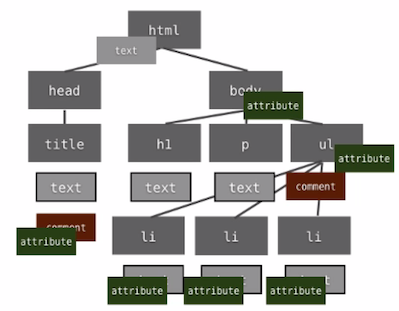
javaScript

Even then smallest <html> document has many nodes!

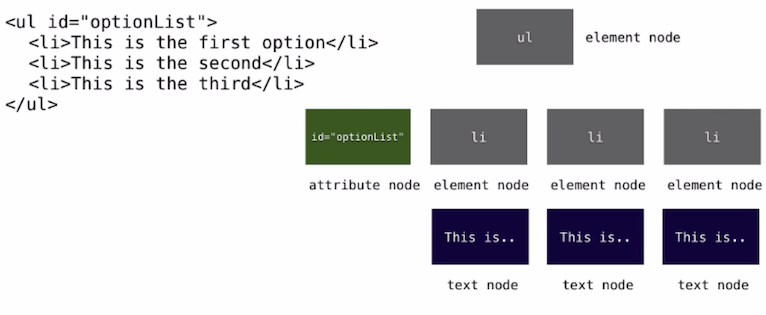


Node Types

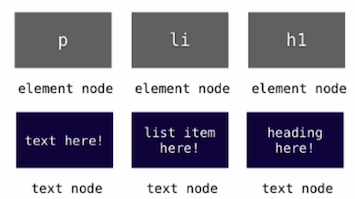
The DOM presents a document as a hierarchy of node objects.

|  |  |
| --- | --- |
| NodeType | Named Constant |
| 1 | ELEMENT\_NODE |
| 2 | ATTRIBUTE\_NODE |
| 3 | TEXT\_NODE |
| 4 | CDATA\_SECTION\_NODE |
| 5 | ENTITY\_REFERENCE\_NODE |
| 6 | ENTITY\_NODE |
| 7 | PROCESSING\_INSTRUCTION\_NODE |
| 8 | COMMENT\_NODE |
| 9 | DOCUMENT\_NODE |
| 10 | DOCUMENT\_TYPE\_NODE |
| 11 | DOCUMENT\_FRAGMENT\_NODE |
| 12 | NOTATION\_NODE |

Element, Attribute and Text Nodes



Element Nodes don’t contain text!



Retrieving Elements by Tag

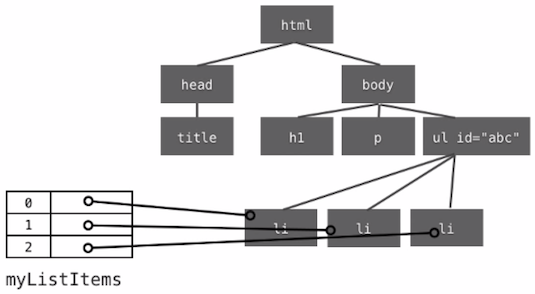
var myListItems = document.getElementsByTagName(“li”);

The getElementsByTagName() method returns a collection of elements in the document with the specified tag name, as a NodeList object. **Does not return an array.**

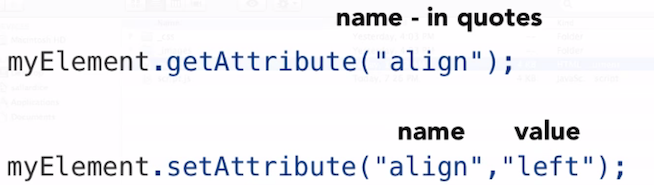
The NodeList object represents a collection of nodes. The nodes can be accessed by index numbers. The index starts at 0.

**Tip:** The parametervalue “\*” returns all the elements in the document.

**Tip:** You can use the **length** property of the NodeList object to determine the number of elements with the specified tag name, and then you can loop through all elements and extract the info you want.



Working with Attributes



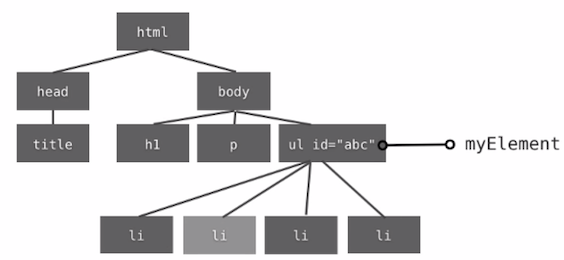
Creating DOM content

First **create** element, then **add** it to the DOM. Example:

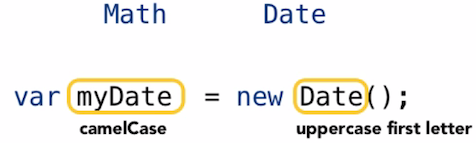
var myNewElement = document.createElement(“li”);

var secondItem = myElement.getElementsByTagName(“li”)[1];

myElement.insertBefore(myNewElement, secondItem);



camelCase var names, Uppercase Object



Define Functions before you call them!

function otherFunction (x) {

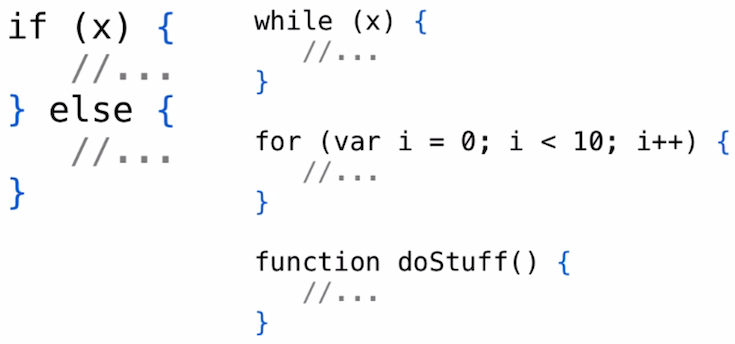
// Some stuff  
}

function someFunction(x){

otherFunction();

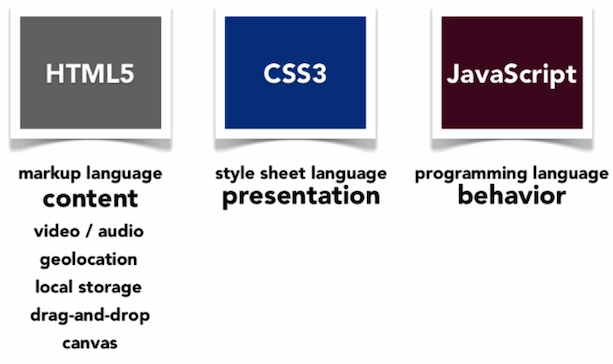
}

Brace Style

****

Guidelines Overview

* Use camelCase for variables, functions and methods
* Open curly braces on the same line
* Always use blocks – even if only one line
* Define your functions before your call them
* Always use semicolons to end a statement
* Always use var when declaring a variable

HTML5 and JavaScript

Web workers

A web worker is JavaScript running in the background, without affecting the performance of the page.

When executing scripts on an HTML page, the page becomes unresponsive until the script is finished.

A web worker is JavaScript that runs in the background, independently of other scripts, without affecting the performance of the page. You can continue to do whatever you want: clicking, selecting things, etc., while the web worker runs in the background.

var worker = new Worker(“anotherjavascriptfile.js”);

// get ready to receive messages from the worker

worker.onmessage = function(e) {

console.log(“The worker called me!”);

};

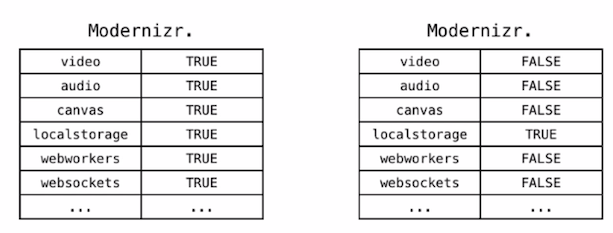
// send messages to the worker

worker.postMessage(“firstFunction”);

Modernizer

Modernizr is a small piece of JavaScript code that automatically detects the availability of next-generation web technologies in your user's browsers. Rather than blacklisting entire ranges of browsers based on “UA sniffing,” Modernizr uses [feature detection](https://modernizr.com/docs/#what-is-feature-detection) to allow you to easily tailor your user's experiences based on the actual capabilities of their browser.

With this knowledge that Modernizr gives you, you can take advantage of these new features in the browsers that can render or utilize them, and still have easy and reliable means of controlling the situation for the browsers that cannot.



Creating Objects

// create an object

var playerFred = { name: “Fred”, score: 1000, rank: 1 };

// add a new property

playerFred.gameType = “MMORPG”;

// add a method

playerFred.logScore = function() {

console.log(this.score);

};

// call the method

playerFred.logScore();

// create another object

var playerBob = { name: “Bob”, highscore: 50, level: “b”};