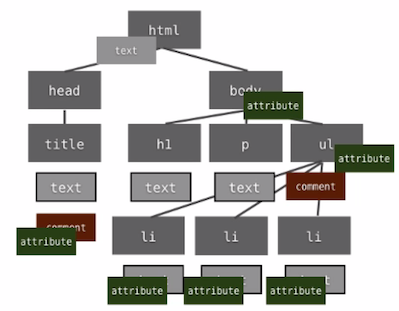
javaScript

Even then smallest <html> document has many nodes!

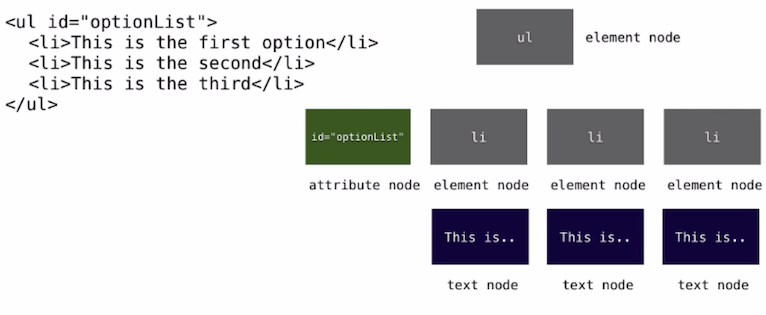


Node Types

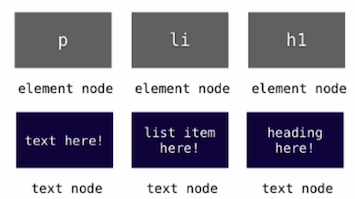
The DOM presents a document as a hierarchy of node objects.

|  |  |
| --- | --- |
| 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

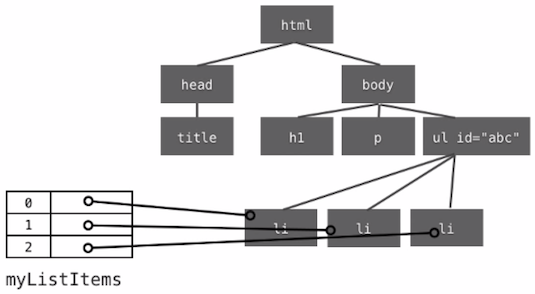
var myListItems = document.getElementsByTagName(“li”);

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.**

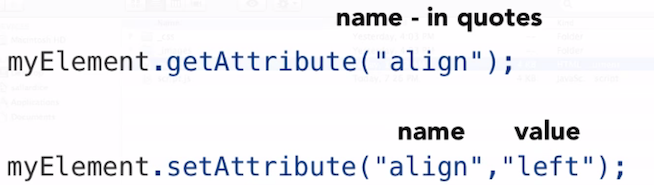
The NodeList object represents a collection of nodes. The nodes can be accessed by index numbers. The index starts at 0.

**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



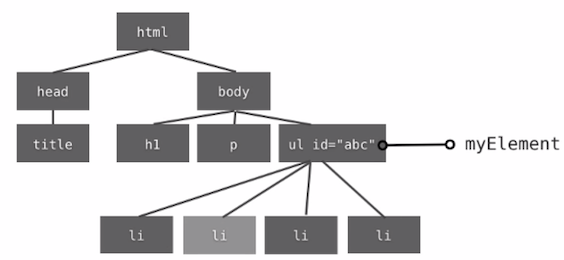
Creating DOM content

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

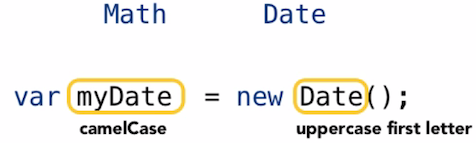
var myNewElement = document.createElement(“li”);

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

myElement.insertBefore(myNewElement, secondItem);



camelCase var names, Uppercase Object



Define Functions before you call them!

function otherFunction (x) {

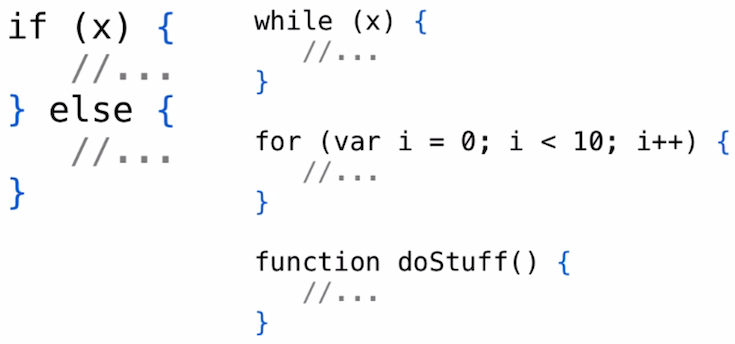
// Some stuff  
}

function someFunction(x){

otherFunction();

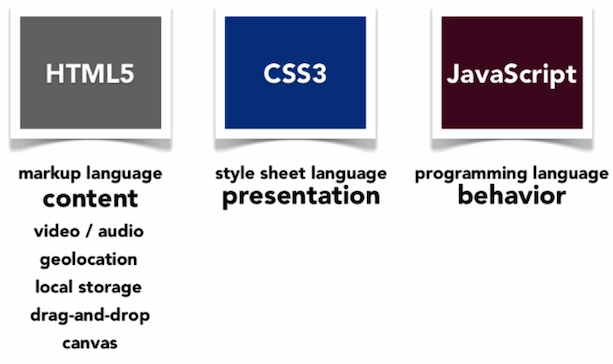
}

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!”);

};

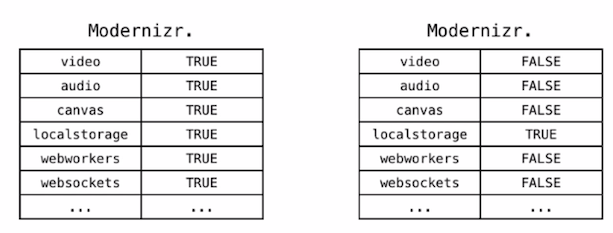
// send messages to the worker

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”};

Creating Objects with the Constructor

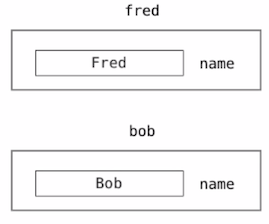
function Player(n){

this.name = n;

}

var fred = new Player(“Fred”);

var bob = new Player(“Bob”);



Creating variables

All JavaScript **variables** must be **identified** with **unique names.**

These unique names are called **identifiers.**

Identifiers can be short names (like x or y), or more descriptive names (age, sum, totalVolume).

The general rules for constructing names for variables (unique identifiers) are:

* Names can contain ***letters, digits, underscores,*** and ***dollar signs.***
* Names must begin with a letter.
* Names can also begin with **$** and **­\_**
* Names are case sensitive (y and Y are different variables)
* Reserved words ***cannot*** be used as names.

Arrays and methods

Arrays are a special type of object. The **typeof** operator in JavaScript returns **“object”** for arrays. Arrays use **numbers** to access its “elements.” The real strength of arrays are the built-in **properties** and **methods.**

**Array Properties**

|  |  |
| --- | --- |
| Property | Description |
| constructor | Returns the function that created the Array object’s prototype |
| length | Sets or returns the number of elements in an array |
| prototype | Allows you to add properties and methods to an Array object. |

**Array Methods**

|  |  |
| --- | --- |
| Method | Description |
| concat() | Joins two or more arrays, and returns a copy of the joined arrays. |
| indexOf() | Search the array for an element and returns its position |
| join() | Joins all elements of an array into a string |
| lastIndexOf() | Search the array for an element, starting at the end, and returns its position. |
| pop() | Removes the last element of an array, and returns that element. |
| push() | Adds new elements to the end of an array, and returns that element. |
| reverse() | Reverses the order of the elements in the array |
| shift() | Removes the first element of an array, and returns that element |
| slice() | Selects a part of an array, and returns the new array |
| sort() | Sorts the elements of an array |
| splice() | Adds/Removes elements from an array |
| toString() | Converts an array to a string, and returns the result |
| unshift() | Adds new elements to the beginning of an array, and returns the new length |
| valueOf() | Returns the primitive value of an array |

Numbers

**JavaScript numbers are 64-bit floating point numbers.** This format stores numbers in 64 bits, where the number (the fraction) is stored in bits 0 to 51, the exponent in bits 52 to 62, and the sign in bit 63.

**Addition vs. Concatenation**

var foo = 5;

var bar = 6;

console.log(foo + bar); // 11

var foo = “5”;

var bar = “5”;

console.log(foo + bar); // 55

var foo = 5;

var bar = “6”;

console.log(foo + bar); // 56 – one is a string

var foo = 5;

var bar = “b”;

console.log(foo \* bar); // NaN

Strings

A javaScript string stories a series of characters like “John Doe.” A string can be any text inside double or single quotes.

String indexes are zero-based: ***The first character is in position 0, the second in 1, and so on.***

Primitive values, like “John Doe”, cannot have properties or methods ***(because they are not objects).***

But with JavaScript, methods and properties are also available to primitive values, because JavaScript treats primitive values as objects when executing methods and properties.

**String Properties**

|  |  |
| --- | --- |
| Property | Description |
| constructor | Returns the string’s constructor function |
| length | Returns the length of a string |
| prototype | Allows you to add properties and methods to an object |

**String Methods**

|  |  |
| --- | --- |
| Method | Description |
| charAt() | Returns the character at the specified index (position) |
| charCodeAt() | Returns the Unicode of the character at the specified index |
| concat() | Joins two or more strings, and returns a new joined string |
| fromCharCode() | Converts Unicode values to characters |
| indexOf() | Returns the position of the first found occurrence of a specified value in a string |
| lastIndexOf() | Returns the position of the last found occurrence of a specified value in a string |
| localeCompare() | Compares two strings in the current locale |
| match() | Searches a string for a match against a regular expression, and returns the matches |
| replace() | Searches a string for a specified value, or a regular expression, and returns a new string where the specified values are replaced |
| search() | Searches a string for a specified value, or regular expression, and returns the position of the match |
| slice() | Extracts a part of a string and returns a new string |
| split() | Splits a string into an array of substrings |
| substr() | Extracts the characters from a string, beginning at a specified start position, and through the specified number of character |
| substring() | Extracts the characters from a string, between two specified indices |
| toLocaleLowerCase() | Converts a string to lowercase letters, according to the host’s locale |
| toLocaleUpperCase() | Converts a string to uppercase letters, according to the host’s locale |
| toLowerCase() | Converts a string to lowercase letters |
| toString() | Returns the value of a String object |
| toUpperCase() | Converts a string to uppercase letters |
| trim() | Removes whitespace from both ends of a string |
| valueOf() | Returns the primitive value of a String object |

**String HTML Wrapper Methods**

The HTML wrapper methods return the string wrapped inside the appropriate HTML tag. **These are not standard methods, and may not work as expected in all browsers.**

|  |  |
| --- | --- |
| Method | Description |
| anchor() | Creates an anchor |
| big() | Displays the string using a big font |
| blink() | Displays a blinking string |
| bold() | Displays a string in bold |
| fixed() | Displays a string using a fixed-pitch font |
| fontcolor() | Displays a string using a specified font color |
| fontsize() | Displays a string using a specified size |
| italics() | Displays a string in italic |
| link() | Displays a string as a hyperlink |
| small() | Displays a string using a small font |
| strike() | Display |