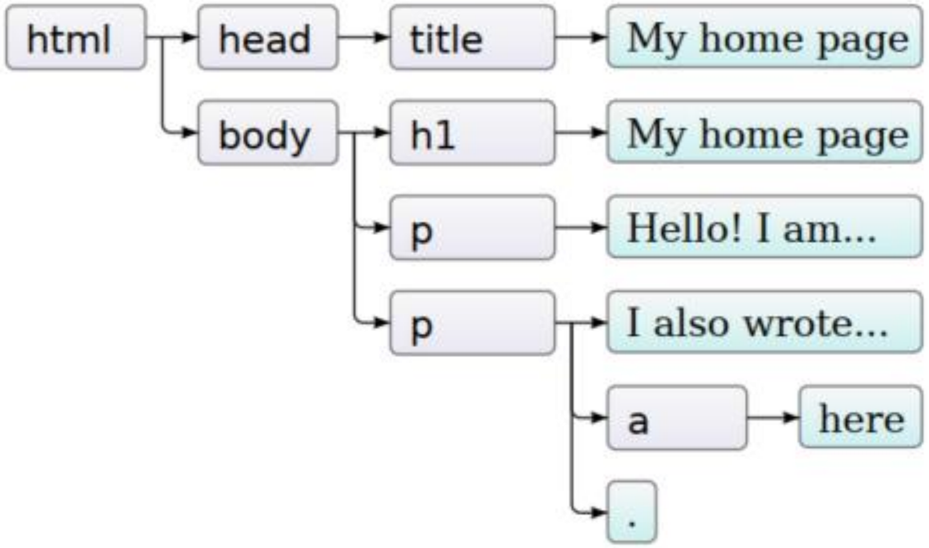

html
head
title
My home page
body
h1
My home page
p
Hello, I am Marijn and this is...
p
I also wrote a book! Read it
a
here
.

DOM

- The **data structure,** the browser uses to represent the document follows this shape.
- For each box, there is an object, which we can interact with to find out things such as what **HTML tag** it represents and which boxes and **text** it contains. This representation is called the **Document Object Model**, or DOM for short.
- The global binding **document** gives us access to these objects.

DOM

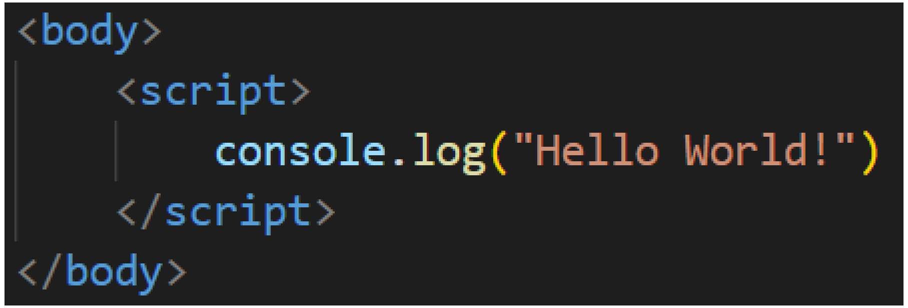
- Another way to visualize our document is as a tree -



Running JS

Running JS

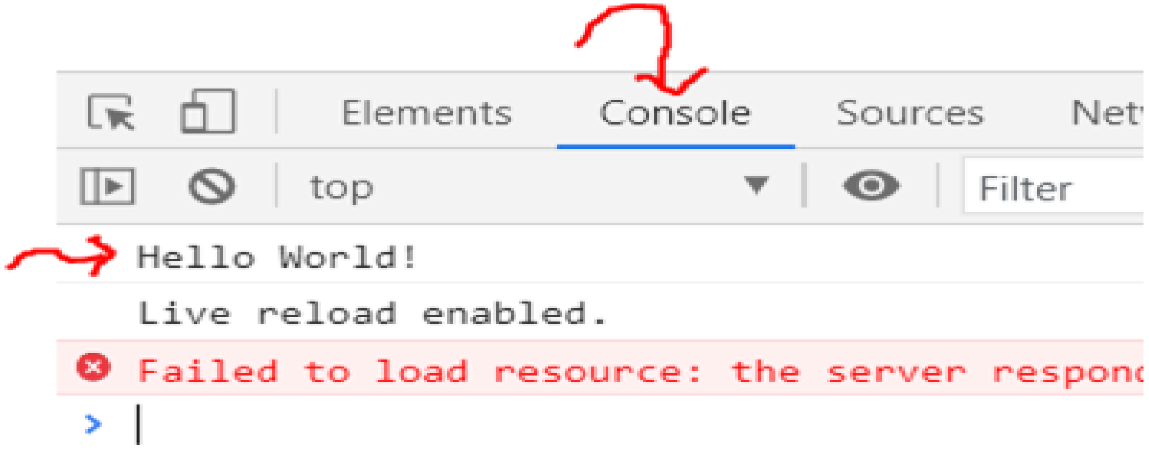
- Before we try manipulating the DOM, let’s see how we can run JS in the browser.
- In VS code, type doc and hit enter. Emmet abbreviations will insert the boilerplate code for you.
- This is pure HTML till now. To write JS code -



```
<body>
    <script>
        console.log("Hello World!")
    </script>
</body>
```

Running JS

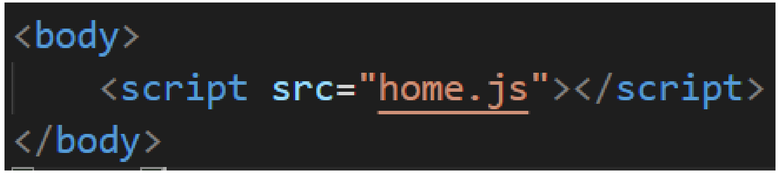
- To see its output, open the file in chrome and press **control+shift+i**
- Click on console to see your output



```
Hello World!
Live reload enabled.
Failed to load resource: the server respond
>
```

Running JS

- If you have to write a lot of JS code, then it is recommended to include it from **external file**.
- Copy your JS code to home.js (or any other file.js)
- And link the JS file like this -



```
<body>
    <script src="home.js"></script>
</body>
```

Manipulating DOM

Finding elements

- We can find elements from HTML body by their tag name. Try this out in VS Code -

```
let link = document.body.getElementsByTagName("a")[0];
console.log(link.href);
```

- To find a specific single node, you can give it an id attribute and use **document.getElementById** instead

```
<p>My ostrich Gertrude:</p>
<p><img id="gertrude" src="img/ostrich.png"></p>

<script>
  let ostrich = document.getElementById("gertrude");
  console.log(ostrich.src);
</script>
```

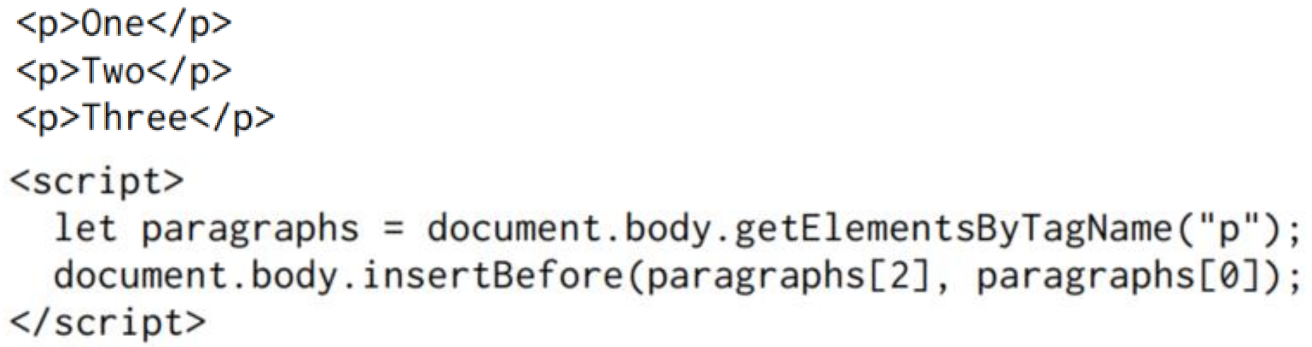
Finding elements

- A third, similar method is **getElementsByClassName**, which, like **getElementsByTagName**, searches through the contents of an element node and retrieves all elements that have the given string in their class attribute.

Changing the document

- Almost everything about the DOM data structure can be changed (ex: gamming score situation).
- Nodes have a **remove method** to remove them from their current parent node.
- To add a child node to an element node, we can use **appendChild**, which puts it at the end of the list of children, or
- **insertBefore**, which inserts the node given as the **first argument** before the node given as the **second argument**.

Changing the document



```
<p>One</p>
<p>Two</p>
<p>Three</p>

<script>
  let paragraphs = document.body.getElementsByTagName("p");
  document.body.insertBefore(paragraphs[2], paragraphs[0]);
</script>
```

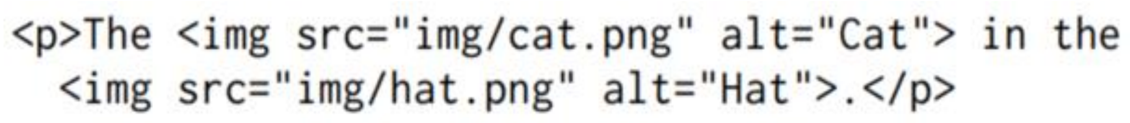
- **A node can exist in the document in only one place**. Thus, inserting paragraph Three in front of paragraph One will first remove it from the end of the document and then insert it at the front, resulting in Three/One/Two.

Changing the document

- All operations that insert a node somewhere will, **as a side effect**, cause it to be removed from its current position (if it has one).
- The **replaceChild method** is used to replace a child node with another one. It takes as arguments two nodes: a new node and the node to be replaced.
- The replaced node must be a child of the element the method is called on (**parent node**). Note that both **replaceChild** and **insertBefore** expect the new node as their first argument.

Creating nodes

- Consider the HTML -

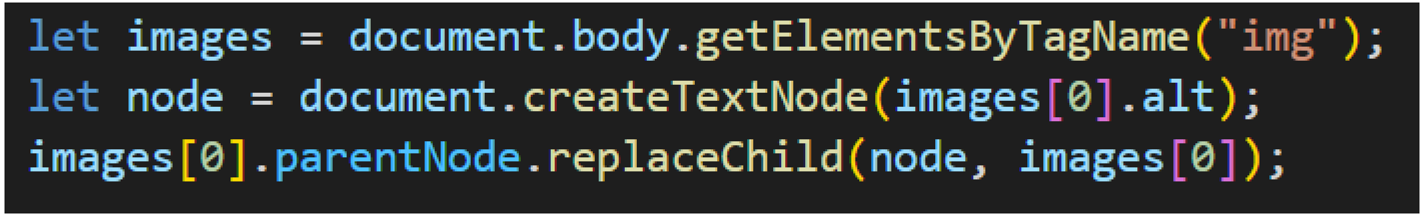


```
<p>The <img src="img/cat.png" alt="Cat"> in the
  <img src="img/hat.png" alt="Hat">.</p>
```

- Say we want to write a **script** that replaces the first image (<img> tag) in the document with the text held in its **alt attributes**, which specifies **an alternative textual** representation of the image.

Creating nodes

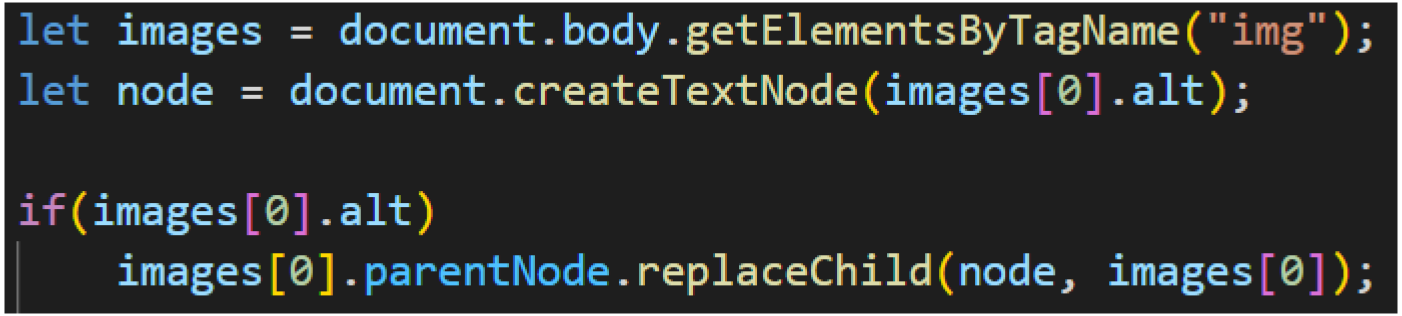
- This involves not only removing the images but adding or creating a new text node to replace them.
- Text nodes can be created with the **document.createTextNode** method.
- Here is how -



```
let images = document.body.getElementsByTagName("img");
let node = document.createTextNode(images[0].alt);
images[0].parentNode.replaceChild(node, images[0]);
```

Creating nodes

- But it won’t be wise to do so if the image has no alt attribute.
- So we modify our code to this -



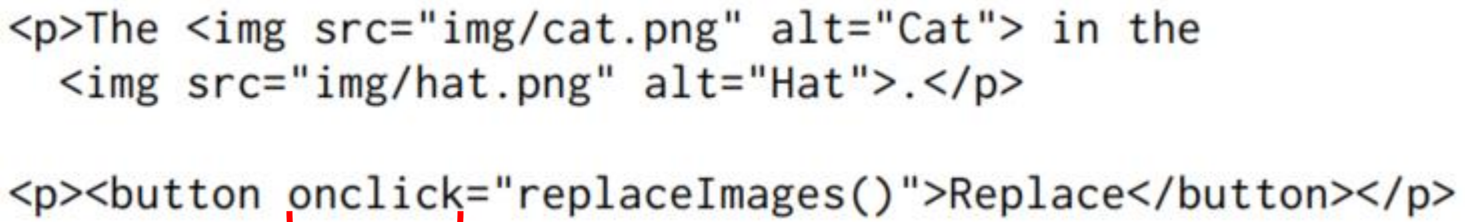
```
let images = document.body.getElementsByTagName("img");
let node = document.createTextNode(images[0].alt);

if(images[0].alt)
    images[0].parentNode.replaceChild(node, images[0]);
```

Creating nodes

- Now, let’s kick it up a notch! We want to do the same thing for every image **when a button is pressed**.
- In other words, when the button is pressed, we check every image to see if they have their alt attribute specified.
- If they do, then we replace them with that text. Else we leave them be.

Creating nodes

- The code -

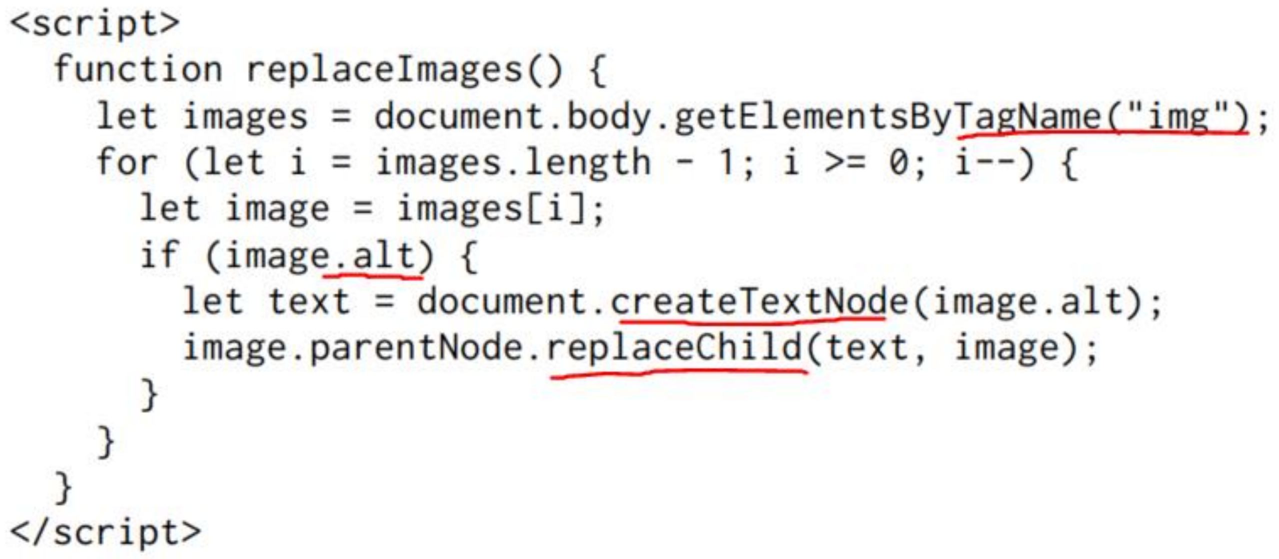


```
<p>The <img src="img/cat.png" alt="Cat"> in the
  <img src="img/hat.png" alt="Hat">.</p>

<p><button onclick="replaceImages()">Replace</button></p>
```

attribute

We are changing (controlling) the behavior of webpage without reloading (…from server).

Creating nodes



```
<script>
  function replaceImages() {
    let images = document.body.getElementsByTagName("img");
    for (let i = images.length - 1; i >= 0; i--) {
      let image = images[i];
      if (image.alt) {
        let text = document.createTextNode(image.alt);
        image.parentNode.replaceChild(text, image);
      }
    }
  }
</script>
```

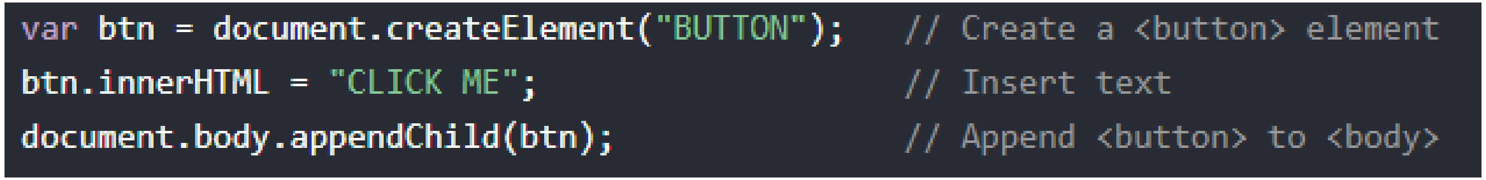
- It is extremely important to operate on the array backwards, from the end to the start. Can you guess why?

Creating nodes

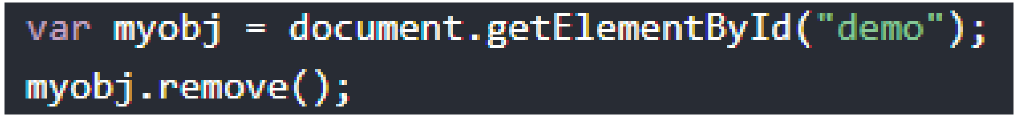
- Given a string, **createTextNode** gives us a text node that we can insert into the document to make it show up on the screen.
- To create element nodes, you can use the **document.createElement** method. This method takes a tag name and returns a **new empty node** of the given type.
- You can use **node.innerHTML** to edit the HTML that goes inside the node.
- And finally you can add the node to a parent node by **node.appendChild** (new node).

Creating nodes

- Consider the JS code -



```
var btn = document.createElement("BUTTON");   // Create a <button> element
btn.innerHTML = "CLICK ME";                   // Insert text
document.body.appendChild(btn);               // Append <button> to <body>
```



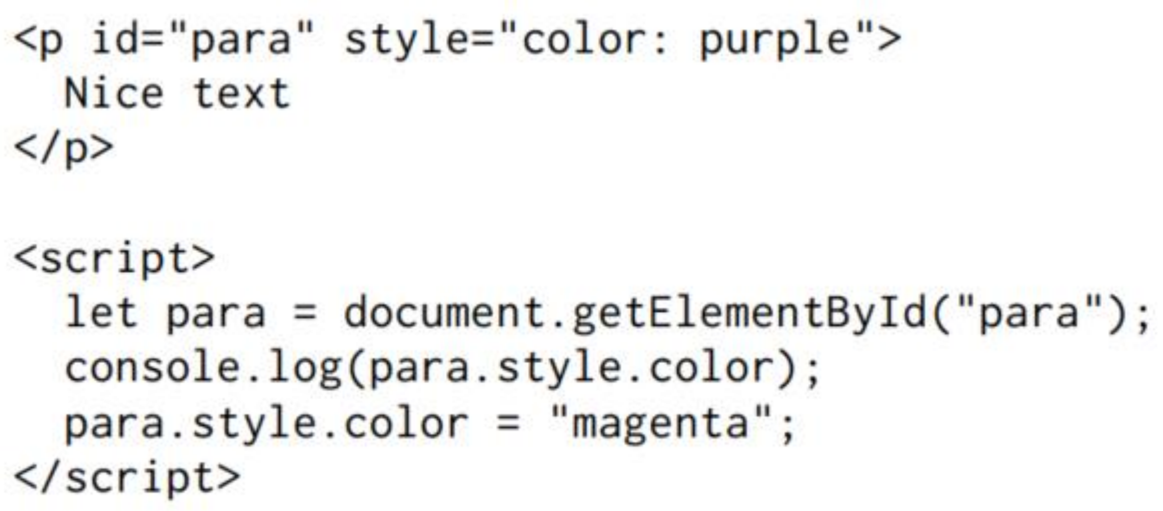
```
var myobj = document.getElementById("demo");
myobj.remove();
```

Manipulating style

- JavaScript code can directly manipulate the **style (.css)** of an element through the **element’s style property**.
- This property holds an <u>object</u> that has properties for all possible **style properties**.
- <u>The values of these properties are strings</u>, which we can write to in order to change a particular aspect of the element’s style.
- Some style property names contain **hyphens**, such as font-family. These property names in the style object for such properties have their **hyphens removed** and the letters after them capitalized (**style.fontFamily**).

Manipulating style

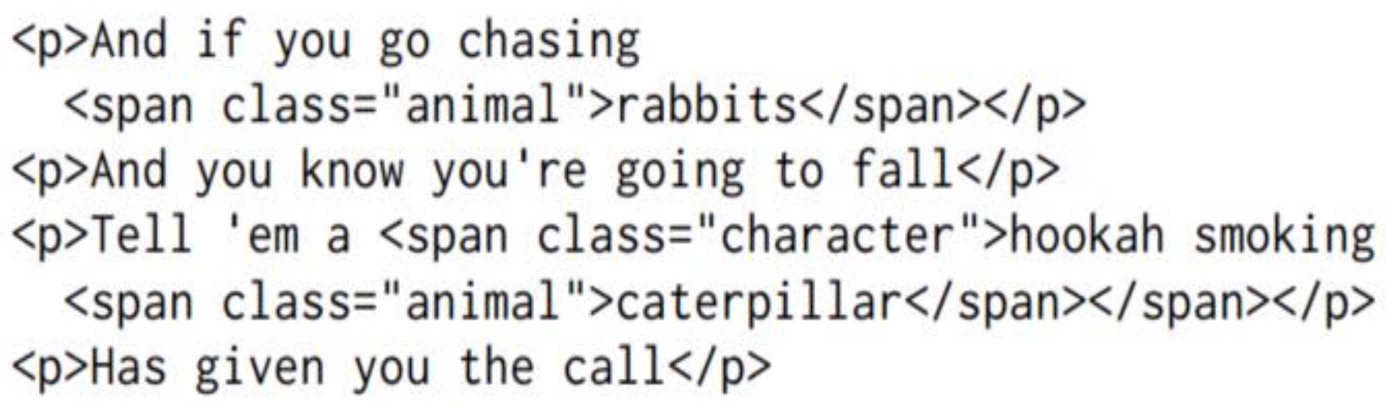
- Example -



```
<p id="para" style="color: purple">
  Nice text
</p>

<script>
  let para = document.getElementById("para");
  console.log(para.style.color);
  para.style.color = "magenta";
</script>
```

Queryselector

- The **querySelectorAll** method, which is defined both on the document object and on element nodes, takes a **selector string** and returns a **NodeList** containing all the elements that it matches. Example -



```html
<p>And if you go chasing
  <span class="animal">rabbits</span></p>
<p>And you know you're going to fall</p>
<p>Tell 'em a <span class="character">hookah smoking
  <span class="animal">caterpillar</span></span></p>
<p>Has given you the call</p>
```

Queryselector

```
<script>
  function count(selector) {
    return document.querySelectorAll(selector).length;
  }
  console.log(count("p"));           // All <p> elements
  // → 4
  console.log(count(".animal"));     // Class animal
  // → 2
  console.log(count("p .animal"));   // Animal inside of <p>
  // → 2
  console.log(count("p > .animal")); // Direct child of <p>
  // → 1
</script>
```

Attributes

- We have already seen that the attributes of a node can be accessed by **node.attribute**. If that attribute is not set, it gives an empty string -

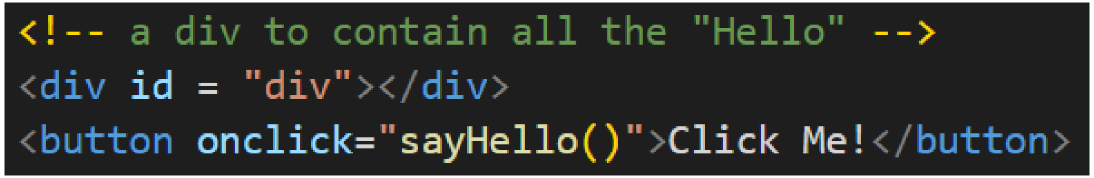
```
<a href="http://www.google.com">Google!</a>
```

```
// target attribute is not set. This prints true -
console.log(document.getElementsByTagName("a")[0].target === "");

document.getElementsByTagName("a")[0].target = "_blank";
// Now this prints _blank -
console.log(document.getElementsByTagName("a")[0].target);
```

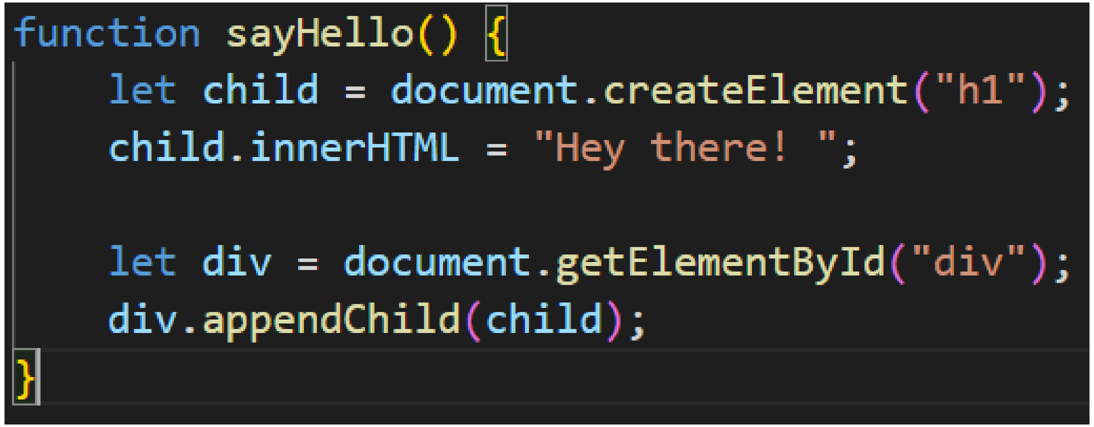
Adding interactivity

Interactivity

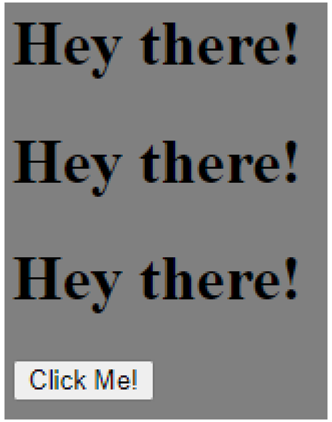
- Let’s say we wanna say hello every time a button is pressed.



```html
<!-- a div to contain all the "Hello" -->
<div id = "div"></div>
<button onclick="sayHello()">Click Me!</button>
```



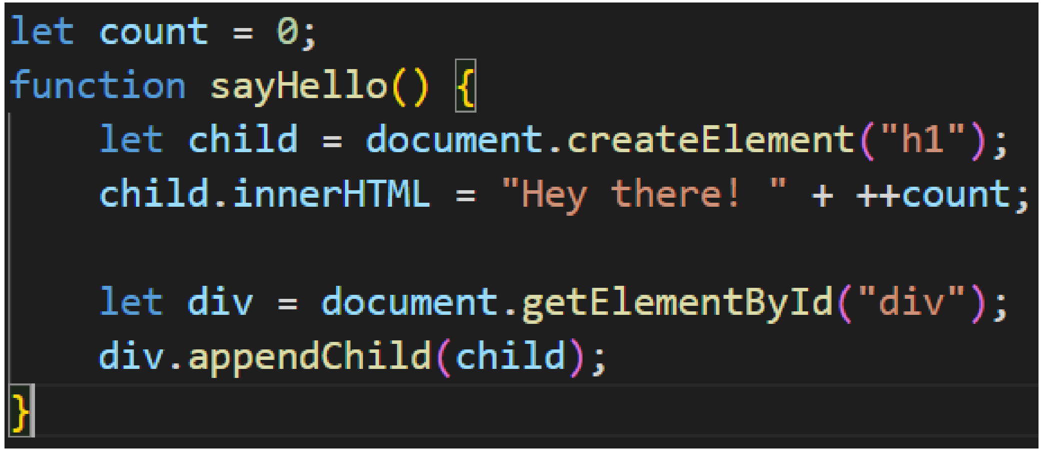
```javascript
function sayHello() {
    let child = document.createElement("h1");
    child.innerHTML = "Hey there! ";

    let div = document.getElementById("div");
    div.appendChild(child);
}
```



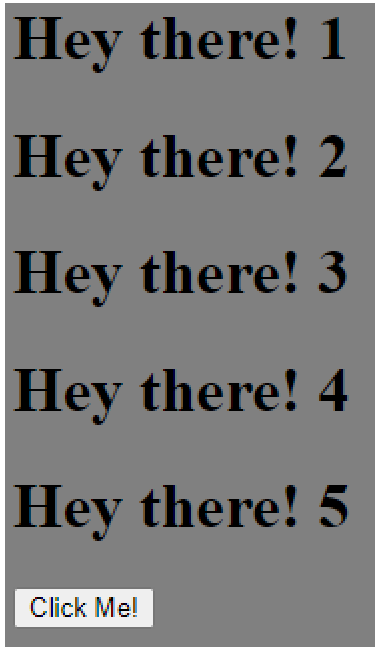
```
Hey there!

Hey there!

Hey there!
```

Interactivity

- Can we add a counter to the greeting message?



```
let count = 0;
function sayHello() {
    let child = document.createElement("h1");
    child.innerHTML = "Hey there! " + ++count;

    let div = document.getElementById("div");
    div.appendChild(child);
}
```



Hey there! 1

Hey there! 2

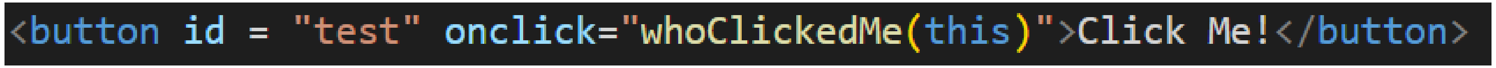
Hey there! 3

Hey there! 4

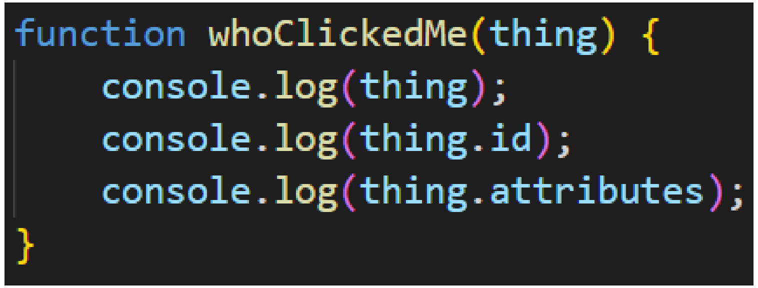
Hey there! 5

Interactivity

- If you want to know which button click called a JS function, you do it **this** way (pun definitely intended) -



```html
<button id = "test" onclick="whoClickedMe(this)">Click Me!</button>
```

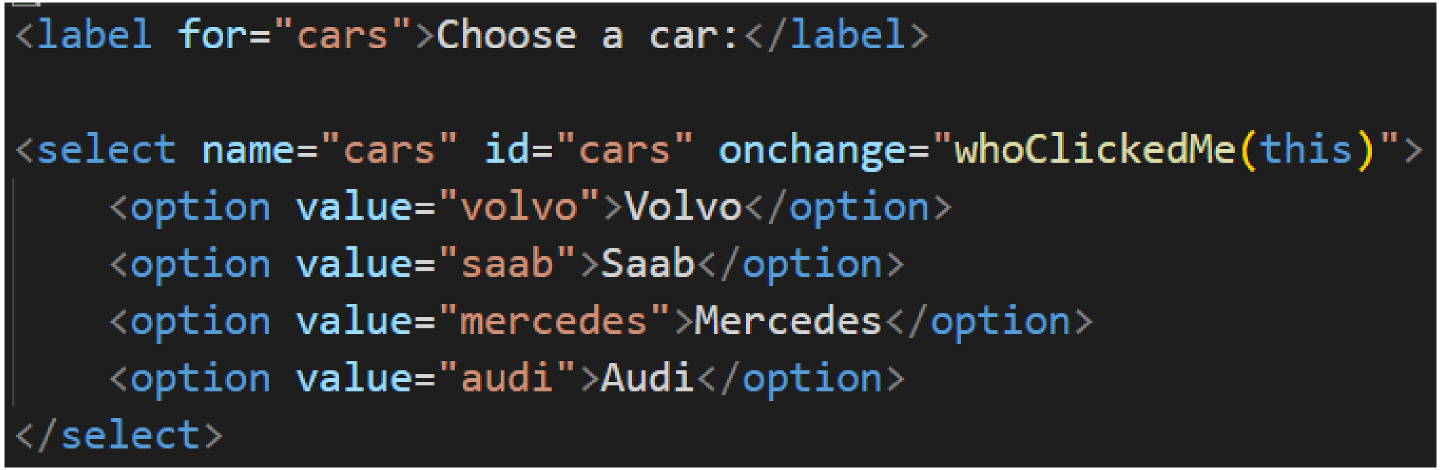


```javascript
function whoClickedMe(thing) {
    console.log(thing);
    console.log(thing.id);
    console.log(thing.attributes);
}
```

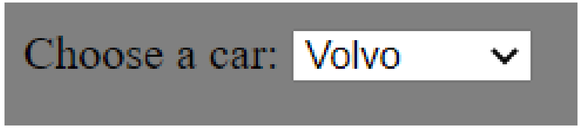
- You can use the onClick attribute on many other HTML elements

Interactivity

- For “select” element, you have an extra “onChanged” attribute.

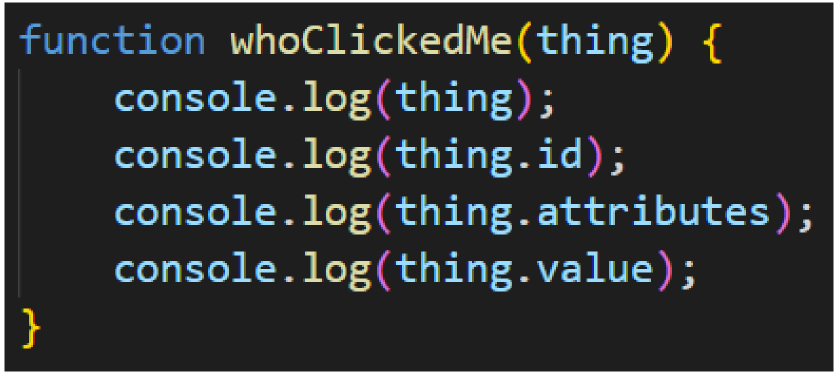


```
<label for="cars">Choose a car:</label>

<select name="cars" id="cars" onchange="whoClickedMe(this)">
    <option value="volvo">Volvo</option>
    <option value="saab">Saab</option>
    <option value="mercedes">Mercedes</option>
    <option value="audi">Audi</option>
</select>
```

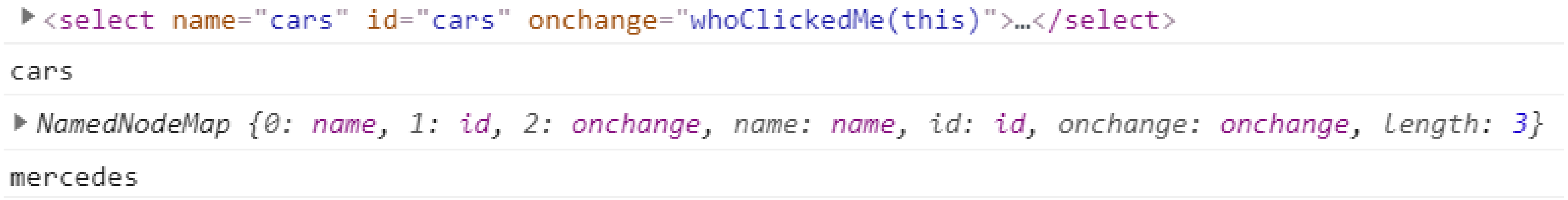


Interactivity

- The JS function and the output-



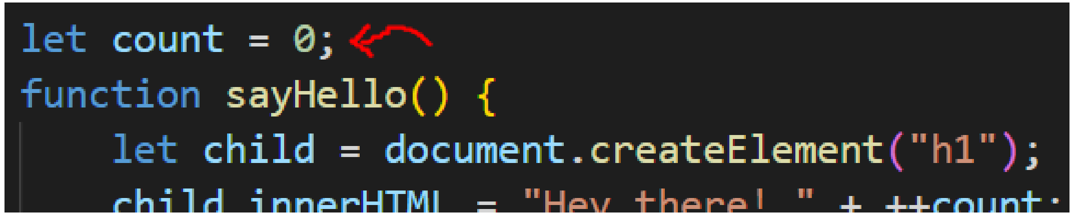
```
function whoClickedMe(thing) {
    console.log(thing);
    console.log(thing.id);
    console.log(thing.attributes);
    console.log(thing.value);
}
```



```
▶<select name="cars" id="cars" onchange="whoClickedMe(this)">…</select>
cars
▶NamedNodeMap {0: name, 1: id, 2: onchange, name: name, id: id, onchange: onchange, length: 3}
mercedes
```

Pro tips

- You can type JS code in the JS console in your browser to execute it dynamically.



```
let count = 0;
function sayHello() {
    let child = document.createElement("h1");
    child.innerHTML = "Hey there! " + ++count;
```

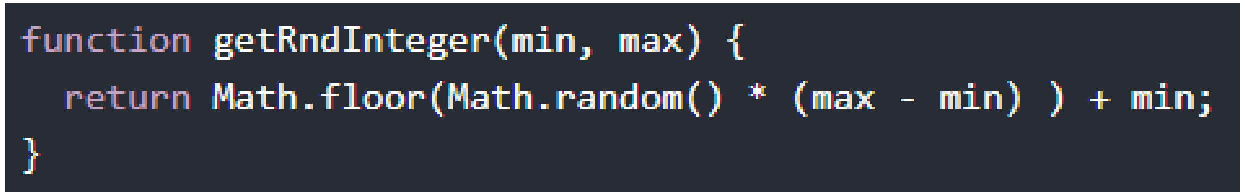
- For example, type count in the console to see its value. In the JS console, you don't need to type console.log(count). Typing *count* will give the same result.
- And don't forget how to take user input -
    - `var name = window.prompt("Enter your name: ");`

Pro tips

- Remind yourself on how to generate random numbers. Use Math.random() for this, which gives a random decimal between 0 (inclusive), and 1 (exclusive).
- Examples -
- Math.floor(Math.random() * 10);   // random integer from 0 to 9
- Math.floor(Math.random() * 101); // random integer from 0 to 100
- Math.floor(Math.random() * 10) + 1; //random int from 1 to 10

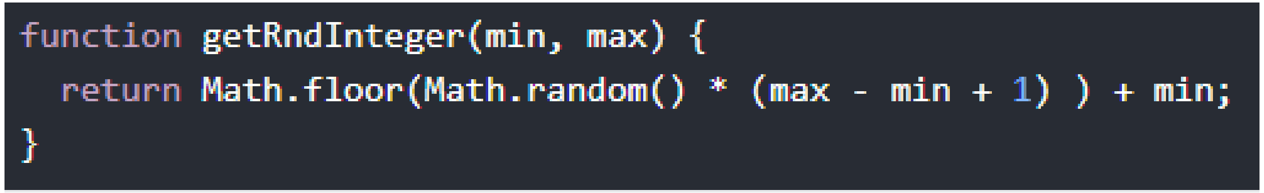
Pro tips

- A proper random function for a number between min (included) and max (excluded)



```
function getRndInteger(min, max) {
  return Math.floor(Math.random() * (max - min) ) + min;
}
```

- … and a number between min (included) and max (included)



```
function getRndInteger(min, max) {
  return Math.floor(Math.random() * (max - min + 1) ) + min;
}
```

Recommended Reading

Recommended Reading

Eloquent JavaScript, by Marijn Haverbeke

- Chapter 14

Thank you!