JavaScript

for Developers

MAX003AC

**Audience:**

Developers with a foundation in C# or Java who need a solid introduction to JavaScript.

**Prerequisites:**

Basic programming skills using C# or Java.

AUTHOR

Michael T Smith  
Microsoft Certified Trainer  
SharePoint / Office Servers and Services MVP  
Senior Instructor, MAX Technical Training

COPYRIGHT

© 2017 MAX Technical Training, Inc. All Rights Reserved. This manual and any training materials supplied with it are copyrighted with all rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system or translated into any language in any form by any means without the w

Portions copyright Microsmith, Inc. Used with permission.

TRADEMARKS

Brand names, company names and product names used herein are trademarks or registered trademarks of their respective companies.

MAX Technical Training, Inc.   
4900 Parkway Drive, Suite 160   
Mason, OH 45040  
513.322.8888

MA-003AC 3/27/17

Table of Contents

[Module 1: Basic Scripting 3](#_Toc481354149)

[JavaScript Basics 3](#_Toc481354150)

[JavaScript or ECMAScript? 3](#_Toc481354151)

[Which Version of JavaScript am I Using? 3](#_Toc481354152)

[Shims, Shivs, and Polyfills 4](#_Toc481354153)

[JavaScript 4](#_Toc481354154)

[Key parts of a script 4](#_Toc481354155)

[Useless Trivia! 15](#_Toc481354156)

[Module 2: The Document Object Model 17](#_Toc481354157)

[The DOM 17](#_Toc481354158)

[Accessing Objects from the DOM 17](#_Toc481354159)

[Document Properties 18](#_Toc481354160)

[Document Methods to Find Elements 18](#_Toc481354161)

[Adding to the DOM 19](#_Toc481354162)

[Write HTML from Strings 19](#_Toc481354163)

[Creating New Elements 19](#_Toc481354164)

[Module 3: Diving in Deeper! 20](#_Toc481354165)

[Data Types 20](#_Toc481354166)

[JavaScript Primitives 20](#_Toc481354167)

[JavaScript Default Objects 20](#_Toc481354168)

[Window Methods 21](#_Toc481354169)

[Navigator Properties 22](#_Toc481354170)

[JavaScript Objects 22](#_Toc481354171)

[Accessing Object Properties 23](#_Toc481354172)

[JavaScript Numbers 24](#_Toc481354173)

[JavaScript Strings 26](#_Toc481354174)

[JavaScript Dates 29](#_Toc481354175)

[JavaScript Booleans 29](#_Toc481354176)

[“this” 29](#_Toc481354177)

[Is JavaScript Compiled? 29](#_Toc481354178)

[Comment 31](#_Toc481354179)

[Variables 32](#_Toc481354180)

[Scopes 32](#_Toc481354181)

[Hoisting 33](#_Toc481354182)

[Strict Mode 33](#_Toc481354183)

[Creating Variables 34](#_Toc481354184)

[Deleting Variables 34](#_Toc481354185)

[Functions 35](#_Toc481354186)

[Function FAQ 35](#_Toc481354187)

[Anonymous Functions 35](#_Toc481354188)

[Closures 35](#_Toc481354189)

[Self-Invoking / Immediately Invoked Functions 36](#_Toc481354190)

[The Module Pattern 36](#_Toc481354191)

[Errors 37](#_Toc481354192)

[Events 37](#_Toc481354193)

[Recursion 37](#_Toc481354194)

[A Factorial Example 37](#_Toc481354195)

[Module 4: jQuery 40](#_Toc481354196)

[jQuery 40](#_Toc481354197)

[First the negatives: 40](#_Toc481354198)

[And then the positives: 40](#_Toc481354199)

[Downloading the library 41](#_Toc481354200)

[jQuery Versions 41](#_Toc481354201)

[Linking to the library 41](#_Toc481354202)

[Selecting Elements using jQuery 42](#_Toc481354203)

[Using the data returned by jQuery 43](#_Toc481354204)

[Setting CSS properties 43](#_Toc481354205)

[Running a function against the return set 44](#_Toc481354206)

[Useless Trivia! 44](#_Toc481354207)

[Module 5: Ajax and Web Services 45](#_Toc481354208)

[Appendix 1: Editors and Tools 46](#_Toc481354209)

[Visual Studio Code 46](#_Toc481354210)

# Module 1: Basic Scripting

JavaScript has been described as a simple scripting language and also as a complex object oriented language. In this module we will explore basic scripting. Many of the topics in the module will be expanded upon in later sections of this course.

## JavaScript Basics

In this module we will look at the basics to:

* write and call a JavaScript function
* find HTML elements in a web page
* change properties of HTML elements
* write loops
* test strings and values (IF statements)
* and a few more…

## JavaScript or ECMAScript?

JavaScript was a creation of Netscape in 1995. Other browser manufactures created their own versions of JavaScript with new, different or incompatible features. ECMAScript is a standard created by Ecma International to standardize the language. ECMAScript has evolved with multiple editions (1, 2, 3, 5, 5.1, 6 and 7). Starting with the 6th edition ECMAScript was renamed so ECMAScript is now called ECMAScript 2015 and the 7th edition is ECMAScript 2016.

* Most modern browsers implement most of ECMAScript 2015 and some of ECMAScript 2016.
* Many browsers have non-ECMAScript features.
* Throughout the web you will find both “ECMAScript” and “JavaScript” used to refer to the language. In this course we will use the more generic “JavaScript” name.

## Which Version of JavaScript am I Using?

JavaScript does not have a built in version property that you can check. Instead, you will need to test to see if a feature exists. Even if there was a version number available, you could not depend on it as each browser vendor has chosen to implement different subsets of the language.

Here are a few resources to help sort out which versions are “out there” and which features are supported by which browser.

* https://en.wikipedia.org/wiki/ECMAScript
* https://en.wikipedia.org/wiki/JavaScript#Version\_history
* http://kangax.github.io/compat-table/es5/
* http://kangax.github.io/compat-table/es6/
* http://caniuse.com

To check for support for a feature:

* Lookup the feature here: http://caniuse.com
* Detect at runtime using https://modernizr.com

### Shims, Shivs, and Polyfills

As all browsers have not implemented HTML and CSS to the same level, or even the same way, we will need tools to “level the playing field”. Shims are wedges that you use to level, adjust or tweak things. In the world of HTML, shims are typically CSS and JavaScript that you add to your page to make browsers more compatible, let older browsers support new features, and just make the developers life a bit easier. A “shiv” is the same as a “shim”, and the term is most often associated with “HTML5shiv.js”.

A “polyfill” is a shim that implements a modern API in an older browser without impacting newer browsers that already support the API. If the polyfill is removed from a newer browser that does not need it, it will make no change to functionality. A shim makes changes so all browsers behave the same. As an example, “normalize.css” changes the default margins and removing normalize.css may change the layout of the page.

The three terms are often used interchangeably.

For a list of shims, shivs and polyfills see:   
 https://github.com/Modernizr/Modernizr/wiki/HTML5-Cross-browser-Polyfills

## JavaScript

JavaScript is an object oriented language. As most languages, JavaScript has no user interface and must be used within a host application. In this class we will only consider one host application, the web browser. The top level object in the browser is named “window”. All other objects, including core JavaScript objects, are properties of this object. In tabbed browsers, each tab has its own window object.

JavaScript:

* Is not Java! While both share many constructs with the rest of the C family of languages, it is not derived from Java.
* Is object oriented. For that matter, everything is an object! But JavaScript does not use classes, class instances or class-style inheritance.
* Is not a standalone application. You cannot create a JavaScript console application to run from the desktop. JavaScript is embedded into a host applications, and for our discussions, JavaScript is embedded in a web browser.
* Has no user interface. Any interaction with a user will be through the host. In the case of a browser this the browser’s debugging console, a web page or things like the alert() popup.

## Key parts of a script

#### Basic JavaScript Syntax

JavaScript is written inside of <script> tags embedded in tag attributes, called from HTML element events (onclick) or loaded from text files.

This example displays a popup message from a script block:

<script type="text/javascript">

alert('hello!');

</script>

Note: The type attribute is optional in HTML 5 and required in earlier versions.

JavaScript can also be directly called from element events: ("onclick" is an event)

<button onclick="javascript:alert('hello');">Click me</button>

#### JavaScript is written as a series of statements separated by semicolons:

var x = 3;

var y = 5;

var z = x \* y;

alert(z);

The semicolons are often omitted when each statement is on its own line.

#### JavaScript is "case sensitive"

These examples would fail:

Alert('hello');

ALERT('hello');

document.GETELEMENTBYID('test');

document.GetElementById('test');

These examples work:

alert('hello');

document.getElementById('test');

Notes:

* JavaScript is very case sensitive. The following are all different variables: “bankaccount”, “BankAccount”, “BANKACCOUNT”.
* With the exception of Math and JSON, most of the keywords and methods in JavaScript start with a lowercase letter.
* Function libraries like Math and JSON use capital letters.

#### Comments

The two toughest coding things you will have to do is figure out someone else's code, or six months after you have written it, figure out your own code. Document your work! Add comments to explain what you did, or where you borrowed the code from.

Add a comment as a single line:

// this code from Mike's book!

var allTables = document.getElementsByTagName('table');

Add a comment to the end of a line:

var x; // this holds the counter

var y; // this is the row number - I should have used better names!

Add a block of comments between "/\*" and "\*/":

/\*

the following routine is used to color code

task lists by looking for key words in cells

and then changing the color of the row

\*/

#### Code Blocks

JavaScript works with blocks of code that are enclosed within curly brackets. Many of the JavaScript blocks that you will find in this course are for IF, FOR and FUNCTION statements. Code blocks are frequent nested inside of other code blocks. Here's an example of using a FOR loop to repeat a block of code for each TD element found in the page. It includes several nested code blocks.

var allTDs = document.getElementsByTagName("TD"); // find all of the TDs

for (var i=0; i< allTDs.length; i++ ) // loop through all of the TDs

{ **// for block starts here**

if (allTDs[i].className=="ms-vb2") //find the TDs styled for lists

{ **// if block starts here**

if (allTDs[i].innerHTML=="In Progress")

{ **// nested if block starts here**

alert("found a cell containing 'In Progress' ");

} **// nested if block ends here**

} **// if block ends here**

} **// for block ends here**

While the brackets are required when the block has more than one line of code, they are optional if the block has only one line. *Your code will be easier to understand if you always add the brackets*.

This works without the brackets:

for ( var i=0; i<10; i++ )

alert( "This is loop # " + i);

alert( "all done" );

But this is easier to understand as the brackets show which lines of code that are used with the FOR:

for ( var i=0; i<10; i++ )

{

alert( "This is loop # " + i);

}

alert( "all done" );

#### Spaces, line breaks and semicolons

While JavaScript generally does not need line breaks, indents or extra spaces, these will make the code much easier for humans to read. Semicolons are the "official" end of line punctuation, but code will still work if the line of code ends with a line break. Semicolons are not needed after closing curly brackets ( } ).

The following works as one long line of code, but is hard to read and debug:

var allTDs = document.getElementsByTagName("TD"); for (var i=0; i< allTDs.length; i++ ) ; { if (allTDs[i].className=="ms-vb2") { if (allTDs[i].innerHTML=="In Progress") { alert("found a cell containing 'In Progress' "); } } }

This code is much easier to read, and you can easily see if you are missing a bracket somewhere:

var allTDs = document.getElementsByTagName("TD"); // find all of the TDs

for (var i=0; i< allTDs.length; i++ ) // loop through all of the TDs

{ // FOR block starts here

if (allTDs[i].className=="ms-vb2") //find the TDs styled for lists

{

if (allTDs[i].innerHTML=="In Progress")

{

alert("found a cell containing 'In Progress' ");

}

}

} // FOR block ends here

Some JavaScript libraries do look like the first example. By removing the excess characters and comments you can shrink the file size and reduce download time, but at the expense of readability. If you do this, keep in mind that end of line semicolons are required and that while block comments ( /\* … \*/ ) will work, "//" comments will not.

#### JavaScript Variables

*Note: This is a very brief introduction to variables. They are covered in depth in the next module.*

Variables are not assigned data types when they are created as in C# or VB.NET.

var a; // a is untyped. “typeof a” returns “undefined”

var b;

a = 5; // a is now typed. “typeof a” returns “number”

b = "abc"; // b is now typed. “typeof b” returns “string”

Multiple variables can be declared in a single statement;

Variables can store simple data types such as strings and numbers. A variable can also store an object such as an HTML element (tag). This example finds a table with an ID of "table1" and stores it in a variable named "atable":

var atable = document.getElementById('table1');

atable.width = "100px";

Variables can store arrays, including arrays of HTML elements:

var allTableCells = document.getElemenstByTagName('td');

alert('there are ' + allTableCells.length + ' TDs in this page');

Individual array elements are addressed by using an indexer ( [index] ) that starts with zero:

var allTableCells = document.getElemenstByTagName('td');

alert( 'The first cell contains: ' + allTableCells[0].innerHTML);

alert( 'The second cell contains: ' + allTableCells[1].innerHTML);

alert( 'The last cell contains: ' + allTableCells[ allTableCells.length - 1 ].innerHTML);

Arrays can be created in code by adding new items to the end of the array:

var carArray = new Array();

carArray[0] = "Chevy";

carArray[1] = "Ford";

carArray[2] = "Jeep";

Arrays can also be created by defining a list of items between square brackets:

var carArray = ["Chevy","Ford","Jeep"];

Arrays are often accessed using a FOR loop:

for ( var x=0; x<carArray.length; x++)

{

document.write(carArray[x] + "<br />");

}

#### JavaScript Functions

*Note: This is a very brief introduction to functions. They are covered in depth in the next module.*

Functions are "reusable blocks of code" that you can write once and then call one or many times from other code.

Notes:

* Functions are stored as variables. If a file has a variable named “foo” and a function named “foo” then the variable will always be overwritten by the function, regardless of the order in the code.
  + This:
    - function foo(){ }  
      var foo;  
      typeof foo; // "function"
  + is the same as this:
    - var foo;  
      function foo(){ }  
      typeof foo; // "function"
  + Functions are created at the Global scope by default. (I.e. they are a property of the window object. window.foo()
  + Functions can be declared inside of another function. This hides it from the Global scope.

A function declaration can be as simple as:

function myfunctionname()

{

// my code here

}

Functions can also have parameters so you can pass different data to the function each time it is called.

function annoyUser(msg)

{

alert(msg);

}

annoyUser("hello!");

annoyUser("JavaScript is fun");

annoyUser("do you like popups?");

#### Conditional Statements

JavaScript includes the common set of tools found in most languages: IF, SWITCH, FOR, DO and WHILE.

##### Testing values - IF

JavaScript's IF statement is used to conditionally run a block of code. The most common mistake made by new JavaScript developers is typing just one equal sign when doing comparisons.

Example - if A equals B then display an alert:

var a = 5;

var b = 6;

if ( a == b ) { alert("it's a match!") }

Note in this example the single equal sign for assignments ( a = 5 ) and the double equal signs for equality ( a == b).

Example - if A is greater than B then display an alert else display a different alert:

var a = 5;

var b = 6;

if ( a > b )

{

alert("A is greater than B");

}

else

{

alert("B is greater than or equal to A");

}

Example - if A and B are not equal then display an alert:

var a = 5;

var b = 6;

if ( a != b ) { alert("it's not a match!") }

##### JavaScript Comparison Operators

== is equal to

=== is exactly equal to (value and type)

!= is not equal to

> is greater than

< is less than

>= is greater than or equal to

<= is less than or equal to

The “===” comparison operator is similar to the following:

if ( a == b && typeof a == typeof b) { } // && = Boolean AND

##### The Conditional “ternary” Operator.

Note: “ternary” = “composed of three items”.

JavaScript includes an operator “?” that behaves as an IF structure.

var x = *condition* **?** *valueIfTrue* **:** *valueIfFalse*

Example:

var state = "OH";

var tax = (state = "OH") ? .065 : 0.00;

// tax = .065

Equivalent to:

var state = "OH";

var tax;

if (state = "OH")

{

tax = .065;

}

Else

{

tax = 0;

}

##### Switch

The switch statements accepts an expression or value and uses that value to select a case clause. The switch statement is often described as the equivalent of nested IF statements.

switch (*expression*) {

case *value1* :

// code for value 1

break; // if break is missing, the code for value2 will also be run!

case *value2* :

// code for value 2

break;

case *value3* : // value3 fall through and runs value4’s code

case *value4* :

// code for value 4

break;

default : // default is optional

// code for default

}

Notes:

* Except where there is a missing break statement, case and default sections can be listed in any order.
* Code blocks can have many lines of code, including other nested statements.

Example:

// where is it manufactured?

var productCategory = "toys";

var location = “”;

switch (productCategory) {

case 'furniture' :

location = ‘Lexington';

break;

case 'toys' :

location = ‘Cincinnati';

break;

case 'appliances' :

case 'clocks' :

case ' aquariums ' :

location = ‘Columbus';

break;

default : // default is optional

location = ‘Dayton';

}

console.log(productCategory + " are made in " + location);

From the above:

|  |  |
| --- | --- |
| appliances | Columbus |
| aquariums | Columbus |
| clocks | Columbus |
| furniture | Lexington |
| sporting goods | Dayton |
| tools | Dayton |
| toys | Cincinnati |

#### Looping Statements

JavaScript supplies several statements for creating code that loops. Loops are blocks of code that run while or until a certain condition is true.

* “for” loops use a counter and run for a known number of iterations.
* “while” loops run zero to many times and exit when an expression changes to false. The expression is evaluated at the start of each loop.
* “do” loops run one to many times and exit when an expression changes to false. The expression is evaluated at the end of each loop.

Notes:

* Something inside of the loop must change the expression, otherwise you will have created an endless loop.
  + while (true) { /\* do this forever \*/ } // endless loop, unless there is a break statement.
* Loops can exit at any time by using break.
  + while (true) { if (someexpression) { break }; /\* do stuff \*/ }
* Loops inside of a function can exit at any time by using return to exit the function.
* Loops can skip to the next iteration by using continue.

##### WHILE

The while loop tests an expression before executing a block of code and repeats the block while the expression equals true.

var names = ["Ohio", "Indiana", "California", "New York", "Florida", "Nevada", "Alaska"];

var i = 0;

**while (names[i] != "Florida")**

{

console.log("Hello " + names[i]);

i++;

}

The following example loop would run forever if not for the break statement. This example produces the same results as above.

var names = ["Ohio", "Indiana", "California", "New York", "Florida", "Nevada", "Alaska"];

var i = 0;

while (**true**)

{

if (names[i] == "Florida")

{

**break; // exit the loop**

}

console.log("Hello " + names[i]);

i++;

}

##### DO WHILE

A do loop is similar to a while loop, but tests the expression at the end of each loop. A do loop will always run at least once.

// this loop counts down to 6, but run at least once, even if the number is less than 6

var num = 10

do {

console.log("The number is " + num);

num--;

} while (num>5);

// 10, 9, 8, 7, 6

Even with num=3 this will still display at least one message.

// this loop counts down to 6, but run at least once, even if the number is less than 6

var num = 3

do {

console.log("The number is " + num);

num--;

} while (num>5);

// 3

##### FOR

A for loop “counts”. It needs three parameters to define a counter variable, a test for completion, and code to increment or decrement the counter.

for (*counter*, *condition*, *incrementDecrement*) { /\* code block \*/ }

The counter is typically initialized in the for statement, but could also be a function or global scoped variable.

Count from 5 to 9:

for (i=5; i<10; i++) { console.log(i) } // 5,6,7,8,9

Count from 5 to 9:

var i = 5;

for (i; i<10; i++) { console.log(i) } // 5,6,7,8,9

In the next example, the “var” may lead you to believe that the variable is scoped to the FOR block. It is not! It is “hoisted” to the function that contains the FOR loop, or to the Global scope.

for (var i=5; i<10; i++) { console.log(i) } // 5,6,7,8,9

console.log(i); // 10

##### Looping through an array

Many of the DOM selectors such as getElementsByTagName return an array of HTML elements. Once you have the HTML elements you want to work with stored in an array, you can then loop through them with a "FOR" loop. JavaScript's FOR statement has three parameters: the initial value of the indexer (typically 0), the test for loop completion (typically when the indexer exceeds the number of items in the array), and how the indexer is incremented (typically by 1). Here's an example that displays an alert 10 times (i = 0 through 9):

var i;

for (i=0; i<10; i++ )

{

alert( "This is loop # " + i);

}

If your FOR loop always loops through all the items in an array then you can use this version:

var someitems = ["apples","oranges","pears"];

var item;

for ( item=0; **item<someitems.length**; item++ )

{

alert("This is item " + item + " and is " + someitems[item] );

}

Or you could use this shorter FOR - IN version:

var someitems = ["apples","oranges","pears"];

var item;

for ( item in someitems )

{

alert("This is item " + item + " and is " + someitems[item] );

}

Notes:

* In both of the above examples, “item” is integer that represents the “nth” item.
* Arrays in JavaScript are numbered starting with zero!
* FOR…IN is similar to FOR…EACH found in other languages. JavaScript even has a FOR EACH, but it is deprecated in favor of the FOR…IN. A simple FOR is preferred over FOR…IN when working with arrays. (Details in a later module!)

We have mentioned blocks before, but now is a good time for a review. Note that FOR is a block structure and that the start and the end of the block is marked with curly brackets. The brackets are optional if there is only one line of code in the block. The following is identical to the previous example:

for ( var i=0; i<10; i++ )

alert( "This is loop # " + i);

JavaScript does not care about line breaks, so both of the following are also identical to the previous example:

for ( var i=0; i<10; i++ ) { alert( "This is loop # " + i); }

for ( var i=0; i<10; i++ ) alert( "This is loop # " + i);

Blank lines, spaces and tabs are free! So use them to improve readability. Here's a more complete and nicely formatted example:

var x = document.getElementsByTagName("TD"); // find all of the TDs

for (var i=0; i<x.length; i++ ) // loop through all of the TDs

{

if (x[i].className=="ms-vb2") //find the TDs styled for lists

{

if (x[i].innerHTML=="In Progress")

{

alert("found a cell containing 'In Progress' ");

}

}

}

##### FOR OF

ECMAScript 2015 (6) adds a FOR OF loop that is similar to the FOR EACH found in other languages. At this time, FOR OF does not work in Internet Explorer 11, but does work in other modern browsers.

See: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for...of

var someitems = ["apples","oranges","pears"];

for ( let item of someitems )

{

console.log( "This is item " + item ); // item is the value, not the index!

}

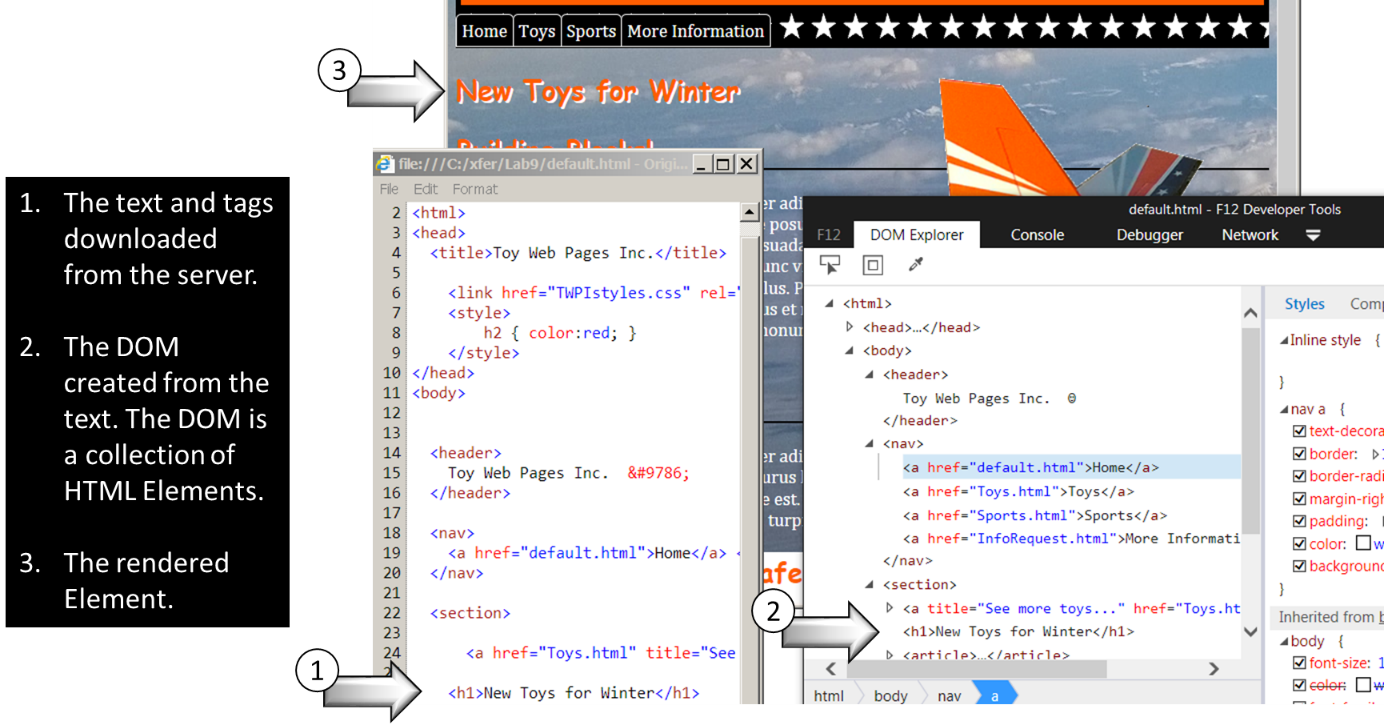
## Useless Trivia!

* JavaScript was first called "Mocha", and when first released with Netscape 2.0 in September 1995 JavaScript was called "LiveScript".
* LiveScript was renamed to JavaScript in December of 1995 as part of a partnership with Sun to integrate Java applets into LiveScript and Netscape.
* Microsoft added JavaScript to Internet Explorer 3.0, but called it JScript
* In November of 1996 JavaScript became formalized into an international standard named ECMAScript

# Module 2: The Document Object Model

## The DOM

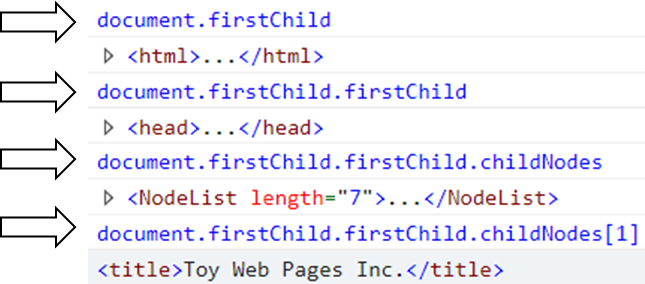
Neither browsers nor JavaScript work directly with the HTML file that was received from the web server. The browser parses this file and converts it into a tree structure called the Document Object Model, or DOM for short.



## Accessing Objects from the DOM

After the HTML has been parsed into the DOM, the individual elements are stored as objects in a tree structure. This structure is the window.document object, which is almost always shortened to just document.

The document object can be accessed by starting at the top, and then drilling down through the collection of child nodes. While this can be done, there are better ways!



### Document Properties

When the DOM is created, browsers will pull out some of the content and make it available as properties of the document object.

* document.anchors – the collection of <a> tags that have “name” attribute. (Not too useful anymore as HTML5 does not support the name attribute on <a> tags!)
* document.links – the collection of <a> tags. I.e. all links on a page.
* document.head – the <head> element - the <head> section of a document.
* document.body – the <body> element - the <body> section of a document.
* document.images – the collection of images.
* docment.url – the document location.
* For a complete list see: https://developer.mozilla.org/en-US/docs/Web/API/Document  
  or https://www.w3schools.com/jsref/dom\_obj\_document.asp

### Document Methods to Find Elements

Most of your JavaScript work will probably be around searching and updating the DOM.

#### Searching for Elements

* getElementById() – Returns the single element with the specified ID. If no element found, this returns null.
* getElementsByClassName() – Returns a collection of elements that have the specified class name. If no element found, this returns an empty collection. (I.e. not a null.)
* getElementsByName() – Returns a collection of form elements that have the specified name attribute value. If no element found, this returns an empty collection. (I.e. not a null.)
* getElementsByTagName() – Returns a collection of elements that have the specified tag name. If no element found, this returns an empty collection. (I.e. not a null.)

Examples:

* var tables = document.**getElementsByTagName**("table");  
  console.log(tables.length) // 0 .. many
* var images = document.**getElementsByTagName**("img");  
  console.log(tables.length) // 0 .. many
* <div class="lefttoright **nav** bluetext">…<div>  
  var navdiv = document.**getElementsByClassName**("**nav**");   
  console.log(tables.length) // 0 .. many
* <div id="**corpLogo**"> … </div>  
  var navdiv = document.**getElementsById**("**corpLogo** ");   
  // navdiv = null or a DIV object

#### Searching for Elements using CSS Selectors

CSS uses selectors to find elements by ID, tag name or class. In modern browsers you can use .querySelector and .querySelectorAll to find elements using CSS-like selectors. These selector range simple select by ID ("#someId") to complex queries like "div.upcoming-events-section div.row:first-child".

* querySelector() – Returns the first element that matches a CSS style selector.
* querySelectorAll() – Returns a collection of elements that match a CSS style selector.

Select for tags:

* var tables = document.**querySelectorAll**("table");  
  console.log(tables.length) // 0 .. many
* var tables = document.**querySelector**("table");  
  console.log(tables.length) // null or a TABLE object

Search for complex patterns:

* var tables = document.**querySelector**("div.upcoming-events-section div.row:first-child");  
  console.log(tables.length) // null or a single element object

Search for elements with a specified attribute:

* var inputtexttags = document.querySelectorAll("**input[type='text']**")

For browser compatibility see:

* https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelector#Browser\_Compatibility
* http://caniuse.com/#feat=queryselector

CSS Selectors:

* https://developer.mozilla.org/en-US/docs/Learn/CSS/Introduction\_to\_CSS/Selectors
* https://www.w3schools.com/cssref/css\_selectors.asp
* http://www.w3.org/TR/CSS2/selector.html

## Adding to the DOM

### Write HTML from Strings

???

### Creating New Elements

???

# Module 3: Introduction to jQuery

## jQuery

A lot of the coding work you will do with JavaScript to customize HTML will be to first find an HTML element and then manipulate that element. Often, all we are doing is some searching, or querying, of the HTML and then doing a quick update. jQuery is ideal for this.

jQuery:

* Is a library of JavaScript code.
* Is a single JavaScript function named “jQuery” and most often accessed with an alias of “$”.
* Is primarily used to “query” the DOM to find elements, and then perform an action against them.
* Includes a library of helper functions such as “$.ajax().

### First the negatives:

* jQuery is not part of JavaScript or the browser. You will either download the jQuery library from the web and upload it to your server, or you will directly link to a copy of the library on the web.
* The library is not large, but it will add to page load time, at least the first time a user visits a page. After the first visit the user's browser will use a local cached copy of the library.
* The names of common things (functions etc.) are not the same as JavaScript.
* It’s something else to learn.

### And then the positives:

* jQuery can create some very compact solutions, often with just one line of code.
* jQuery is just a single function call into a library so it can be intermixed with regular JavaScript - in fact all of jQuery is just one big JavaScript function named jQuery() or $().
* There are a lot of web resources for jQuery.
* jQuery is extensible. There are many add-in libraries available.

What follows is a very, very, brief overview of jQuery along with a few notes on using jQuery in SharePoint. For more on jQuery see:

* jquery.com - the official web site for jQuery
* "jQuery: Novice to Ninja" by Earle Castledine and Craig Sharkie (See Link 401)

## Downloading the library

Downloading your own copy of the library is often the preferred method of accessing the library, especially if you are not supporting internet users. You will then know the library is where you need it, will still be there when you need it, and that it will not change.

## jQuery Versions

For a long time jQuery had two parallel versions, the 1 series and the 2 series. The 2 series is faster and smaller, but does not support Internet Explorer before version 9.

In 2016 jQuery 3.0 was released. Like the 2 series, it does not support IE before version 9. While very compatible to older versions, there are a few breaking changes that could impact older code. See here: https://blog.jquery.com/2016/06/09/jquery-3-0-final-released/

Tip! Rename the downloaded jQuery library from its full name (jquery-1.4.2.min.js) to just jquery.js so when you download updated versions you won't have to edit all of your scripts.

## Linking to the library

Here's an example of a link to a jQuery library stored on your server:

<script src="/scripts/jquery-1.4.2.min.js" type="text/javascript"></script>

or if you follow the renaming tip above:

<script src="/scripts/jquery.js" type="text/javascript"></script>

This link can be used in each page or web part as needed, or loaded once from a master or layouts page.

**Linking to an external library:**

If you don't want a local copy of the library, you can link to a file stored in a common location on the internet. You can take advantage of Content Delivery Networks (CDNs) as public locations for commonly used files such as jQuery. Be aware that if you go this route, your users will always need access to the internet while accessing your site. While this may be a disadvantage for users on your local network, this can reduce load on your web servers when supporting thousands of internet users. But also consider that these remote sites are outside of your control and could be changed, compromised or deleted at any time. CDNs can be free, for a fee or supplied by your hosting service. Here's a sample of loading the library from Microsoft's AJAX CDN site:

<script src="http://ajax.microsoft.com/ajax/jquery/jquery-1.4.2.min.js" type="text/javascript"></script>

**Where should you put the link to the library?**

If you only need the library "here and there" then only load it as needed in the page that needs it. If you will be using jQuery in many pages then consider adding the library link to the site's master or layouts page. Whichever you do, don't do both or you will get errors.

To add the library in a Content Editor Web Part, add the script link just before your script area:

<script src="/scripts/jquery-1.4.2.min.js" type="text/javascript"></script>

<script type="text/javascript">

//your jQuery code here

</script>

To add the library in the master page, just add the script link somewhere in the <HEAD> section of the page.

**Want to see it work?**

If you are impatient and want to see some jQuery in action then consider how you would discover and highlight all of the tables in a page. You could write a JavaScript routine to loop through all of the tables and then turn on borders and set the width and color.

Here’s a JavaScript only way to style each table.

var tables = document.getElementsByTagName("TABLE")

for (i=0;i<tables.length;i++)

{

tables[i].style.border="solid red 2px";

}

Here’s a jQuery JavaScript hybrid. jQuery returns an array of tables, and normal JavaScript styles each table. We are using jQuery ($) to with a CSS style selector to find the table elements.

var tables = $("table")

for (i=0;i<tables.length;i++)

{

tables[i].style.border="solid red 2px";

}

Here's a “pure” jQuery example to do the same style change:

$("table").css("border","solid red 2px");

The "$" is a shortcut for "jQuery". Typing "jQuery" in place of the "$" will also work. Basically the above says find all "table" elements (tags) and set their CSS border property.

## Selecting Elements using jQuery

jQuery is a query tool, and searching for and retrieving HTML objects is what jQuery is best at. When searching for tags based on CSS properties, jQuery uses the same notation as CSS to select elements: "." for classes, "#" for IDs etc.

Tags are selected by adding the tag in quotes:

$("table")

Tags with IDs are selected using "#":

$("#someID")

Tags assigned to a CSS class are selected using ".":

$(".someClassName")

Tags can be selected with combinations of selectors just like in CSS:

$(".ms-sitetitle A")

(This selects all anchor tags inside of elements assigned the class named ms-sitetitle)

## Using the data returned by jQuery

jQuery will return an array of all of the matched HTML elements. You can store the results of a "query" in a variable or directly update all of the objects found in a single step.

For example, both of the following produce the same result. The example first finds all tables and applies CSS formatting to all of the found elements. The second example finds all tables, stores the result as an array of tables, and then loops through the array to apply CSS properties to the tables.

$("table").css("border","solid red 2px")

var tables = $("table")

for (var i=0; i<tables.length; i++)

{

tables[i].style.border="solid red 2px"

}

Important! When performing a jQuery query ($(selector)), jQuery always returns a collection, even if it found one or zero items. The jQuery methods deal with this and do not raise an error. ($("table").css("border","solid red 2px") will not raise an error if there are no tables or only one table.) If you really need to return a single object, then use: var x = $("#*someid*").get(0);

## Setting CSS properties

To set a single property:

$( *selector* ).css("*propertyName*","*propertyValue*")

$("table").css("border","solid blue 2px")

To set multiple properties (note the curly brackets and the colons):

$( selector ).css( { "propertyName" : "propertyValue" , "propertyName" : "propertyValue" , "propertyName" : "propertyValue" } )

$("table").css( { "border-style":"solid", "border-color":"blue", "border-width":"2px" } )

## Running a function against the return set

jQuery does not have a built in function for everything you can think of. For example, if you need to do a double search and replace ("<" and ">" in the example here) you might write an anonymous function and call it from one of the jQuery methods. This will call the function for each item found by jQuery.

$('.ms-gridT1').html(function(index,oldhtml) {

return oldhtml.replace(/&lt;/g,'<').replace(/&gt;/g,'>')

})

or with a little better formatting:

$('.ms-gridT1').html(

function(index,oldhtml)

{

return oldhtml.replace(/&lt;/g,'<').replace(/&gt;/g,'>')

}

)

If you do a search at jQuery.com for ".html" you will find that it can call a function:

.html( function(*index*, *oldhtml*) )

Here "index" is the position of the item in the return set and "oldhtml" is the original value of the HTML in the element in the returned item in the set.

## Useless Trivia!

* According to builtwith.com, over 700,000 web sites are now using jQuery

# Module 4: Ajax and Web Services

## Ajax and JavaScript

???

## Ajax and jQuery

???

# Module 5: Diving in Deeper!

Three types of code:

* Global – code between script tags.
* Function – code within a function or in event attributes (onclick=”…")
* Eval – code executed from within an eval().

???

## Data Types

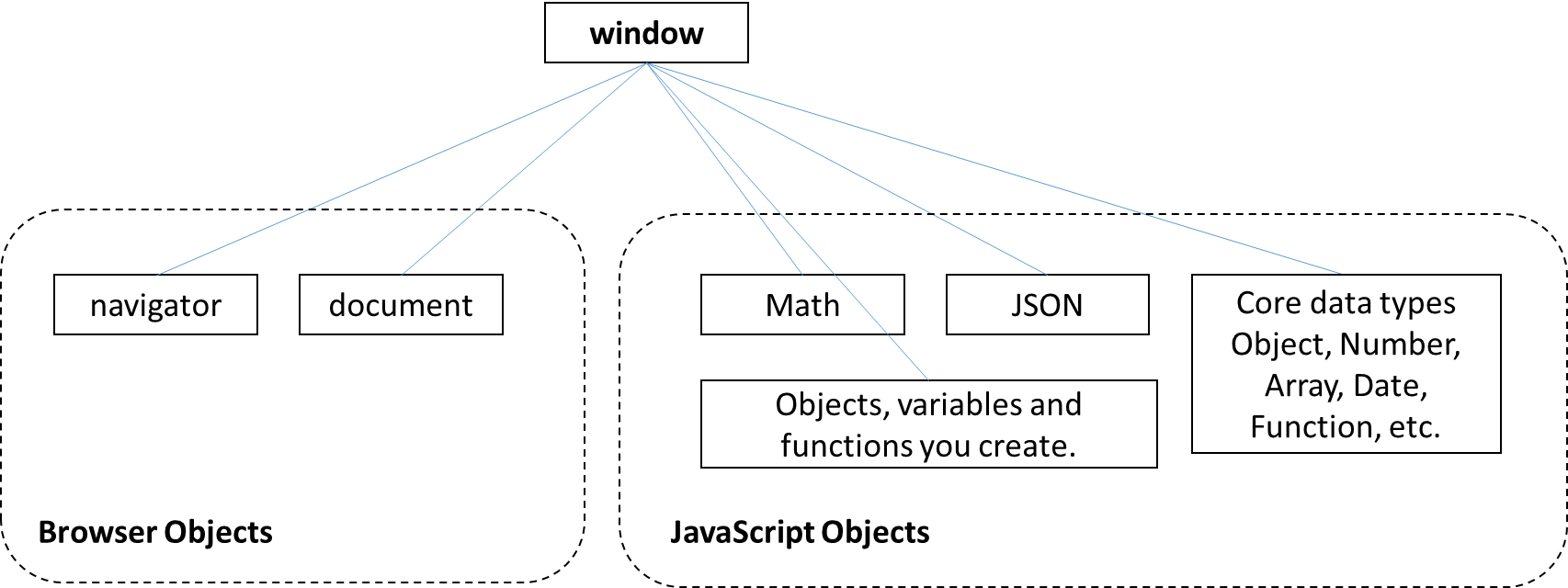
### JavaScript Primitives

A “primitive” is a simple type that only has a value.

* Values: (you cannot have a variable of type null!)
  + null
  + undefined
* Types: (you can have a variable of type boolean.)
  + boolean
  + number
  + string

### JavaScript Default Objects

When working with web pages, JavaScript runs in the context of a browser. The top most object is JavaScript is window. The window object has properties that contain all of the other objects that JavaScript has access to.



* window – The top most object. Contains all other object.
* window.navigator – The web browser and its configuration.
* window.document – The currently loaded document. This object is typically defined using an HTML file. The file is parsed to create the Document Object Model (DOM) that is accessible by JavaScript through the document object.
* window.Math – JavaScript’s math library (random, sine, sqrt, etc.)
* window.JSON – JavaScript’s JSON / JavaScript Object Notation library.
* window.Object, window.Array, window.Number, etc. – the core data types.
* window.*yourObjects* – all other objects created by your JavaScript code.

window is the default object. You do not need to type “window” in front of any of the above objects.

Samples:

* navigator.userAgent // "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:51.0) Gecko/…"
* navigator.userLanguage // "en-US"
* Math.sqrt(2) // 1.4142135623730951
* JSON.stringify(bankaccount) // {"AcctNum":1234,"Type":"Saving","Closed":false," … }
* document.doctype // <!DOCTYPE html>
* document.head.title // "Your site title"  
  document.title // "Your site title"
* document.getElementsByTagName("h1") // *collection of all H1 elements*

### Window Methods

The window object contains all of the built-in JavaScript methods. In the list below, each method could be writing with the window object and a dot. Example: window.alert("hello world");

A partial list of frequently used window methods.

* alert("*message*") – displays a popup message.
* confirm("*message*") – displays a popup message with OK and Cancel buttons. Clicking OK returns true and clicking Cancel returns false.
* prompt("*message*","defaultValue") – Prompts the user for a string value. Clicking OK returns the string and clicking Cancel or pressing Esc returns null.
* console – returns the console object that displays messages to the browser’s console.
  + console.log("*message*") – writes a message to the console.
  + console.warn("*message*") – writes a warning message to the console.  
     console.warn("here be dragons") 
  + console.group() – indents all following console output. (can be nested)
  + console.groupEnd() – unindents all following output.
  + console.time("*timername*") – creates a timer.
  + console.timeEnd("*timername*") – displays the elapsed time and removes the timer.
  + console.trace() – displays the call stack to this point in time.
  + See: https://developer.mozilla.org/en-US/docs/Web/API/Console
* print() – opens the print dialog box for the current window.
* Also see: https://developer.mozilla.org/en-US/docs/Web/API/Window

### Navigator Properties

The navigator object supplies information about the browser, and services supplied by the browser like geolocation. Many of these properties are “non-standard” and are inconsistently implemented across browsers. As an example, Internet Export, Firefox and Chrome all return “Netscape” for the navigator.appName property and they all return “Gecko” for navigator.product property.

* navigator.userAgent – The user agent string sent to the web server to identify the browser. (These are often untrustworthy. navigator.userAgent is being removed from the standards.)
  + IE 11: "Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; Zune 4.7; MS-RTC LM 8; .NET4.0E; InfoPath.3; rv:11.0) like Gecko"
  + Firefox: "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:51.0) Gecko/20100101 Firefox/51.0"
  + Chrome: "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/57.0.2987.133 Safari/537.36"
  + Notes:
    - Each of the above claim to be Mozilla/5.0!
    - Each gives hints as to their real id: “Trident” is the IE engine and “rv:11.0” is the browser version. IE says “like Gecko” while Firefox says “Gecko”.
    - Some specialized browsers and some search engines “lie”!
* navigator.oscpu – Returns the version of the operating system. (Not supported by IE 11 or Chrome.)
* navigator.onLine – Returns true if the browser has access to a network.
* navigator.language – Returns the language set in the browser’s user settings. Example: “en-US” for US English.
* navigator.languages – Returns an array with all of the user’s preferred languages. navigator.language is the first element of the array.
* See: https://developer.mozilla.org/en-US/docs/Web/API/Navigator

### JavaScript Objects

JavaScript objects have properties. Each property has a name and can contain a value (a primitive), function, or another object.

Properties can be defined:

* When the object is created: var account = { AcctNum: 1234, Type:"Saving" };
* At any time: account.Balance = 1234.5;

Properties can be:

* A primitive value.
* An object
* A function

Objects can be created as “object literal”:

* An empty object:  
   var obj = {};
* An object with primitive properties:  
   var bankaccount = { AcctNum: 1234, Type: "Saving", Closed: false, Balance: 1234.5 };
* An object with both primitive and object properties:  
   var customer = { CustNum: 1000, Account: bankaccount, Manager: "Jones" };
* Note: The curly bracket notation used here is similar to the JSON format. JSON strings are often returned by web services and can be quickly converted to a JavaScript object using JSON.parse().
  + var jsonstring = '{ AcctNum: 1234, Type: "Saving", Closed: false, Balance: 1234.5 }';  
    var bankaccount = JSON.parse(jsonstring);

Objects can be created using the “new” keyword:

* new Object();
  + var bankaccount = new Object();  
    bankaccount.AcctNum = 1234;  
    bankaccount.Type = "Saving";  
    bankaccount.Closed = false;  
    bankaccount.Balance = 1234.5;

The “new” keyword can be used to create primitives and other objects, but is rarely used.

* new Array();
* new Boolean();
* new Date();
* new Function();
* new Number();
* new RegExp();
* new String();

### Accessing Object Properties

JavaScript supports two ways of accessing an object’s properties, dot notation and square brackets. Due to the square brackets, an object can also be treated as a container of name/value pairs and can even be enumerated in a for loop.

* var bankaccount = { AcctNum: 1234, Type: "Saving", Closed: false, Balance: 1234.5 };  
  var customer = { CustNum: 1000, Account: bankaccount, Manager: "Jones" };
  + customer.CustNum // 1000
  + customer[“CustNum”] // 1000
  + customer.bankaccount.Balance // 1234.5
* Retrieving all of an object’s properties: (Excluding prototype properties.)
  + var props = Object.getOwnPropertyNames(customer);   
    for (var v in props) { console.log(props[v]) }
* The existence of a property can be tested:
  + bankaccount.ABC == undefined // false
  + Note: If bankaccount.ABC is set to null then:
    - bankaccount.ABC == undefined // true
    - bankaccount.ABC === undefined // false
* New properties are added dynamically:
  + bankaccount.lastCheckNumber === undefined // true  
    bankaccount.lastCheckNumber = 123;  
    bankaccount.lastCheckNumber === undefined // false

### JavaScript Numbers

JavaScript only has one numeric data type, “number”.

Number FAQ:

* All numbers are 64 bit floating point numbers.
* Floating point numbers “***approximate***” decimal values.
  + .333\*100 = 33.300000000000004
  + 1/3 = 0.3333333333333333, but 10/3 = 3.3333333333333335
  + 3.3333333333333333 \* 3 = 10
  + 3.3333333333333335 \* 3 = 10
  + 3.3333333333333329 \* 3 = 10
  + 3.3333333333333339 \* 3 = 10.000000000000001
  + n = Math.sqrt(2) // 1.4142135623730951  
    n \* n = 2.0000000000000004
  + Math.sqrt(2) \* Math.sqrt(2) == 2 // false
* Maximum value: 1.7976931348623157e+308 Minimum value: 5e-324
  + These two values can be retrieved using Number.MAX\_VALUE and Number.MIN\_VALUE.
* 15 places of precession.
  + 999999999999999 = 999999999999999  
    i.e. 999,999,999,999,999 = 999,999,999,999,999 (don’t type “,” in JS numbers!)
  + 9999999999999999 rounded to 10000000000000000  
    i.e. 9,999,999,999,999,999 rounded to 10,000,000,000,000,000
  + 100000000000000001 gets rounded to 100000000000000000  
    i.e. 100,000,000,000,000,001 gets rounded to 100,000,000,000,000,000
* Up to 17 digits past the decimal point.
* The Math.round() function is useful for floating point number adjustments.
  + Math.round() rounds to the nearest whole number:
    - Math.round(3.49999) // 3
    - Math.round(3.5) // 4
  + To round to two decimal places, multiply the value by 100, round and then divide by 100.
    - Math.round(3.4444 \* 100) / 100 // 3.44
    - Math.round(3.6666 \* 100) / 100 // 3.67
* The .toFixed(n) method is useful when the result needs to be a **string** and have a fixed number of decimal places.
  + var n = 1.23456;   
    n.toFixed(2) // "1.23"   
    n.toFixed(3) // "1.235"
  + The string can be returned to a number using parseFloat() or parseInt().  
    parseFloat(n.toFixed(3)) // 1.235

#### Infinity

JavaScript numbers also define Infinity and -Infinity. Numbers that exceed Number.MAX\_VALUE and Number.MIN\_VALUE return Infinity.

* Division by zero does not raise an error in JavaScript, it returns “Infinity”.
* Infinity is a number!
  + typeof Infinity; // “number”
* Infinity +, -, \*, / any number is also Infinity
  + Infinity \* 2 // Infinity

#### NaN

Invalid numeric operations return a special type called “Not a Number”, or NaN.

* NaN never equals anything!
  + NaN == 5 // false
  + NaN == NaN // false
  + NaN == null // false
* NaN is a number!
  + typeof NaN; // “number”
* Examples that return NaN:
  + 5 \* “a” // NaN
  + Math.sqrt(-1) // NaN

#### Number variables have methods

Everything in JavaScript is an object, and all objects have properties and/or methods.

* toExponential() – returns a string using exponential notation.
  + (12345.12).toExponential() // "1.234512e+4"
* toFixed(*places*) – returns a number as a string formatted to a fixed number of decimal places.
  + (12345.12345).toFixed(2) // "12345.12"
* toLocaleString(*locale*) – returns a number as a string formatted for a locale.
  + (12345.12).toLocaleString("fr") // "12 345,12"
  + (12345.12).toLocaleString("us") // "12,345.12"
  + (12345.12).toLocaleString("de") // "12.345,12"
* toLocaleString(*locale,options*) – returns a number as a string formatted for a locale.
  + (12345.12).toLocaleString("us", { style: 'currency', currency:'USD' } ) // "$12,345.12"
  + (12345.12).toLocaleString("de", { style: 'currency', currency:'EUR' } ) // "12.345,12 €"
* toPrecision() – Returns a string to the number of digits of precision.
  + var x = 5.123456; x.toPrecision(1); // "5"
  + var x = 5.123456; x.toPrecision(3); // "5.12"
  + var x = 5123456; x.toPrecision(3); // "5.12e+6"
  + var x = 5; x.toPrecision(3) // "5.00"
* toString() – returns a string representing the number.
* valueOf – returns the value of the variable.
  + var n = 5;  
    n.valueOf(); // 5

#### To and From HEX

* Numbers can be written in hexadecimal format by prefixing the value with “0x”:
  + var num = 0x10 // 16
  + var num = eval(“0x10”) // 16
  + var num = parseInt(“0x10”,16) // 16
  + Tip: parseInt(*str*,*base*) can be used to convert a string to a number.
    - var num = parseInt(“1100”,2) // 12
    - var num = parseInt(“14”,8) // 12
* Numbers can be converted to a hexadecimal string using .toString():
  + var hexstring = num.toString(16) // “0x10”
  + Tip: .toString(*n*) can convert a number to any base! (“n” must be 2 to 36)
    - n=12;
    - n.toString(2) // 1100
    - n.toString(8) // 14
    - n.toString(10) // 12
    - n.toString(16) // c

#### The Math object

JavaScript’s Math object includes a list of constants and math functions.

Constants: E, LN10, LN2, LOG10E, LOG2E, PI, SQRT1\_2, SQRT2

Functions: abs, acos, asin, atan, atan2, ceil, cos, exp, floor, log, max, pow, random, round, sin, sqrt, tan, min

### JavaScript Strings

#### Escape Characters

You can embed special characters into strings using a backslash followed by a character.

* \0 Null
* \' Single quote 'It\'s raining!'
* \" Double quote "He said \"It's raining!\""
* \\ Backslash "Enter Unicode characters as \\uxxxx.”
* \n New line "this is on the first line \nthis is on the second line"
* \r Carriage return "this is overwritten by \rthis text" (may behave the same as \n.)
* \v Vertical tab Generally only applies to printers.
* \t Tab
* \b Backspace
* \f Form feed Generally only applies to printers.
* \uxxxx Unicode character "\u0061" = “a”
* Backslash followed by a carriage return lets you break long lines of text. The following creates a single long string:
  + var text = " Lorem ipsum dolor sit amet, consectetuer adipiscing elit. \  
    Maecenas porttitor congue massa. Fusce posuere, magna sed pulvinar \  
    ultricies, purus lectus malesuada libero, sit amet commodo magna eros quis urna.";  
    console.log(text);
  + Note: There must not be any other characters between the backslash and the carriage return.

#### String Properties

* .length – Number of characters in a string. (Not the number of bytes! JavaScript typically stores each character as two bytes.)

#### String Methods

Here are a few of the string methods:

* charAt(*i*) Returns the character at a specified index or position.
* includes(*str*) Returns true or false if *str* is found in the string.
* endsWith(*str*)
* indexOf(*str*)
* lastIndexOf(*str*)
* match(*regEx*) Regular Expression match.
* repeat(*i*)
* replace(*regEx|str, str*)
* search(*regEx*) Regular expression match.
* slice(beginIndex,endIndex) Extracts a substring by start and end positions.
* split(*str*) Splits a string into an array.  
   "red, gren, blue".split(",") = ["red","green","blue"]
* startsWith(*str,i*) Returns true if the string starts with *str*. “*i*” is an optional start position.
* substr(*i, cnt*) Returns the string starting at *i* and including *cnt* characters.
* substring(*i, i*) Returns the string between to indexes.
* toLowerCase() Returns a lowercase version of the string.
* toUpperCase() Returns an uppercase version of the string.
* trim() Returns a string with white space (spaces, tabs, etc.) removed from both ends.
* For more string methods see: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/String

Notes:

* Strings are indexed start with 0. A string with a length of 10 has character 0 through character 9.
* .slice() can use negative numbers to count from the end of a string.
* Strings are immutable. They cannot be changed, rather they are copied with any modification and the original then disposed.

#### String Concatenation vs. an Array of Strings

In some cases you can speed up string concatenation by using an array of strings, followed by a .join(). Running the sample below in your collection of browsers may produce some surprising results. In tests may at the time this writing, an array is faster in IE while string concatenation is faster in Firebox. Regardless of the browser, an array.push() was about the same speed as string concatenation.

// string concatenation

t1 = window.performance.now();

var cnt = 1000000

var s = "";

for (i = 0; i<cnt; i++)

{

s = s + "0123456789";

}

t2 = window.performance.now();

console.log("Time for plus: " + ( t2-t1 ));

// adding strings as new array elements.

t1 = window.performance.now();

var a = [];

for (i = 0; i<cnt; i++)

{

a[i] = "0123456789";

}

var s = a.join("");

t2 = window.performance.now();

console.log("Time for array: " + ( t2-t1 ));

// using array.join()

t1 = window.performance.now();

var a = [];

for (i = 0; i<cnt; i++)

{

a.push("0123456789");

}

var s = a.join("");

t2 = window.performance.now();

console.log("Time for array: " + ( t2-t1 ));

### JavaScript Dates

???

### JavaScript Booleans

???

### “this”

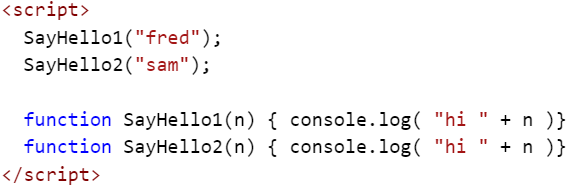
???

### Is JavaScript Compiled?

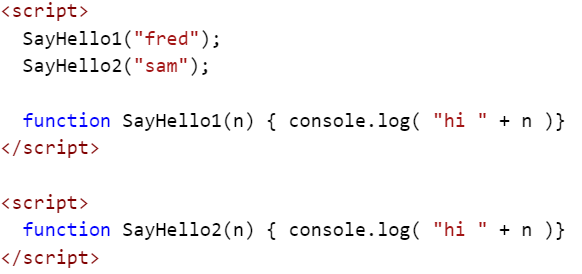
JavaScript code is reparsed each time the page is loaded. No binaries are created and saved anywhere. (.NET compiles to .DLL files.) So is it interpreted or compiled? Early versions of JavaScript were pure interpreters that parsed and executed the JavaScript code as it was read. Modern browsers optimize JavaScript by “just in time” compiling script blocks and then executing the compiled version. As a JavaScript developer you should probably be thinking of JavaScript as being interpreted.

#### Do I need to define a function before calling it?

When a script is parsed it is compiled “just in time” (JIT) and then executed. In the example below, both functions will be properly called, even though they are defined after where they are called.

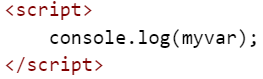


But, if you have two scripts (files or script blocks) they are compiled and executed one at a time. In the next example, the call to SayHello2 will fail as the second script has not been compiled before the first script was run.

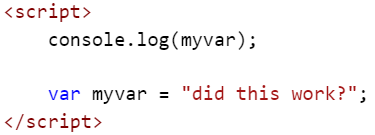


What about variables?

This gets interesting… The following code generates an error: “’myvar’ is undefined”. This is expected as it is not defined anywhere.

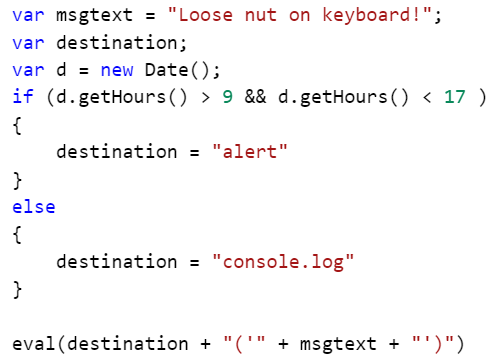


The next code sample does not generate an error! The message “undefined” was written to the console. The “compile pass” found the variable, but the “execute pass” has not yet found the variable initialized with a value. When “console.log” was executed, the variable “myvar” existed, but did not have a value.

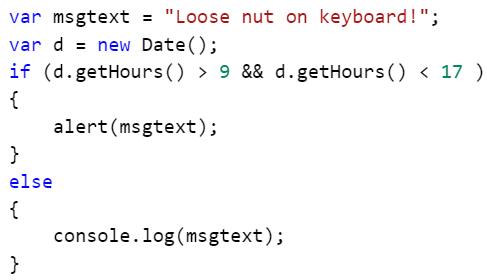


#### eval()

One aspect of JavaScript that is pure interpreter is the eval() function. Eval takes a string, parses it and then executes it.



Avoid eval() where ever possible. It’s too easy to create unexpected code and in cases opens up your code to hackers. The above example can be rewritten into a much better form.



Notes:

* The .compile() method does not compile code! It compiles regular expressions. It was deprecated in JavaScript version 1.5.
* Google has a tool named “Closure Compiler” that rewrites and compacts JavaScript code to reduce download size. It does not compile code!
* There are a number of languages that compile into JavaScript.
  + List of languages that compile to JS  
    <https://github.com/jashkenas/coffeescript/wiki/List-of-languages-that-compile-to-JS>

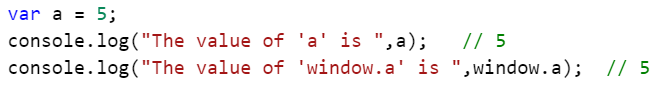
## Comment

JavaScript supports two kinds of comments:

* // Ignore to end of line.
  + var x = 5; // 5 must be between 2 and 7!
  + var z = x/y; // if y = 0 then z = Infinity
* /\* … \*/ Comment block.
  + /\*  
     this is  
     a multiline  
     code block  
    \*/
  + var x = 2; /\* this is an inline comment block \*/ var y = 3;

## Variables

In most languages, a variable is a pointer to a location in memory. In JavaScript, a variable is a property of some object. Global variables are properties of the window object.



### Scopes

In JavaScript, variables have up to three levels of scopes.

* Global
* Function
* Block {}

#### Global Scope

Variables are added to the global scope when:

* Defined outside of any function with or without “var”.
* Defined within a function without using “var”.
  + var a = 123; // global  
    b =123; // global  
    function foo() {  
     var c = 123; // local to foo()  
     d = 123; // global  
    }
  + Variables created in functions without “var” are said to be “hoisted” to the global scope. These variables are properties of the window object.
  + If you add “use strict” to a scope, then attempts to create variables without declaring them with “var” will raise an error.

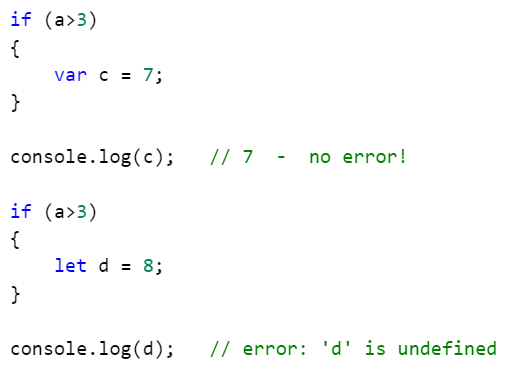
#### Function Scope

Variables are added a function scope when:

* Declared within the function’s code block ({}) using “var”.

#### Block {} Scope

ECMAScript 2015 (6) adds a “let” keyword that can be used in place of “var” to create variables that are scoped to a code block ({}) such as an “if” or “for”.



“let” is supported in the latest versions of most browsers. Earlier browsers will report an error (“Expected ‘;’” or similar) on lines with “let”, so use only where you know which browsers will be used. To see browser support see: <http://caniuse.com/#search=let>

### Hoisting

#### Variables

Variables created without “var” are automatically created at the Global scope, i.e. as properties of the window object.

function sayhello(txt)

{

h = "Hello ";

var c = "! And welcome to New York!";

return h + txt + c;

}

console.log(h); // outputs “Hello” as “h” has been “hoisted” to the Global scope.

console.log(c); // raises an error as “c” is not available outside of the function.

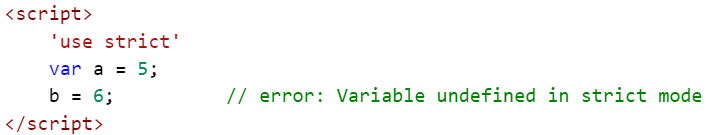
***Hoisting of improperly declared variables is one of the most common new developer errors. Always declare variables using “var”.***

#### Functions

Default declarations of function create them at the Global scope, as properties of the window object. Exceptions include anonymous functions and functions defined inside of other functions.

### Strict Mode

To prevent the accidental creation of hoisted variables you can add ‘use strict’ to your code.



* Strict mode can be applied to entire scripts or within individual functions.
* Introduced with ECMAScript 5.
* See: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Strict_mode>
* For browser support: <http://caniuse.com/#feat=use-strict>

### Creating Variables

???

* Declaring a variable with the same name of an existing variable or function will overwrite the existing item.
  + Example of a variable overwriting a function:  
     function hello(name) { console.log("hello " + name) }  
     hello("Mike") // “hello Mike”  
     typeof hello // "function"  
     hello = 6;  
     typeof hello // "number" (the function has been replaced with a variable)
  + Even the built in functions can be overwritten.  
     typeof alert // "function"  
     alert = 5;  
     typeof alert // "number" (alert(*msg*) no longer works!)

### Deleting Variables

???

* Garbage collection is a background process and cannot be forced.
  + How garbage collection in JavaScript works:   
    https://developer.mozilla.org/en-US/docs/Web/JavaScript/Memory\_Management
* All variables are discarded when a new page loads.

???

delete

* You cannot not delete a variable declared using “var”.
  + This code:  
     function test() { var x = 123; console.log(x); delete x; console.log(x) }  
    returns:  
     123  
     123  
    i.e. “x” was not deleted.
* Properties can be deleted.
  + var obj = { a:123, b:234 }  
    console.log(obj.a); // 123  
    delete obj.a;  
    console.log(obj.a); // undefined
* Note: You can delete a variable that was “var” declared in a browser’s console. The console uses eval() to run your code.

set to null

set to undefined

## Functions

Functions or Methods?

Each language defines these terms a little differently.

* Often a “function” is defined as a standalone block of code with a name while a “method” is part of an object.
  + C# and Java only have methods.
  + C only has functions.
  + In “Functional Programming” a method has side effects and a function does not. (A function has no side effects when it returns same value every time it is called with the same parameters.
* In JavaScript, every function is a property of some object, either an object you created or the Global object (window). So, they could be methods.
* In JavaScript, they are named “function”, so we will call them functions in this course.
* Just to add to the confusion… functions are not only properties of objects, they are objects!

### Function FAQ

* A function is a JavaScript object and is stored as a variable.
* The function’s object has a .length property which is the number of parameters.
* The text of a function can be returned using .toString().
  + function DoubleIt(x) { return x\*2 }  
    DoubleIt(2); // 2  
    DoubleIt.toString(); // "function DoubleIt(x) { return x\*2 }"
* A function can be created from a string.
  + Example with 1 parameter:
    - var DoubleIt2 = Function("x","return x\*2")
  + Example with 2 parameters:
    - var AddThem = Function("a","b","return a + b;")

### Anonymous Functions

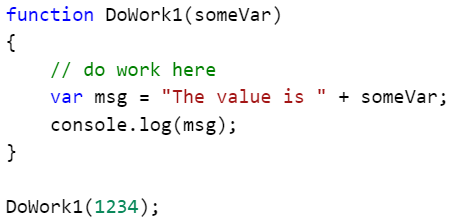
???

### Closures

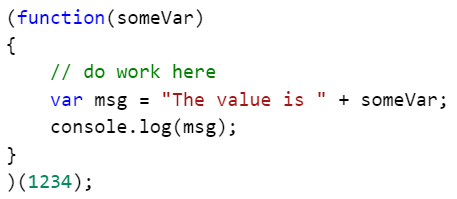
???

### Self-Invoking / Immediately Invoked Functions

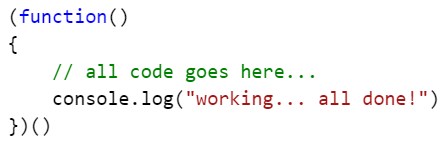
When it’s important that you do not add any new objects at the Global scope, you can create a Self-Invoking function. When creating a normal function you can wrap both variables and functions and only place one object on the global scope, the function itself.



A self-invoking function wraps an anonymous function in parentheses and calls it with () add to the end. The function object is created, called and then disposed, never being added to the Global scope.



Entire programs can be wrapped this way so as to not add any Globally scoped variables or functions.



## The Module Pattern

???

<http://www.adequatelygood.com/JavaScript-Module-Pattern-In-Depth.html>

## Errors

???

try catch

## Events

???

## Recursion

Simply put, recursion is a programing pattern where a function calls itself until an end condition is reached. As almost every recursive function can be rewritten using a while loop, the choice usually comes down to the data and the clarity of the solution. Recursive solutions are often clearer to read.

Examples of use:

* Calculating factorials.
* “Walking” XML trees, or the DOM.

Notes:

* Recursion is not an endless loop. There must be a planned for exit.
* Recursion can be memory intensive as each recursive call adds to the stack.
* JavaScript has limits on the number of recursive calls. Each browser has different limitations and varies from 1000 to over 20000.
  + When that limit is reached you may see:
    - “Maximum call stack size exceeded”
    - “too much recursion”
* Many recursive functions can be rewritten as a while loop.
* Depending on the browser and the function, loops can be many times faster than recursion. (This is mostly due to the stack overhead.)

### A Factorial Example

A factorial is the product of number and all integers below it. 6! = 6 \* 5 \* 3 \* 2 \* 1 What’s it good for? One use is to determine the maximum number of ways you can arrange a series of numbers.

3! = 3 \* 2 \* 1 which equals 6, so there are six different ways of arranging a collection of three things.

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 1 | 3 | 2 |
| 2 | 1 | 3 |
| 2 | 3 | 1 |
| 3 | 1 | 2 |
| 3 | 2 | 1 |

We need a function that accepts a number:

var factorial = function(num) {}

We need a test to see if we are done and an exit:

if (num <=0 ) { return 1}

And if not done we need to multiply the number by one less than itself. ( 3 \* 2 \* 1 );

else   
 {  
 return num \* factorial(num - 1);  
 }

Here’s the function and a use of it:

var factorial = function(num)

{

if (num <=0 ) { return 1 }

else { return num \* factorial(num - 1); }

}

factorial(3) // 6

factorial(4) // 24

factorial(6) // 720 this is the number of ways you can arrange 6 guests around a table.

factorial(52) // this is the number of ways you can shuffle a deck of cards!

And if you call it many times:

for (var i=1; i<=20; i++) { console.log( i, factorial(i) ) }

1 1

2 2

3 6

4 24

5 120

6 720

7 5040

8 40320

9 362880

10 3628800

11 39916800

12 479001600

13 6227020800

14 87178291200

15 1307674368000

16 20922789888000

17 355687428096000

18 6402373705728000

19 121645100408832000

20 2432902008176640000

The above example can also be written using while loop.

function factorial(num)

{

var result = num

while (num>1)

{

result = result \* (num - 1);

num--

}

return result;

}

# Appendix 1: Editors and Tools

## Visual Studio Code

Visual Studio Code is a free editor from Microsoft. Windows, macOS and Linux. While it has built-in support for JavaScript, TypeScript and Node.js, its real power comes from a large library of extensions.

Features:

* Multiple editors, both built-in and via extensions, including:
  + JavaScript, JSON, CSS, Sass, Less, TypeScript, Markdown, PHP, PowerShell, Python, Go, Dockerfile, T-SQL, C# and C++
* Coloring, auto indenting
* IntelliSense and autocomplete
* Snippets
* GIT support
* Debugging
* Integrated terminal
  + $SHELL on Linux and OS X, PowerShell on Window 10 and cmd.exe on earlier versions of Windows
* Many extensions



Links:

* Visual Studio Code: https://code.visualstudio.com/
* Download: https://code.visualstudio.com/Download
* Available Extensions: https://marketplace.visualstudio.com/VSCode

Getting Started with Visual Studio Code:

* JavaScript: https://code.visualstudio.com/Docs/languages/javascript
* HTML: https://code.visualstudio.com/docs/languages/html
* CSS: https://code.visualstudio.com/docs/languages/css
* GIT: https://code.visualstudio.com/docs/editor/versioncontrol

To add extensions:

1. Open the editor, click the View menu and then Extensions.
   1. A list of currently listed extensions will be displayed.
2. Search for an extension.
3. Click Install. (A few extensions may require a restart of VSCode.)