JavaScript Notes

AKA: ECMAScript

HTML and CSS are DECLARATIVE languages; they declare what something is (HTML) and what that something is supposed to look like (CSS).

 If you actually want something specific to happen on a web page (such as opening a new window or processing a form), you will need to use an IMPERATIVE language, a language that says "do this!"

JavaScript is an "event-driven" programming language, which means that functions in JavaScript are triggered by events.

 What is an EVENT? An event, in JavaScript, is any user interaction with the computer.

 Clicking the mouse is an event.

Pressing a key on the computer keyboard is an event.

Moving the mouse around on the screen is an event.

Loading an HTML page in the browser is an event.

Any action initiated by the user constitutes an event.

RESOURCE is: a resource is any piece of digital media, whether text, sound, graphic, animation, form element, or CSS document, among others.

Resources can include individual pieces of code, functions, windows, dialog boxes, scrollbars, interface elements, etc. Almost anything in the computer, anything digital, can potentially be a resource which you may then access and/or manipulate in JavaScript.

User interacts with the computer 🡪 generates an event 🡪 events are really functions that, when triggered, access and manipulate a resource (ie. User interaction triggers JavaScript program that uses resources to provide user with some FEEDBACK)

USER INTERFACE, the user interface being that portion of the program which the user will interact with that includes both interface elements (such as buttons and text fields) as well as feedback (such as dialog boxes, screens, sounds, etc).

Imperative languages use LOGIC to make simple decisions and perform repetitive actions in a program.

OPERATIONS are used to perform mathematical calculations, comparisons, transfers of data, and other activities within the computer itself.

Data – stored information in VARIABLES

NOTE: JavaScript is case sensitive

Declaration and Initialization

When creating variables in an imperative programming language, there is a two-step process: 1) declaration (declaring a variable), and 2) initialization (initializing a variable).

When you DECLARE a variable, you tell the computer that that variable exists; you also tell it what that variable's name is.

When you INITIALIZE a variable, you fill that variable with data for the FIRST time (initial = first), setting it to its initial, or beginning, state.

How to declare variables in JavaScript?

var george;

Initialize a variable?

George = 1;

Gets Operator is “=”

COMMAND: instruction or set of instructions

NOTE: JavaScript is derived from the programming language called "C".

DATA TYPES

"primitive" data types: numbers, strings, and booleans.

NOTE: JavaScript DOES recognize a difference between integers and floats, and it has a few built-in features which allow you to manipulate numbers as either one or the other.

boolean: false and true, 0 and 1, off and on.

The values of false, 0, and off are all equivalent, while the values of true, 1, and on are also equivalent.

Beyond the primitive data types, and objects, there are two more data types you should be aware of: null and undefined.

The variable that is null contains NO information. You should know, however, that null is NOT 0; Zero (0) is an integer, a number. The null variable is empty.

null variable is differentiated from the undefined variable. An undefined variable has not been declared, and therefore does not exist, or has not been initialized, and so contains no data.

Typed and Untyped Languages

In JavaScript, variables may contain ANY type of data.

You can change the datatype of a variable at will. This is possible because JavaScript is an untyped language

In an untyped language, a variable may be of any data type, and an individual variable's data type may be changed at any time.

In a typed language, a given variable may only ever be ONE data type.

Every object in JavaScript has two main components: a set of PROPERTIES, and a set of METHODS.

Properties are generally passive in nature, describing the state of some feature of the object. Methods, on the other hand, are generally active, performing some sort of task.

constructor merely defines the basic structure of the object;

To get an object that you can use, you need to create an INSTANCE of the object (also known as INSTANTIATING an object).

When you create an instance of an object, you are creating ONE copy of the object which you have control over.

new SomeObject();

these parentheses are called the "function call" operator (which calls/invokes a function), and are an essential part of the object instantiation process.

A FUNCTION, in JavaScript, is a collection of lines of code, a program, which does something (like switch an image, or pop up a new window, or BUILD AN OBJECT, etc).

FUNCTIONS are also OBJECTS themselves; this is because almost everything in JavaScript is built out of objects.

Dot syntax is a way of accessing child elements of a parent object (remember parents and children?) by using period/dot (.) characters between parent and child elements to indicate a path through a complex tree of relationships, much as the slash (/) character is used in URL syntax to indicate a path through a complex directory structure on a web server.

Dot syntax always creates a path from the most general to the most specific element, reading from left-to-right, and, as you can see from the above example, there are NO SPACES IN THE NAMES of the elements or between the dots!

A METHOD is really a FUNCTION which is attached to an object, just as a PROPERTY is really a VARIABLE attached to an object (again, a function is a program which performs some action)!

I could access the image, fred, on my web page using something called the "simplified document object model", which allows me to use dot syntax to point to an element on an HTML page.

window.document.fred.src = picture0.src;

window is the instance of the Window object which represents the current web browser window.

document is the instance of the Document object which represents the current HTML page in the web browser window.

fred is the name of the IMG tag on the HTML page

src is the SRC attribute of the fred IMG tag.

NOTE: image switching / img switching / switch

 all images involved in a switch MUST have the same dimensions, the same width and height!

A VARIABLE may hold an INSTANCE of an OBJECT. Each object has PROPERTIES and METHODS. Properties are essentially variables attached to an object, while methods are essentially FUNCTIONS attached to an object.

JavaScript is an event-driven programming language, where JavaScript commands are triggered by events caused by user interaction.

JavaScript uses special EVENT HANDLERS inserted as ATTRIBUTES into ordinary HTML tags to access user events, such as clicking the mouse.

most event handlers MUST be placed into tags which can ordinarily accept mouse events, such as the A (anchor) tag, or the INPUT tag in a FORM.

<a href="#" onclick="var george = 1;">Link Word</a>

More than one command can be added

<a href="#" onclick="var george = 1; var fred = 2; var answer = george + fred;">Link Word</a>

Window Object

-Autodeclared

-instance refers to current window

<a href="#" onclick="alert('Hi');">Link Word</a>

Attributes may NOT have double-quote marks inside of the double-quote marks beginning and ending the value for the attribute, as this will break your HTML code. Because we NEED quote-marks of some kind to delimit the string value inside the function call operator for the alert method, we MUST use single-quote marks within the bounds of the double-quote marks marking the attribute value. When we write commands within ordinary JavaScript scripts, however, we will be able to go back to using double-quote marks to delineate string values; we'll talk about this later.

You don't have to use the onclick event handler from within an A (anchor) tag. Instead, you can use the javascript absolute URL as the value for the HREF attribute in order to trigger JavaScript commands.

Example:

<a href="javascript:alert('Hi');">Link Word</a>

**Note:** When using the javascript absolute URL, you must NEVER have any spaces in your JavaScript code; otherwise, the javascript URL will fail to operate.

The following example is WRONG because there are spaces in the URL:

<a href="javascript:var fred = 1; alert(fred);">Link Word</a>

If your JavaScript command requires spaces (as in the above example), you MUST then use the onClick event handler rather than the javascript absolute URL.

As I mentioned earlier, you do NOT have to put your onclick event handlers ONLY in A (anchor) tags; you may place the onclick event handler in ANY clickable tag, such as the INPUT tag in a form.

Example:

<form>  
<input type="button" value="Alert Me" onclick="alert('Hi There');" />   
</form>

The main SCRIPT tag for a page usually goes in the HEAD section of your HTML document, although it can also be placed within the BODY section. Between the opening and closing SCRIPT tags, you will write out your lines of JavaScript code.

The SCRIPT tag has ONE attribute, TYPE. The TYPE attribute tells the browser which programming language is being coded inside of the SCRIPT tag.

Note: There are currently 7 (seven) versions of JavaScript

<script type="text/javascript"> // Here's a couple of lines of JavaScript code: var george = 1; alert("Howdy"); </script>

Note: LANGUAGE is an old attribute that is replaced with TYPE

Note: The <!-- right after the script tag and the //--> right before </script> are used to comply with older browsers that do not have javascript enabled.

code inside the SCRIPT tag is executed as the HTML page loads.

An HTML page may, in fact, have several SCRIPT tags on it, both in the HEAD and in the BODY of the document.

JavaScript code may ALSO be coded into an external text file, called a ".js" file

<script src="abc.js"></script>

Note that some scripts won't validate for XHTML unless you include  
  
/\* <![CDATA[ \*/ right after the script statement and /\*]]>\*/ right before the closing script statement.

<script type="text/javascript">  
**/\* <![CDATA[ \*/**  
*// content of your Javascript goes here*  
**/\*]]>**\*/  
</script>

XHTML is subject to the same syntactical rules as XML. Because of this, an XHTML processor treats the characters < and & as markup, even if they reside inside a <script> block.

Follow following format:

<script type="text/javascript">  
**/\* <![CDATA[ \*/**  
*// content of your Javascript goes here*  
**/\*]]>**\*/  
</script>

NOTE: Ensure there is NO SPACE before text/javascript

OPERATORS

+ - = \ \* are operators

1 – n are operands

operators act on operands

WRITE METHOD

You use the write method of the document object to write to an XHTML page.

document.writeln(“msg”);

document.write("<span style = \"color: purple; font-size: 22px;\">XYZ</span>")

NOTE: \ is an escape character

PROMPT METHOD

You can obtain user input by using the prompt method of the window object.

IE DOES NOT LIKE PROMPTS: The problem with using prompts is that IE7 has basically disabled prompts for so-called security reasons. For this reason, I suggest avoiding prompts. If you do use IE7 and get a message to allow scripts and accept it, then you should refresh the page and the prompt will be normally be displayed.

number = window.prompt( "Enter a number", "" );

This will open a popup dialog box that allows the user to enter a string.

This will assign the user input to the variable 'number'.

To repeat, note that the text displayed by prompt dialog boxes can only be plain text, not XHTML formatted text.

CONVERT STRING TO INTEGER

var number = “12”;

var numb = parseInt( number );  //parseFloat

FORM: INPUT OUTPUT

You can obtain user input by using a form input text box, a form button, and the onclick event handler. This is the preferred way for handling multiple input.

<head>  
<script type="text/javascript">  
/\* <![CDATA[ \*/  
function process()  
{  
var number1, number2, n1, n2, sum;  
number1 = document.myform.num1.value;  
number2 = document.myform.num2.value;  
n1 = parseInt(number1);  
n2 = parseInt(number2);  
sum = n1 + n2;  
alert("The sum of the numbers is " + sum); }  
/\*]]>\*/  
</script>  
</head>  
<body>  
<div align="center">  
<form name="myform" action="">  
First Number: <input type="text" name="num1" size="10" />  
<br /><br />  
Second Number: <input type="text" name="num2" size="10" />  
<br /><br />  
<input type="button" onclick="process()" value="SUBMIT" />  
<input type="reset" value="RESET" />  
</form>  
</div>  
</body>

It should be noted that:  
number1 = myform.num1.value;  
works in IE but NOT in Firefox. Firefox REQUIRES it to be  
number1 = document.myform.num1.value;

Another way to obtain a value from a form input text box is to use the getElementById method. Assume that the form input text box is:

<input type="text" id="num1" size="10" />

number1 = document.getElementById("num1").value

PROBLEM WITH document.write() 🡺 usually puts output to a NEW page

SOLUTION: to output the results, using either form input text boxes or a form textarea. For example:

document.myform.result.value = sum;

Sum is: <input type="text" name="result" size="10" />

LOGIC

Computer logic is not like human logic, it's really more of a routing process, or a mechanization process. In fact, it is comprised of only TWO forms in JavaScript: BRANCH logic, and LOOP logic.

Branch logic allows for simple decisions to be made, based on a CONDITION. A condition can be anything, and is TESTED, returning a value of either true or false (a boolean!). IF STATEMENTS

There are really only three kinds of loops, with many, many variations.

Loop Type 1: Repeat an action WHILE, or for as long as, a condition is true.  
Loop Type 2: Repeat an action FOR a particular length of time, or FOR a given number of repetitions.  
Loop Type 3: Repeat an action until something occurs to stop it

for (myCounter = 0; myCounter < 3; myCounter++) {

sortShrimpIntoBasket();

}

Note: the local variable being declared here does NOT require the var keyword.

BASIC FUNCTION SYNTAX

function doStuff() { }

function

name of the function

list of parameters

NOTE: watch out for carriage returns. They could be interpreted as ;

Call functions from Event Handlers or javascript absolute url:

<a href="#" onClick="doStuff();">Link One</a>

<a href="javascript:doSomethingElse();">Link Two</a>

**Local variables** are temporary. A local variable is declared, initialized, and used for a short period of time, then thrown away. A local variable in JavaScript can only be used within a single function; no other functions have access to that particular variable; when a function is finished executing, the local variables for that function are thrown away.

**Global variables** are persistent. A global variable is created when your web page is loaded, can be accessed by many different processes and functions, and lasts until your web page is unloaded from the browser window.

**PROTECTED NAMESPACE – local variables can have same names in diff methods**

Global variable names start with the

Local variable names start with my

Functions PARAMETERS

function doStuff(fred) {

alert(fred);

}

function doStuff(fred, george) {

alert(fred);

alert(george);

}

window.open("http://www.yahoo.com/", "marian", "width=500,height=400");

1. The first argument is the URL (absolute or relative) that will appear in the new window.

2. The second argument is the NAME of the new window (which will NOT be visible to the user, but rather identifies the window in JavaScript).

3. The third argument is a series of parameters, describing the appearance of the new window; this third argument may NOT contain any spaces, nor will it contain any additional quotation marks.

To get a new browser window with more features, I have to state ALL of the features that I want, explicitly, in the third argument.

window.open("http://www.yahoo.com/", "marian","top=100,left=300,width=500,height=400,resizable,scrollbars,menubar,status,location,toolbar");

The window.open() method RETURNS a reference to the new browser window ITSELF.

JavaScript has a long list of event handlers which it understands. Some of the most commonly used event handlers are:

onclick

onmouseover

onmouseout

onchange

onload

onfocus

onblur

event handlers would be added to an HTML tag, and would be triggered upon that event occuring to the tag.

In order to support older browsers, most event handlers must be placed within clickable tags, such as A (anchor) and INPUT.

onClick, onMouseOver … will not validate for XHTML

Note:

When a page is completely loaded, when all of its graphics and text and sound files and movies are fully downloaded, the onLoad event is triggered. The onLoad event handler ALWAYS goes in the BODY tag of the HTML page.

When you SELECT something on the computer, you are giving that thing FOCUS. When you DESELECT something, you are giving it BLUR.

CREATING METHODS:

 The location property of the window instance of the Window object gives us access to the location bar, and will advance the page in the web browser when assigned to the desired URL.

window.location = "http://www.mozilla.org/";

There is a special keyword of the SELECT tag, selectedIndex, which returns the number of the option which the user has selected.  
  
Example:  
  
myForm.destList.selectedIndex

myForm.destList.options[myForm.destList.selectedIndex].value

var mydest = myForm.destList.options[myForm.destList.selectedIndex].value;

**Load-time errors**: An error that violates the grammatical rules of the language when the script is loaded.

**Run-time errors:** A run-time error occurs when the JavaScript interpreter encounters a problem during the execution of a program.

The most common type of run-time error occurs when you try to access an object or variable that doesn't exist.

Firefox:

Click Tools/Error Console/Errors to find JavaScript errors in Firefox. The line number and approximate position of the error is displayed. Note that you can also check for CSS errors by clicking Tools/Error Console/Warnings.

IE:

status bar in the lower left corner of the browser shows an error icon or says 'error on page' or 'done but with errors', the gray error box may appear or you may need to click on the error icon at the lower left of the page to see the JavaScript error and what line contains the error. For testing purposes, I suggest you have "Disable script debugging" as UNCHECKED, under Tools/Internet Options/Advanced on your browser.

Chrome:

In Chrome, click the "Control this page" button to the right of the address bar and select Tools/Javascript console. Then click on the error link at the right of the page to see which line has the error along with the corresponding error message.

Safari:

In Safari, you must enable the Develop menu. To do so, choose Edit/Preferences and then click the Advanced tab. There is a check box entitled "Show develop menu in menubar" that should be checked. Once the setting is enabled, a menu named "Develop" appears in the Safari menu bar. The Develop menu provides several options for debugging and otherwise working with the page that is currently loaded. You can click Show Error Console to display a list of JavaScript and other errors.

jQuery is a JavaScript library intended to make JavaScript programming easier and more fun.

A JavaScript library is a complex JavaScript program that both simplifies difficult tasks and solves cross-browser problems. In other words, jQuery solves the two biggest headaches with JavaScript—complexity and the finicky nature of different web browsers.

**jQuery Syntax**  
  
The jQuery syntax is tailor made for selecting HTML elements and perform some action on the element(s).  
  
Basic syntax is: $(selector).action()  
  
A dollar sign to define jQuery  
A (selector) to "query (or find)" HTML elements  
A jQuery action() to be performed on the element(s)

**The Document Ready Function**  
  
All jQuery methods are inside a document.ready() function:  
$(document).ready(function(){  
  
// jQuery functions go here...  
  
});  
  
This is to prevent any jQuery code from running before the document is finished loading (is ready).

Event handlers are method that are called when "something happens" in HTML. The term "triggered (or "fired") by an event" is often used.