JAVASCRIPT

Overview
Data types
Primitive methods and control structures
jQuery

Bruno Oliveira: bmo@estg.ipp.pt
Marco Gomes: mfg@estg.ipp.pt
Miguel Andrade: mja@estg.ipp.pt

2017/2018

Programação em Ambiente Web

P.PORTO

ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

Overview

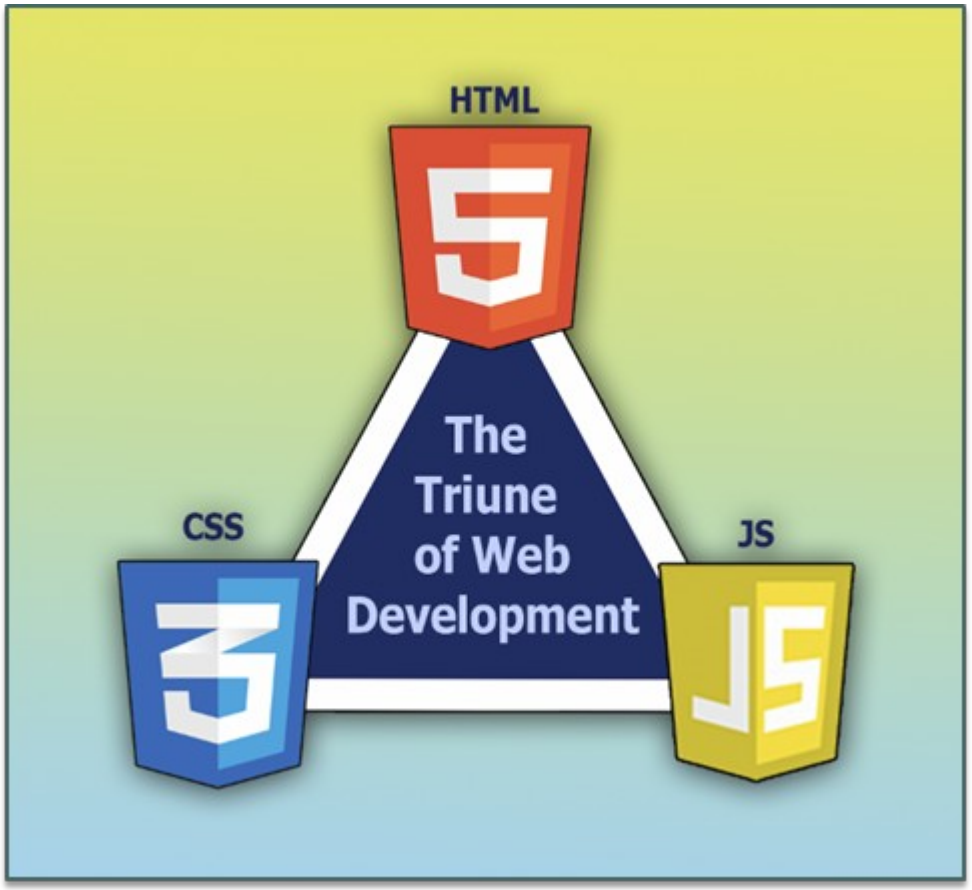
- Client-side programming language ("scripting language")
  - Used to make web pages interactive
  - insert dynamic text into HTML (ex: user name)
  - **react to events** (ex: page load user click)
  - get information about a user's computer (ex: browser type)
  - perform calculations on user's computer (ex: form validation)
- Client-side scripting (JavaScript) benefits:

usability: can modify a page without having to post back to the server (faster UI)

efficiency: can make small, quick changes to page without waiting for server

event-driven: can respond to user actions like clicks and key presses

Overview



Overview (Wikipedia)

As a multi-paradigm language, JavaScript supports **event-driven**, **functional**, and **imperative** (including object-oriented and prototype-based) programming styles.

It has an API for working with text, arrays, dates, regular expressions, and basic manipulation of the DOM, but the language itself **does not include any I/O**, such as networking, storage, or graphics facilities, relying for these upon the host environment in which it is embedded.

Initially only implemented client-side in web browsers, JavaScript engines are now embedded in many other types of host software, including server-side in web servers and databases, and in non-web programs such as word processors and PDF software, and in runtime environments that make JavaScript available for writing mobile and desktop applications, including desktop widgets.

Although there are strong outward similarities between JavaScript and Java, including language name, syntax, and respective standard libraries, the two languages are distinct and differ greatly in design

Overview

JavaScript is **not** Java!

A newsgroup wit once said that “Java is to JavaScript what ham is to hamster.”

Java was developed at Sun Microsystems. JavaScript was developed at Netscape.

Java applications can be independent of a Web page, whereas JavaScript programs are embedded in a Web page and must be run in a browser.

What JavaScript Is Used For

JavaScript programs are used to **detect** and **react** to **user-initiated** **events**, such as a **mouse** going over a **link** or **graphic**.

They can improve a Web site with **navigational aids**, **scrolling** **messages** and **rollovers**, **dialog** boxes, **dynamic images**, and so forth.

JavaScript lets you control the **appearance** of the page as the document is being parsed. Without any network transmission, it lets you **validate** what the user has entered into a **form** before submitting the form to the server.

What JavaScript Is Used For

JavaScript can **test** to see if the user has plugins, alert the user and send the user to another site to get the plug-ins if needed.

JavaScript has **string functions** and supports **regular expressions** to check for **valid** *e-mail addresses*, *Social Security numbers*, *credit card data*, and the like.

JavaScript serves as a programming language. Its **core language** describes such **basic constructs** as **variables** and **data types**, control **loops**, **if/else** statements, **switch** statements, **functions**, and **objects**.

What JavaScript Is Used For

JavaScript is used for **arithmetic calculations**, manipulates the **date** and **time**, and works with **arrays**, **strings**, and **objects**.

JavaScript handles **user-initiated events**, sets **timers**, and changes **content** and **style** *on the fly*.

JavaScript also reads and writes **cookie** values, and **dynamically** creates HTML based on the cookie value.

JavaScript can also handle **animation** and successfully used for producing web-based **applications** and **games**.

What JavaScript Is Like

Unlike Java, with JavaScript, there's **no natural place** to start building the code like we did down in the **main** method in Java (and this really confuses and disturbs more "procedural" programmer types). The web page itself can be viewed as a type of main.

Whatever you want to happen on the page with JavaScript, you either embed the code directly in the page to perform some action, or in a separate file that can be called up using a function when some action is performed (like clicking a link or button, dragging the cursor over an image, loading a page, etc).

JavaScript Where

JavaScript can appear in several places:

Inline (JavaScript inside a tag)

Internal (JavaScript in a <script> tag)

External (JavaScript in a separate file with a .js extension)

Dynamic (In an external file or service loaded by JavaScript)

The first three are somewhat similar to CSS

JavaScript Inline

Inline JavaScript appears inside an individual tag. The JavaScript commonly appears inside a *event attribute*, such as *onClick*

```
<!DOCTYPE html>
<html>
<head>
<title>Hello World 1</title>
</head>
<body>
<form>
<input type="button" value="Hello World" onClick="alert('Hello Yourself!')">
</form>
</body>
</html>
```

JavaScript Inline


```
<!DOCTYPE html>
<html>
<head>
<title>Hello World 2</title>
</head>
<body>
<p><a href="#" onClick="alert('Hello Yourself!')">Hello
World</a></p>
</body>
</html>
```

Notice how the *a* tag has two attributes (properties) with special values:
href attribute is "#" although it might also be empty " ", or contain "*javascript:*; "
- this disables normal link behavior

onclick attribute is *alert('Some Message');*

- *onclick* is an ***event*** — the JavaScript runs when the event happens, in this case when the link receives a click

JavaScript Inline


```
<!DOCTYPE html>
<html>
<head>
<title>Hello World 3</title>
</head>
<body>
<p><a href="javascript:alert('Hello Yourself!')">Hello World</a></p>
</body>
</html>
```

In this variation, the **JavaScript** is actually inside the **href** attribute. This is equivalent to the **onClick** example—we don't need to explicitly specify the "click" event, because the **href** takes care of that for us.

JavaScript Internal

Internal JavaScript appears inside a <script> tag, like this:

```
<!DOCTYPE html>
<html>
<head>
<title>Hello World 4</title>
</head>
<body>
<h2>Hello World</h2>
<script>
    // JavaScript goes here, between the opening and closing <script>
tags.
    // Notice use of "//" comment style while in between the <script>
tags.
    alert('Hello Yourself!');
</script>
</body>
</html>
```

JavaScript Internal

"Document" is the complete page object

```
<!DOCTYPE html>
<html>
<head>
<title>Hello World 5</title>
</head>
<body>
<h2>Hello World</h2>
<h2>
<script>
    document.write("Hello Yourself!");
</script>
</h2>
</body>
</html>
```

JavaScript Internal


```
<!DOCTYPE html>
<html>
<head>
<script>
    function popUp() {
        alert("Hello Yourself!") // note missing ; - this is ok
    }
</script>
</head>
<body>
<input type="button" onClick="popUp()" value="Hello World">
</body>
</html>
```

JavaScript External (file)

To use an external JavaScript file in a web page, use the <script> tag with the *src* attribute pointing to the JavaScript file.

Example:

```
<script src="somejavascript.js"></script>
```

When using the <script> tag to load an external JavaScript file, do **not** also use the tag as a container for internal JavaScript — that is, do **not** put JavaScript (or anything thing else) between the opening tag and the closing tag.

External JavaScript files are text files containing JavaScript, and nothing else.
Specifically, you do **NOT** need the <script> tag at the start.

Use your preferred (text) editor, or IDE!

When using two or more *script* tags on a page, the order of the tags can be significant, in terms of JavaScript processing.

JavaScript Inline vs external vs internal

Inline JavaScript can be useful for certain specific tasks, but inline should be your third choice. (except for declaring events)

External JavaScript files should be your first choice, **internal JavaScript** your second.

JavaScript Dynamic

Dynamic JavaScript (no, we didn't forget!)

It’s more a technique

Used to be called DHTML (D for Dynamic)

Basically, your JavaScript will change your HTML without browser refreshing the page.

Currently it’s more used to communicate with the server (or a service), pass some data and retrieve new content for the page. AJAX is a common use. It’s an acronym meaning Asynchronous JavaScript and XML. Lately we don't use XML anymore, we use JSON as a data format.

But more on that later … We will cover AJAX, JQUERY and Node.js during the course

JavaScript Variables

You create variables by **declaring** them. Then you assign values to them using the JavaScript **assignment operator**, the single **'='** symbol. When you name your variables, you need to follow the rules for naming variables in JavaScript, as well as consider the meaningfulness of the name.

Declaring Variables

To **declare** text as a variable, you use the **var** keyword (*var* is short for *variable*), which tells the browser that the text to follow will be the name of a new variable:

```
var variableName;
```

For example, to name your variable **numberOfOranges**, the declaration looks like this:

```
var numberOfOranges;
```

In this example, you have a new variable with the name **numberOfOranges**. The semicolon ends the statement. The variable **numberOfOranges** does not have a **value** assigned to it yet.

Using Case in Variables

JavaScript variables are case sensitive—**numberOfOranges**, **numberoforanges**, **NUMBERoFoRANGES** and **NumberOfOranges** are four different variables.

JavaScript Variables

You can give your variables a value at the same time that you declare them or you can assign them a value later in your script.

To assign a value to a variable, you use the JavaScript assignment operator, which is the equal to (=) symbol. If you want to declare a variable and assign a value to it on the same line, use this format:

```
var variableName = variableValue;
```

For example, to name your variable **numberOfOranges** and give it the numeric value **18**, use this statement:

```
var numberOfOranges = 18;
```

Here is what each piece of the code above does:

var *This is a special keyword in JavaScript that is used to define a variable.*
numberOfOranges *This names the variable numberOfOranges .*
= *This is the assignment operator, which assigns the value on its right side to the variable name on its left side.*
18 *This is the value that will be assigned to the variable, which in this case will be 18.*
; *The semicolon ends the statement, and the browser will move on to any additional statements in the code.*

JavaScript Variables

You can declare multiple variables or declare/assign multiple variables at the same time using only one **var**.

To assign a value to a variable, you use the JavaScript assignment operator, which is the equal to (=) symbol. If you

Without assigned values:

```
var variableName1, variableName2, variableName3;
```

With assigned values:

```
var variableName1 = 10, variableName2 =“doc”, variableName3 = 3.14159;
```

You might even do this, but show them on their own lines:

```
var variableName1 = 10;
var variableName2 = 20;
var variableName3 = 30;

alert(variableName1);
```

JavaScript Variables

Using Allowed Characters

An important rule to remember is that a variable name must begin with a **letter** or an **underscore** character (_) or a **dollar** character ($). The variable name cannot begin with a number or any other character that is not a letter (other than the underscore). The other characters in the variable name can be letters, numbers, underscores, or dollar characters. *Blank spaces* are not allowed in variable names

reserved words.

abstract	delete	goto	null	throws
as	do	if	package	transient
boolean	double	implements	private	true
break	else	import	protected	try
byte	enum	in	public	typeof
case	export	instanceof	return	use
catch	extends	int	short	var
char	false	interface	static	void
class	final	is	super	volatile
const	finally	long	switch	while
continue	float	namespace	synchronized	with
debugger	for	native	this	
default	function	new	throw	

JavaScript Variables

Data types

So far, you've seen examples of variable values that are numbers.

In JavaScript, the variable values, or types, can include **number**, **string**, **Boolean**, and **null**.

Unlike stricter programming languages, JavaScript does not force you to declare the type of variable when you define it.

Instead, JavaScript allows virtually any **value** to be assigned to any **variable**.

Although this gives you flexibility in coding, you need to be careful because you can end up with some unexpected results—especially when adding numbers.

JavaScript Data types

Numbers

JavaScript does not require numbers to be declared as integers, floating-point (decimal) numbers, or any other number type.

Instead, any number is seen as just another number, whether it is 7, –2, 3.453, or anything else. The number will remain the same type unless you perform a calculation to change the type.

For instance, if you use an integer in a variable, it won't suddenly have decimal places unless you perform a calculation of some sort to change it (dividing unevenly, for instance). As you've seen, you define a number variable by using the keyword var:

```
var variableName = number;
```

Here are some examples:

```
var payCheck = 1800;
var phoneBill = 35.50;
var savings = 1.50;
var spareTime = -24.5;
```

JavaScript Data types

Strings

String variables are variables that represent a string of text. The string may contain letters, words, spaces, numbers, symbols, or most anything you like. Strings are defined in a slightly different way than numbers, using this format:

```
var variableName = "stringText";
```

Here are some examples of string variables:

```
var myRide = "2011 Nissan Rogue";
var myOldRide = "2001 Nissan Frontier (currently in Arizona)";
var myPC = 'Dell Core i7 , 16GB RAM, 500GB SSD & 2TB SATA HD';
var myOldPC = "Dell Pentium 4, 6MB RAM, 500GB SATA HD";
var gobbledyGoop = "Wuzzup, dude? Groovy! I am @ home 4 now just
chillin...";
```

As you can see, strings can be short, long, or anything in between. You can place all sorts of text and other characters inside of string variables. However, the quotation marks, some special characters, and the case sensitivity of strings need to be considered (which I'll discuss momentarily).

JavaScript Data types

Strings

Matching the Quotation Marks

In JavaScript, you define strings by placing them inside quotation marks (quotes, for short), as you saw in the examples.

JavaScript allows you to use either **double quotes (" ")** or **single quotes (' ')** to define a string value. The catch is that if the string is opened with double quotes, it must be closed with double quotes.

```
var myRide="2011 Nissan Rogue";
```

The same goes for single quotes:

```
var myHomeTown='Mountlake Terrace, Washington';
```

BE AWARE: Trying to close the string with the wrong type of quotation mark, or leaving out an opening or closing quotation mark, will cause problems.

JavaScript Data types

Strings methods

methods: `charAt, charCodeAt, fromCharCode, indexOf, lastIndexOf, replace, split, substring, toLowerCase, toUpperCase`

```
var s = "Connie Client";
var fName = s.substring(0, s.indexOf(" ")); // "Connie"
var len = s.length; // 13
```

JavaScript Data types

Arrays

JavaScript arrays are used to store multiple values in a single variable.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:
var car1 = "Saab";
var car2 = "Volvo";
var car3 = "BMW";

Or,

An array can hold many values under a single name, and you can access the values by referring to an index number.

Creating an Array

Using an array literal is the easiest way to create a JavaScript Array.

Syntax:
var array_name = [item1, item2, ...];

JavaScript Data types

Arrays

Using the JavaScript Keyword new.

```
var cars = new Array("Saab", "Volvo", "BMW");
```

Arrays are objects

Arrays are a special type of objects. The typeof operator in JavaScript returns "object" for arrays.

But, JavaScript arrays are best described as arrays.

Arrays use numbers to access its "elements". In this example, person[0] returns John:

```
var person = ["John", "Doe", 46];
```

Objects use names to access its "members". In this example, person.firstName returns John:

```
var person = {firstName:"John", lastName:"Doe", age:46};
```

JavaScript Data types

Array methods

```
var a = ["Stef", "Jason"]; // Stef, Jason
a.push("Brian"); // Stef, Jason, Brian
a.unshift("Kelly"); // Kelly, Stef, Jason, Brian
a.pop(); // Kelly, Stef, Jason
a.shift(); // Stef, Jason
a.sort(); // Jason, Stef
```

methods: `concat, join, pop, push, reverse, shift, slice, sort, splice, toString, unshift`

push and pop add / remove from back
unshift and shift add / remove from front
shift and pop return the element that is removed

JavaScript Data types

Boolean

A **Boolean** variable is one with a value of **true** or **false**. Here are examples:

```
var RexCodes = true;
var RexIsNotCool = false;
```

Notice that the words true and false do not need to be enclosed in quotes. This is because they are **reserved words**, which JavaScript recognizes as Boolean values.

Instead of using the words **true** and **false**, JavaScript also allows you to use the number **1** for true and the number **0** for false, as shown here:

```
var RexCodes = 1; // 1 means "true"
var RexIsNotCool = 0; // 0 means "false"
```

Boolean variables are useful when you need variables that can only have values of **true** and **false**, such as in event handlers

JavaScript Data types

Null

Null means that the variable has no value. It is not a **space**, nor is it a **zero**; it is simply **nothing**.

If you need to define a variable with a value of null, use a declaration like this:

```
var variableName = null;
```

As with the Boolean variables, you do not need to enclose this value in quotation marks as you do with string values, because JavaScript recognizes **null** as a keyword with a **predefined value** (**nothing**).

Null variables are useful when you test for **input** in scripts, as you'll learn later on.

JavaScript Data types

Undefined

An **undefined** value is given to a variable that has not been assigned a value yet.

```
var myVar;
alert(myVar);
```

```
myVar = 200;
alert(myVar);
```

JavaScript Using variables

Now that you know how to assign values to variables, you will want to know how to use them later in your script. To use a variable, you simply type the variable name where you need to use its value. For example, you can pop up an alert message in the browser using **window.alert()**:

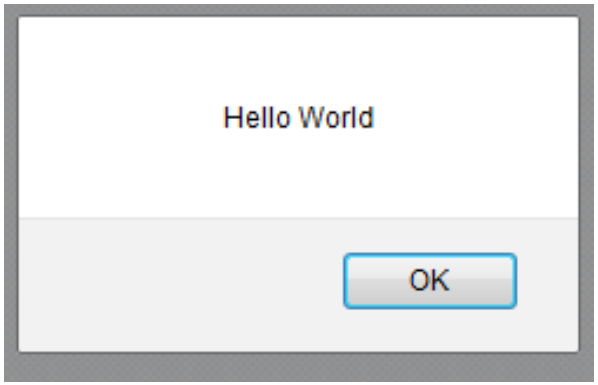
```
window.alert("Hello World");
```

This could be altered to use a variable instead, like this:

```
var message = "Hello World";
window.alert(message);
```

The result will be the same: an alert with the text **Hello World** in it.

Notice that you do not need quotes around the variable name since it was already defined as a string.



JavaScript Operators

Operators

Operators are the symbols used to work with *variables*. You're already familiar with operators from simple arithmetic; plus and minus are operators.

While both **x++** and **++x** add one to **x**, they are *not* identical; the former increments x *after* the assignment is complete, and the latter *before*.

For example, if **x** is 5, **y = x++** results in **y** set to 5 and **x** set to 6, while **y = ++x** results in both x and y set to 6. The operator **--** (minus sign) works similarly.

If you mix *numeric* and *string* values when adding two values together, the result is a string. For example, "catch" + 22 results in "catch22".

Operator	What it does
x + y (Numeric)	Adds x and y together
x + y (String)	Concatenates x and y together
x - y	Subtracts y from x
x * y	Multiplies x and y together
x / y	Divides x by y
x % y	Modulus of x and y (i.e., the remainder when x is divided by y)
x++, ++x	Adds one to x (same as x = x + 1)
x--, --x	Subtracts one from x (same as x = x - 1)
-x	Reverses the sign on x

JavaScript Comparisons

Comparisons

You'll often want to **compare** the value of one variable with another, or the value of a variable against a *literal value* (i.e., a value typed into the expression).

For example, you might want to compare the value of the day of the week to "Monday", and you can do this by checking if **todaysDate == "Monday"** (note the two equals signs).

Comparison	What it does
x == y	Returns true if x and y are equal
x === y	Returns true if x and y are identical
x != y	Returns true if x and y are not equal
x !== y	Returns true if x and y are not identical
x > y	Returns true if x is greater than y
x >= y	Returns true if x is greater than or equal to y
x < y	Returns true if x is less than y
x <= y	Returns true if x is less than or equal to y
x && y	Returns true if both x and y are true
x || y	Returns true if either x or y is true
!x	Returns true if x is false

JavaScript Assignments

Assignments

When you put a value into a variable, you are **assigning** that value to the variable, and you use the '=' assignment operator to do the job.

For example, you use the equals sign to make an assignment, such as **name = "Rex Winkus"**.

There are a whole set of assignment operators. Other than the equals sign, the other assignment operators serve as shortcuts for modifying the value of variables.

For example, a shorter way to say **x = x + 5** is to say **x+ = 5**.

Assignment	What it does
x = y	Sets x to the value of y
x += y	Same as x = x + y
x -= y	Same as x = x - y
x *= y	Same as x = x * y
x /= y	Same as x = x / y
x %= y	Same as x = x % y

JavaScript Functions

Functions

A function consists of the word **function** followed by the function **name**. There are always **parentheses** after the function name, followed by an **opening curly brace** (you'll hear me call this an "opening squiggle"). The statements that make up the function go on the following lines, and then the function is closed by a **closing curly brace** (you'll hear me call this a "closing squiggle"). Here's what an example of what a function looks like:

```
function sayWhat() {
    alert("JavaScript is Groovy, Man!");
}
```

JavaScript Conditionals

Conditional Code

A conditional tells code to run if something is true and possibly if something is not true

```
if (conditional) {
    /* then this part */
}
else {
    /* else this part */
}
```

Another way to say the same thing (shorthand):

```
(conditional) ? { /* then this part */} : {/* else this part */ }
```

Neither way is more “right” than the other.

JavaScript Loops

For loop

(same as java)

```
var sum = 0;
for (var i = 0; i < 100; i++) {
    sum = sum + i;
}
```

```
var s1 = "hello";
var s2 = "";
for (var i = 0; i < s.length; i++) {
    s2 += s1.charAt(i) + s1.charAt(i);
}
// s2 stores "hheellllloo"                    JS
```

JavaScript Loops

While loop

(same as java)

```
while (condition) {
    statements;
}
```

```
do {
    statements;
} while (condition);
```

JavaScript Math object

Math object

methods: `abs, ceil, cos, floor, log, max, min, pow, random, round, sin, sqrt, tan`

properties: `E, PI`

```
var rand1to10 = Math.floor(Math.random() * 10 + 1);
var three = Math.floor(Math.PI);
```

JavaScript Case

BE AWARE: JavaScript is **case sensitive**!

When you name a *variable* or *function*, pay attention to your uppercase and lowercase letters. **myFunction** is not that same thing as **MyFunction** or **myFUNCTION** or **myfunction**.

Also, you must refer to built-in objects with the proper casing. **Math** and **Date** start with uppercase letters, but not **window** and **document**.

Most built-in methods are combined words by capitalizing all but the first, such as **getElementById** (often referred to as *camelCase*).

JavaScript Comments

Comments

As with all programming languages—and as we've seen with Java—comments are an important part of the coding process even though they don't actually do anything in the code itself.

They are helpful hints for other people who might read your code. More often, they can remind you of why you wrote that weird piece of code a month ago.

Single-line comments look like this:

```
// This is a single-line comment
return true; // Comment after code
```

Multiline comments look like this:

```
/* This comment can wrap
into the next line  and even
the next line and the next */
```

JavaScript Spaces and newlines

White Space and Newlines

Most white-space such as spaces, tabs, and empty lines is ignored in JavaScript and usually just aids readability.

In fact, on large-scale production code, all nonessential white-space is usually stripped out so that script files download quicker.

Also known as “compressed”

Jquery

jQuery is a cross-platform JavaScript library designed to simplify the client-side scripting of HTML.

Recommended code Academy JQuery course: http://www.codeacademy.com

What is jQuery?

- jQuery is a library of JavaScript Functions.
- jQuery is a lightweight "write less, do more" JavaScript library.
- The jQuery library contains the following features:
- HTML element selections
- HTML element manipulation
- CSS manipulation
- HTML event functions
- JavaScript Effects and animations
- HTML DOM traversal and modification
- AJAX
- Utilities
- jQuery mobile extends jQuery support to mobile devices with additional mobile oriented functionality.

JavaScript

Jquery

Include via external file or from CDN

```
<script
  src="https://code.jquery.com/jquery-3.3.1.min.js"
  integrity="sha256-FgpCb/KJQlLNfOu91ta32o/NMZxltwRo8QtmkMRdAu8="
  crossorigin="anonymous"></script>
```

Start point

jQuery can be used event driven, or have a starting point

```
$(function () {
        // This anonymous function is the first function to be called when the page loads.
        // jQuery code, event handling callbacks here
});
```

Chaining

jQuery commands typically return a jQuery object, so commands can be chained:

```
$('div.test')
  .add('p.quote')
  .addClass('blue')
  .slideDown('slow');
```

Creating new DOM elements

Typical jQuery use, change something in the DOM

```
$('select#carmakes')
  .append($('<option />')
    .attr({value:"VAG"})
    .append("Volkswagen"));
```

Most simple description: Select something using CSS selectors, do something. (hide, show, alter content, animate ...)