DOM

Document **O**bject **M**odel

A standard for how to get, change, add, or delete HTML elements.

What to expect?

- Quick introduction to DOM
- How the DOM represents an HTML document in memory?
- How to use APIs to create web content and applications

JavaScript makes the HTML page active and dynamic via the DOM.

You want a button to change colours when it gets clicked or an image to appear when the mouse hovers over text.

First, you need to reference those elements from your JavaScript.

Log out

What is DOM?

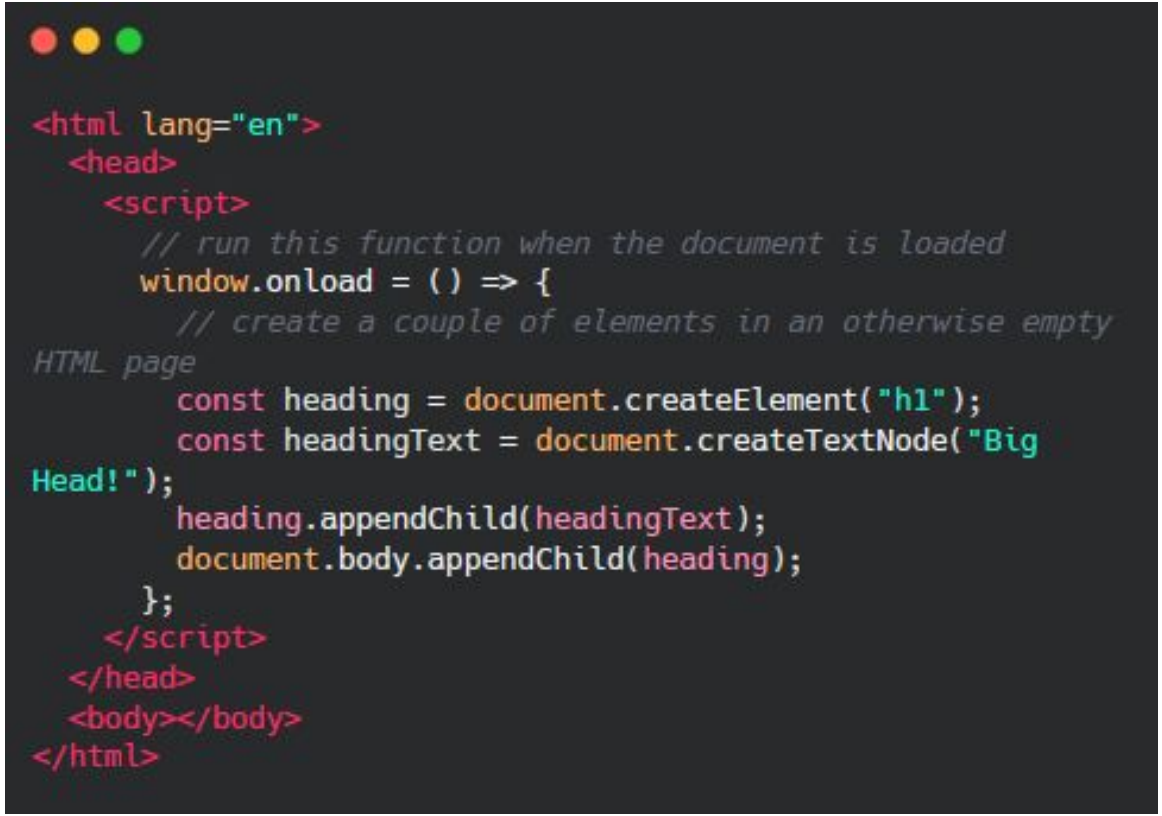
- A **programming interface** for web documents.
- DOM is not a programming language.
- Represents the page so that programs can change the document structure, style, and content.
- The DOM is a tree-like representation of the web page that gets loaded into the browser.
- The DOM represents the document as **nodes and objects.**
- Without it, the JavaScript language wouldn't have any model or notion of web pages, HTML documents, SVG documents, and their component parts.
- Web API used to build websites

Accessing the DOM

When you create a script, whether inline in a <script> element or included in the web page, you can immediately begin using the API for the document or window objects to manipulate the document itself.

The DOM was designed to be independent of any particular programming language, making the structural representation of the document available from a single, consistent API

Let’s look at an example:



```
<html lang="en">
  <head>
    <script>
      // run this function when the document is loaded
      window.onload = () => {
        // create a couple of elements in an otherwise empty
HTML page
        const heading = document.createElement("h1");
        const headingText = document.createTextNode("Big
Head!");
        heading.appendChild(headingText);
        document.body.appendChild(heading);
      };
    </script>
  </head>
  <body></body>
</html>
```

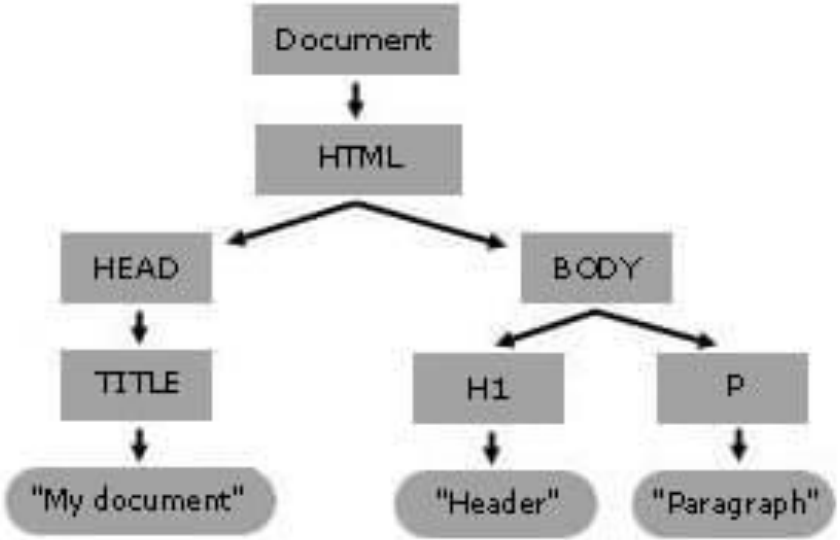
DOM contd

It represents the web page using a series of objects. The main object is the document object, which in turn houses other objects which also house their own objects, and so on.

```
<html lang="en">
  <head>
    <title>My Document</title>
  </head>
  <body>
    <h1>Header</h1>
    <p>Paragraph</p>
  </body>
</html>
```

DOM Tree

The DOM is a tree-like representation of the web page that gets loaded into the browser.

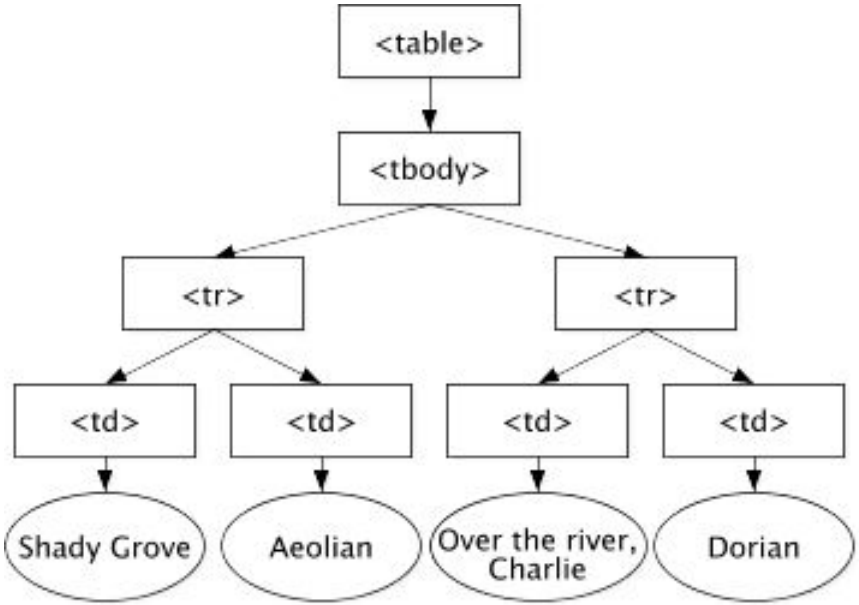


When a web browser parses an HTML document, it builds a DOM tree and then uses it to display the document.

Let’s look at the different objects

The Document Object

This is the top most object in the DOM. It has properties and methods which you can use to get information about the document using a rule known as dot notation.



Example

```
<!DOCTYPE html>
<html>
<body>

<form id="LoginForm" action="/action_page.php">
  First name: <input type="text" name="fname"
value="Donald"><br>
  Last name: <input type="text" name="lname" value="Duck"><br>
  <input type="submit" value="Submit">
</form>

<p>Click the "Try it" button to display the number of elements
in the form.</p>

<button onclick="myFunction()">Try it</button>

<p id="displayspace"></p>

<script>
function myFunction() {
  var x =
document.getElementById("LoginForm").elements.length;
  var y =
document.getElementById("LoginForm").elements[0].value;
  document.getElementById("displayspace").innerHTML = "Found "
+ x + " elements in the form." + "first name entered is: " +
y;
}
</script>

</body>
</html>
```

The createElement() method

Create a specified element and insert it into the DOM:

```
const para = document.createElement("p");
para.innerText = "This is a paragraph created via DOM.";
document.body.appendChild(para);
```

Finding HTML Elements

If you want to manipulate HTML elements.

But we need to find the elements first. Multiple ways to do this:

- Finding HTML elements by id
- Finding HTML elements by tag name
- Finding HTML elements by class name
- Finding HTML elements by CSS selectors
- Finding HTML elements by HTML object collections

Finding HTML Elements

Method	Description
document.getElementById(*id*)	Find an element by element id
document.getElementsByTagName(*name*)	Find elements by tag name
document.getElementsByClassName(*name*)	Find elements by class name

getElementById()

Method to get an element from the document by its unique id attribute.

```
<div id='first'> Div one </div>
<p> Paragraph one </p>
<div> Div two </div>
<p> Paragraph two </p>
<div> Yet another div</div>
```

```
var firstDiv = getElementById("first")
```

getElementsByTagname()

Method to access one or more elements by their HTML tag name

```
divs = document.getElementByTagname("div");
```

```
<div> Div one </div>
<p> Paragraph one </p>
<div> Div two </p>
<p> Paragraph Two </p>
<div> Yet another div </div>
```

getElementByClassName()

```html
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTML DOM</h2>

<p>Finding HTML Elements by Class Name.</p>
<p class="intro">Hello World!</p>
<p class="intro">This example demonstrates the
<b>getElementsByClassName</b> method.</p>

<p id="demo"></p>

<script>
const x = document.getElementsByClassName("intro");
document.getElementById("demo").innerHTML =
'The first paragraph (index 0) with class="intro" is: ' +
x[0].innerHTML;
</script>

</body>
</html>
```

Finding HTML Elements by CSS Selectors : querySelectorAll)

Access one or more elements from the DOM that matches one or more CSS selectors.

```
<div> Div one </div>
<p> Paragraph one </p>
<div> Div two </p>
<p> Paragraph Two </p>
<div> Yet another div </div>
```

```
var paragraphs = document.querySelectorAll( 'p' );
paragraphs.forEach(paragraph => paragraph.display = 'none')
```

Finds all HTML elements that match a specified CSS selector (id, class names, types, attributes, values of attributes, etc), use the querySelectorAll() method.

```html
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTML DOM</h2>
<p>Finding HTML Elements Using <b>document.forms</b>.</p>

<form id="frm1" action="/action_page.php">
  First name: <input type="text" name="fname"
value="Donald"><br>
  Last name: <input type="text" name="lname" value="Duck">
<br><br>
  <input type="submit" value="Submit">
</form>

<p>These are the values of each element in the form:</p>

<p id="demo"></p>

<script>
const x = document.forms["frm1"];
let text = "";
for (let i = 0; i < x.length ;i++) {
  text += x.elements[i].value + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

Finding HTML Elements by HTML Object Collections

Difference Between an HTMLCollection and a NodeList

HTML collection	NodeList
getElementsByClassName() and getElementsByTagName() methods return a live HTMLCollection.	querySelectorAll() method returns a static NodeList.
A collection of document elements.	A collection of document nodes (element nodes, attribute nodes, and text nodes).
Items can be accessed by their name, id, or index number.	Items can only be accessed by their index number.
It is always a live collection. Example: If you add a element to a list in the DOM, the list in the HTMLCollection will also change.	Most often a static collection. Example: If you add a element to a list in the DOM, the list in NodeList will not change.

Let’s look at all the other elements you can see as well

- document.anchors
- document.body
- document.documentElement
- document.embeds
- document.forms
- Document.head
- Document.images
- Document.links
- Document.scripts
- document.title

Changing HTML Elements

Property	Description
element.innerHTML = *new html content*	Change the inner HTML of an element
element.attribute = new value	Change the attribute value of an HTML element
element.style.*property = new style*	Change the style of an HTML element
Method	**Description**
element.setAttribute*(attribute, value)*	Change the attribute value of an HTML element

Changing HTML Content

innerHTML property

document.getElementById(id).innerHTML = new HTML

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript can Change HTML</h2>

<p id="p1">Hello World!</p>

<script>
document.getElementById("p1").innerHTML = "New Paragraph text
generated";
</script>

<p>The paragraph above was changed by a script using DOM.</p>

</body>
</html>
```

Changing the Value of an Attribute

```html
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTML DOM example</h2>
<img id="image" src="smiley.gif" width="160" height="120">

<p>The original image was a .gif, but the script changed it to
an image with .jpg extension</p>

</body>
</html>
```

JavaScript HTML DOM example

The original image was a .gif, but the script changed it to an image with .jpg extension

```html
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTML DOM example</h2>
<img id="image" src="smiley.gif" width="160" height="120">

<script>
document.getElementById("image").src = "landscape.jpg";
</script>

<p>The original image was a .gif, but the script changed it to
an image with .jpg extension</p>

</body>
</html>
```

JavaScript HTML DOM example

The original image was a .gif, but the script changed it to an image with .jpg extension

Dynamic HTML content

document.write(): write directly to the HTML output stream

Never use document.write() after the document is loaded. It will overwrite the document.

The **setAttribute**() method sets a new value to an attribute.

If the attribute does not exist, it is created first.

element.setAttribute("style", "background-color:red;"); vs

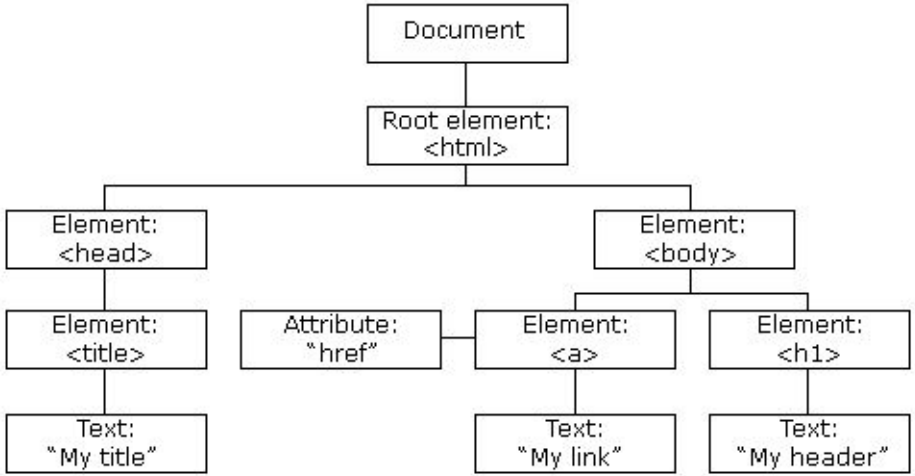
element.style.backgroundColor = "red";

Adding and Deleting Elements

Method	Description
document.createElement(*element*)	Create an HTML element
document.removeChild(*element*)	Remove an HTML element
document.appendChild(*element*)	Add an HTML element
document.replaceChild(*new, old*)	Replace an HTML element
document.write(*text*)	Write into the HTML output stream

DOM Nodes

- The entire document is a document node
- Every HTML element is an element node
- The text inside HTML elements are text nodes
- Every HTML attribute is an attribute node (deprecated)
- All comments are comment nodes

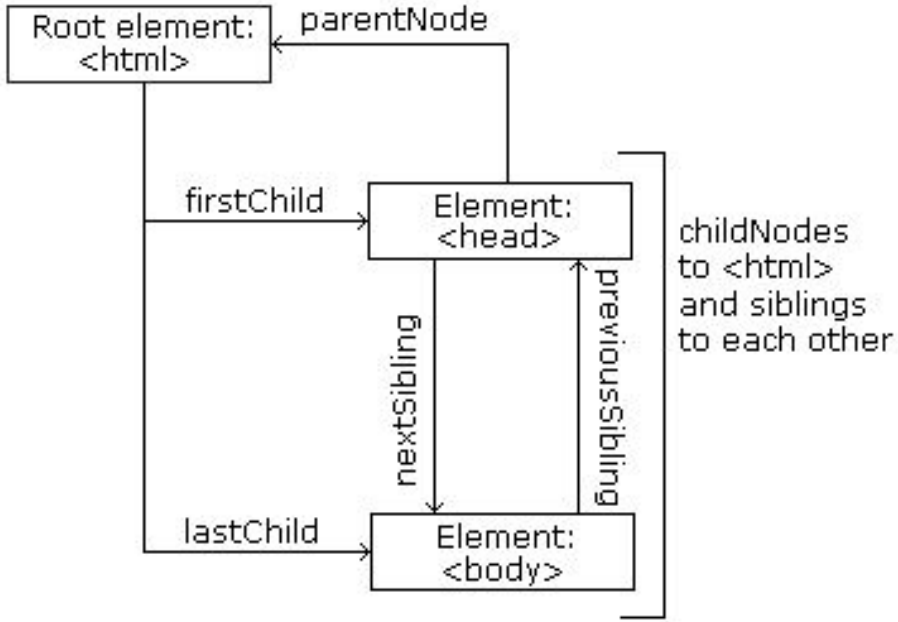


```
<html>

  <head>
    <title>DOM Tutorial</title>
  </head>

  <body>
    <h1>DOM Lesson one</h1>
    <p>Hello world!</p>
  </body>

</html>
```



The node method

You can access any element by using the following properties with the node object:

- node.childNodes – accesses the child nodes of a selected parent
- node.firstChild – accesses the first child of a selected parent
- node.lastChild – accesses the last child of a selected parent.
- node.parentNode – accesses the parent of a selected child node.
- node.nextSibling – accesses the next consecutive element (sibling) of a selected element.
- node.previousSibling – accesses the previous element (sibling) of a selected element

A **common error in DOM processing** is to expect an element node to contain text.

```
<title id="demo">DOM Tutorial</title>
```

```
myTitle = document.getElementById("demo").innerHTML;
```

```
myTitle = document.getElementById("demo").firstChild.nodeValue;
```

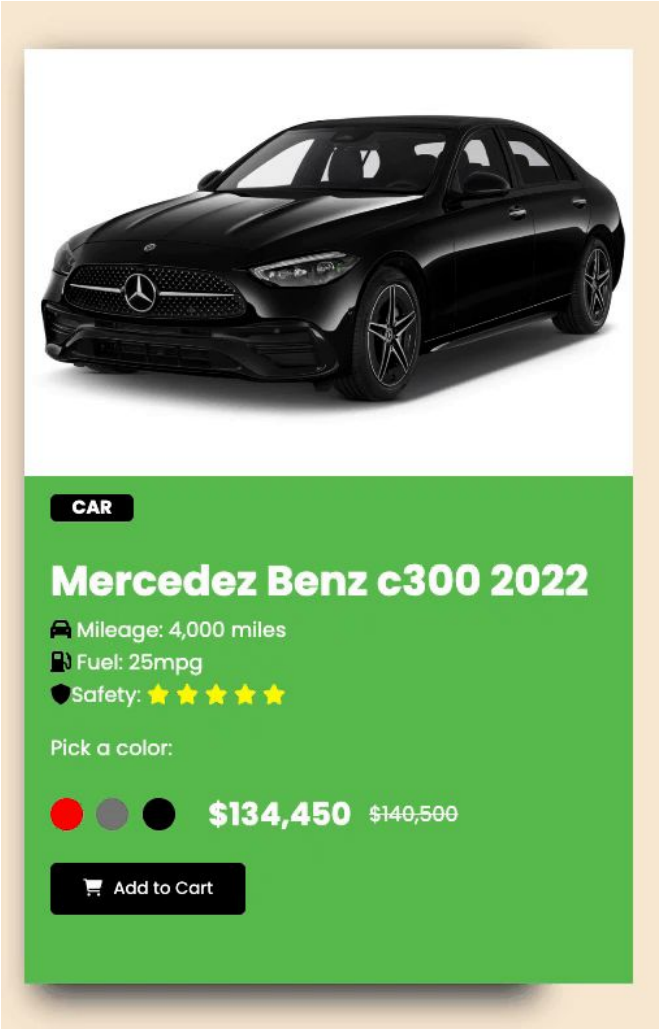
```
myTitle = document.getElementById("demo").childNodes[0].nodeValue;
```

Types of Nodes

Node	Type	Example
ELEMENT_NODE	1	<h1 class="heading">W3Schools</h1>
ATTRIBUTE_NODE	2	class = "heading" (deprecated)
TEXT_NODE	3	W3Schools
COMMENT_NODE	8	<!-- This is a comment -->
DOCUMENT_NODE	9	The HTML document itself (the parent of <html>)
DOCUMENT_TYPE_NODE	10	<!Doctype html>

DOM Events

HTML DOM allows JavaScript to react to HTML events:



A JavaScript can be executed when an event occurs, like when a user clicks on an HTML element.

To execute code when a user clicks on an element, add JavaScript code to an HTML event attribute:

onclick=JavaScript

Onload and Onunload functions

The onload and onunload events are triggered when the user enters or leaves the page.

The **onload** event can be used to check the visitor's browser type and browser version, and load the proper version of the web page based on the information.

The onload and onunload events can be used to deal with cookies.

DOM EventListener

- Attaches an event handler to the specified element (without overwriting existing event handlers).
- You can add many event handlers(even of the same type) to one element.
- You can add event listeners to any DOM object not only HTML elements. i.e the window object.
- The addEventListener() method makes it easier to control how the event reacts to bubbling.
- When using the addEventListener() method, the JavaScript is separated from the HTML markup, for better readability and allows you to add event listeners even when you do not control the HTML markup.
- You can easily remove an event listener by using the removeEventListener() method.

Event Bubbling or Event Capturing?

Two ways of event propagation in the HTML DOM - bubbling and capturing.

Event propagation is a way of defining the element order when an event occurs. If you have a <p> element inside a element, and the user clicks on the <p> element, which element's "click" event should be handled first?

In bubbling the inner most element's event is handled first and then the outer: the <p> element's click event is handled first, then the element's click event.

In capturing the outer most element's event is handled first and then the inner: the element's click event will be handled first, then the <p> element's click event.

```
addEventListener(event, function, useCapture);
```