JavaScript (JS)

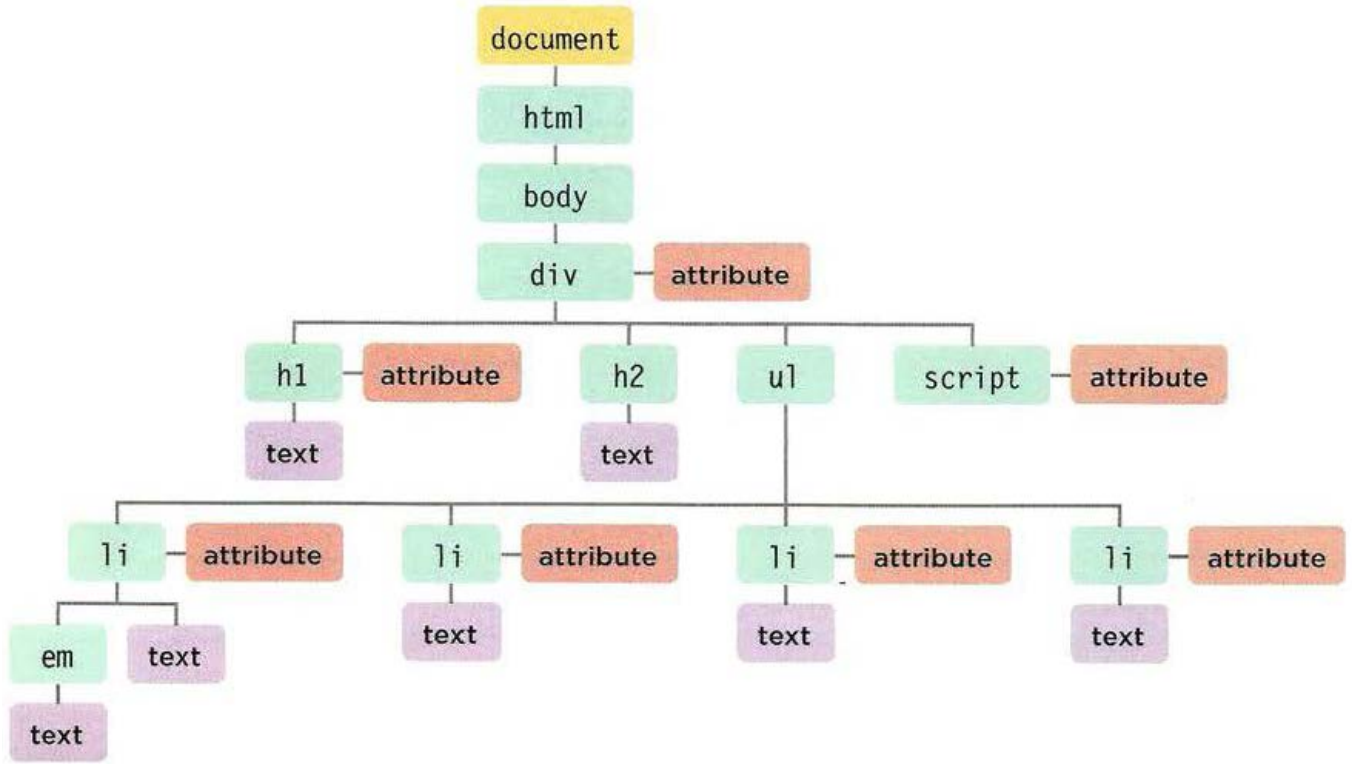
DOM Manipulation

What is the Document Object Model (DOM)?

“The **Document Object Model (*DOM*)** is a programming interface for HTML, XML and SVG documents. It provides a structured representation of the document (a **tree**) and it defines a way that the structure can be accessed from programs so that they can **change the document structure, style and content**.”

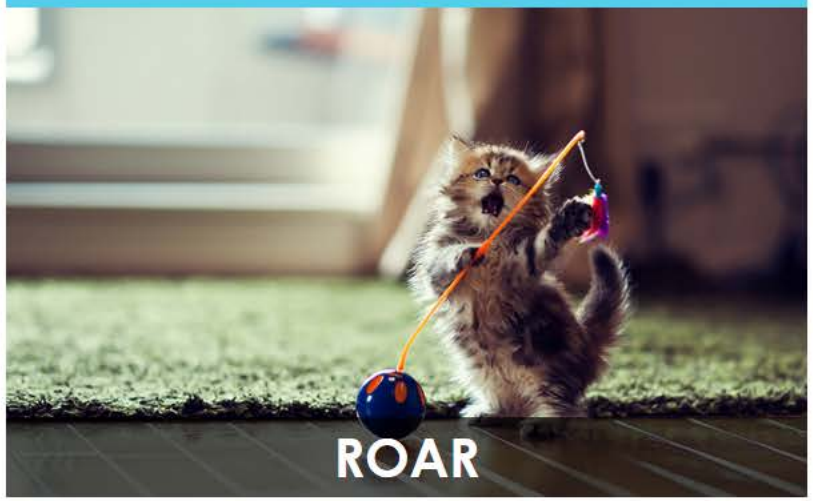
— MDN

```
<html>
  <body>
    <div id="page">
      <h1 id="header">List</h1>
      <h2>Buy groceries</h2>
      <ul>
        <li id="one" class="hot"><em>fresh</em> figs</li>
        <li id="two" class="hot">pine nuts</li>
        <li id="three" class="hot">honey</li>
        <li id="four">balsamic vinegar</li>
      </ul>
      <script src="js/list.js"></script>
    </div>
  </body>
</html>
```



Source: JavaScript & jQuery by Jon Duckett

HELLO, THIS IS THE TITLE.



If you are looking for more kittens, check out
reddit.com/r/aww

Elements | Network | Sources | Timeline | »

```
<!DOCTYPE html>
<html>
▼<head>
    <meta charset="utf-8">
    <title>catscatscats.com</title>
    <link rel="stylesheet" type="text/css" href="style.css">
  </head>
▼<body>
    <h1 id="header">Hello, this is the title.</h1>
  ▼<div id="content">
    ▼<div class="figure">
        <img src="attack.jpg">
        <h2>Roar</h2>
      </div>
    </div>
  ▼<div id="footer">
    ▼<h2>
        "If you are looking for more kittens, check out "
        <a href="https://www.reddit.com/r/
        aww/">reddit.com/r/aww</a>
      </h2>
    </div>
  </body>
</html>
```

html body h1#header

Styles | Event Listeners | DOM Breakpoints | Properties

Filter

```
element.style {
}
#header {                                   style.css:11
  font-size: 2.75em;
  letter-spacing: 3px;
  text-transform: uppercase;
  padding: ▶ 20px;
  width: 50%;
  margin: ▶ 40px auto 0 auto;
  color: white;
  background-color: rgb(84, 206, 236);
```

40
der –
adding 20
560.500 × 55
20
–
–

More Info & Links: https://css-tricks.com/dom/

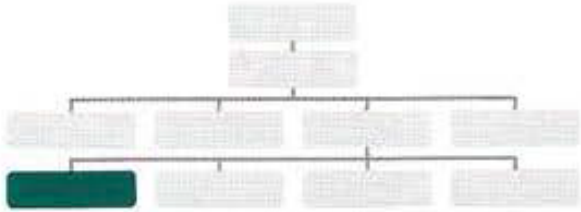
Accessing Elements

METHODS THAT RETURN A SINGLE ELEMENT NODE:

`getElementById('id')`

Selects an individual element given the value of its `id` attribute.
The HTML must have an `id` attribute in order for it to be selectable.

First supported: IE5.5, Opera 7, all versions of Chrome, Firefox, Safari.

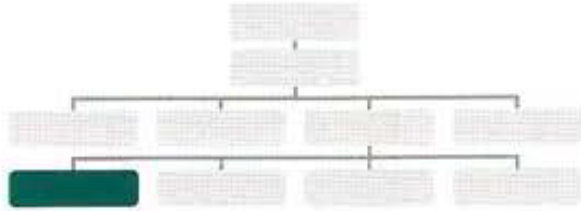


`getElementById('one')`

`querySelector('css selector')`

Uses CSS selector syntax that would select one or more elements.
This method returns only the first of the matching elements.

First supported: IE8, Firefox 3.5, Safari 4, Chrome 4, Opera 10



`querySelector('li.hot')`

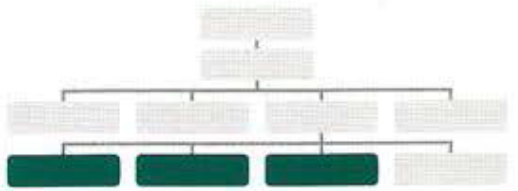
Source: JavaScript & jQuery by Jon Duckett

METHODS THAT RETURN ONE OR MORE ELEMENTS (AS A NODELIST):

`getElementsByClassName('class')`

Selects one or more elements given the value of their `class` attribute.
The HTML must have a `class` attribute for it to be selectable.
This method is faster than `querySelectorAll()`.

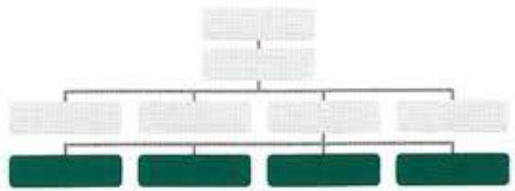
First supported: IE9, Firefox 3, Safari 4, Chrome 4, Opera 10
(Several browsers had partial / buggy support in earlier versions)



`getElementsByTagName('tagName')`

Selects all elements on the page with the specified tag name.
This method is faster than `querySelectorAll()`.

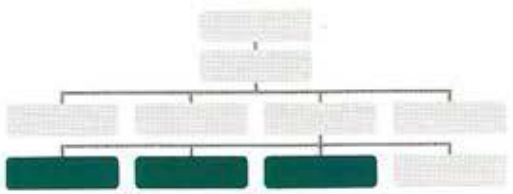
First supported: IE6+, Firefox 3, Safari 4, Chrome, Opera 10
(Several browsers had partial / buggy support in earlier versions)



`querySelectorAll('css selector')`

Uses CSS selector syntax to select one or more elements and returns all of those that match.

First supported: IE8, Firefox 3.5, Safari 4, Chrome 4, Opera 10



Source: JavaScript & jQuery by Jon Duckett

Traversing the DOM

Property	*Description*
element.nextElementSibling	Next element or null
element.previousElementSibling	Previous element or null
element.children	Array* of children
element.firstElementChild	First child element or null
element.lastElementChild	Last child element or null
element.parentElement	Parent element

Creating Elements

DOM Manipulation Method

Step #1: Create new element(s)

```
var paragraphElement = document.createElement("p");
```

Step #2: Add text content, style, etc. as needed

```
paragraphElement.textContent = "Hodor! Hodor hodor, hodor.";
```

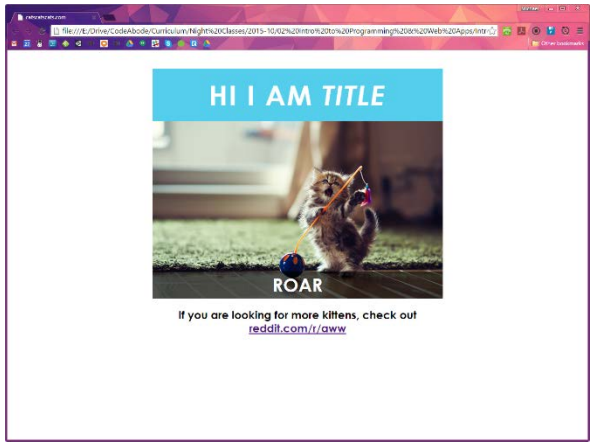
Step #3: Add the newly created element(s) to an element already on the page

```
var contentDiv = document.getElementById("content");
contentDiv.appendChild(paragraphElement);
```

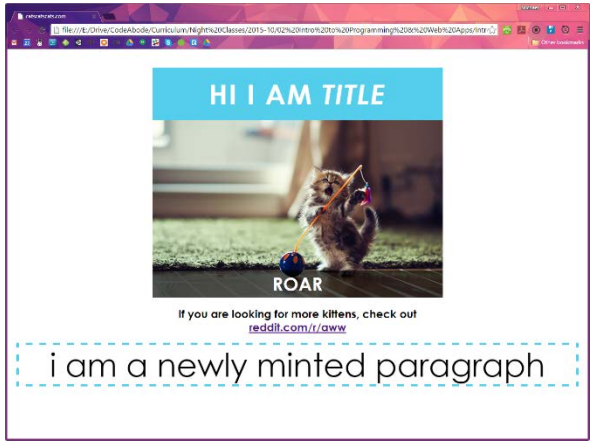
Example: Creating a List

```
<body>

    <div id="wrap">
        <h1>Lots of text</h1>
        <div id="content"></div>
        <h1>List of things</h1>
        <div id="list-container"></div>
        <div id="copywrite">
            &copy; 2015
        </div>
    </div>

</body>
```

```
// Create an unordered list element (e.g. ol)
var orderedList = document.createElement("ol");

// Create three list items (e.g. li) and give each one some text
var listItem1 = document.createElement("li");
listItem1.textContent = "Hodor!";
var listItem2 = document.createElement("li");
listItem2.textContent = "Hodor hodor?";
var listItem3 = document.createElement("li");
listItem3.textContent = "Hodor, HODOR hodor.";

// Append the list items to the ordered list
orderedList.appendChild(listItem1);
orderedList.appendChild(listItem2);
orderedList.appendChild(listItem3);

// Add the list to the list-container div
var listContainerDiv = document.getElementById("list-container");
listContainerDiv.appendChild(orderedList);
```




```
var newParagraph = document.createElement("p");
newParagraph.textContent = "i am a newly minted paragraph";
newParagraph.style.fontSize = "70px";
newParagraph.style.border = "5px dashed rgb(84, 206, 236)";
newParagraph.style.margin = "20px";
```




```
var bodyElement = document.body;
bodyElement.appendChild(newParagraph);
```

Adding/Removing Elements

- *parent*.appendChild(...)
- *parent*.insertBefore(...)
- *parent*.replaceChild(...)
- *parent*.removeChild(...)
  - Or the newer: *element*.remove(...)