Book 2: JavaScript - Basic Operations

Book for Weeks 2-3: Introduction to JavaScript.

Site: Insight2 @ CCSF
Course: CNIT133-META-Interactive Web Pages-Rubin-JavaScri-Spring 2014
Book: Book 2: JavaScript - Basic Operations
Printed by: Thuong Ho
Date: Monday, January 27, 2014, 3:29 PM

Table of contents

1 Introduction to JavaScript

Textbook References
(Remember that you do not need all four books. Select one that suits your level.)

Flanagan book: Ch. 1-5 and some pages in Ch. 6, 14, and 18

Gosselin book: Ch. 1-2, 8 and p. 140-148

McDuffie book: Ch. 1-5

Deitel book: Ch. 8

Today, virtually every major website uses JavaScript, also called ECMAScript, to enhance the functionality of HTML pages. Image-switching buttons, e-commerce shopping carts, forms processing functions, even the new Flash and Shockwave movies, require the use of JavaScript programs. To one degree or another, every web designer today MUST be able to use JavaScript, even if it is only to copy and modify someone else's code to use in a new webpage.

Because almost every site today uses JavaScript, it is important that YOU possess at least a fairly good knowledge of JavaScript and the fundamentals of computer programming as they apply to JavaScript.

Many of the JavaScript scripts used in today's websites are fairly simple; this is not likely to remain true for long, however, as many cutting-edge technologies (Ajax, Dynamic HTML, Flash, and others) rely on JavaScript programming to unlock their full potential. Unfortunately, not everyone can really learn to program successfully; if everyone could do it, companies wouldn't pay programmers such huge sums of money. Still, you must learn a fair amount of JavaScript in order to survive in today's competitive Internet marketplace.

In this course, I will endeavor to explain the concepts behind JavaScript and programming in general. Thus, you will be writing your scripts from scratch, instead of copy/modify/paste, as was done in CNIT 132.

We are now, for the most part, leaving behind the world of HTML and entering a more complex, more technical, environment which I will do my best to de-mystify and explain. I will be giving you as much detail as I can without obscuring the meaning of my explanations. This course is intended both as an introduction to programming, and as an explanatory reference for certain key aspects of JavaScript.

If you are seriously interested in a career as a web designer, you should purchase: *"JavaScript: The Definitive Guide"* by David Flanagan, O'Reilly and Associates, Publisher. This book is THE JavaScript reference book for programmers. It is both comprehensive and well written; you'll find it an indispensable resource for information about the language. If you have programming experience then you may find it easier to work in JavaScript with a copy of this book, as JavaScript is both enormous and filled with arcane details. If you lack programming experience then you should consider getting a less difficult book, such as the Gosselin, McDuffie or Deitel book.

For a cursory introduction to JavaScript, I recommend the tutorial, JavaScript for the Total Non-Programmer.

Finally when creating each javascript assignment, I suggest that it is usually helpful to completely work out your solution to the assignment on paper before actually typing any code on the computer.

2 Programming Overview: Terms and Concepts

Up to this time, you may have programmed solely in HTML and CSS, which 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!", a language, in short, like JavaScript.

Simple activities in JavaScript, like opening a new window or creating custom status bar messages, require minimal programming expertise; scripts for these sorts of things can be had in any decent JavaScript "cookbook". The purpose of these lectures is to present an overall concept of programming in JavaScript which will give you a foundation, upon which you can build, to permit you to go beyond the simple scripts.

In JavaScript, a program, called a FUNCTION, is a collection of lines of text-based code which performs some activity.

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.

You should already know what a RESOURCE is: a resource is any piece of digital media, whether text, sound, graphic, animation, form element, or CSS document, among others. Think beyond your current definition of resource. 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.

When you are programming in JavaScript, you will write FUNCTIONS which are triggered by user-initiated EVENTS; these functions will, in their turn, access and manipulate RESOURCES for various purposes. In other words, the user will interact with the computer, triggering a JavaScript program; that program will do something, then use resources to display something for the user to see, hear, or experience (this is called FEEDBACK).

You, as the programmer, are designing the feedback for the user, as well as figuring out how the program itself is supposed to work. You have to design every aspect of the user experience or 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).

When you are programming in JavaScript, you need to ask yourself what YOU would expect if you were using this device. Would it beep? Would the buttons hilite as you clicked them, and would you want to see rollover effects (like active/inactive button states)? What kind of response would you expect from the computer once the program is finished executing? Would you want to see dialog boxes? Should the HTML page advance to a new page? Should a window open? And what would really IRRITATE you? What do you want to make sure DOESN'T happen? You, as the programmer, must define every aspect, every detail, of the UI (User Interface), including every aspect of how it behaves.

But remember that old adage, KISS: "Keep It Simple, Stupid!" Don't include flashy highlights and

rollovers that distract from the functionality of the program. A good interface is almost invisible, assisting the user without being obtrusive.

Having said all of this, the user interface is, in some respects, merely the clothing on the living creature of your program. The real flesh and bone of a program lies in the calculations, the operations, the decisions being made by the program itself which actually do the task at hand. You can obsess about what your buttons are going to look like, or what your dialog boxes are going to say, until your hair falls out, but if you haven't got code inside of your program deciding something or solving something or calculating something or moving something around, you're not going to have anything useful in the end.

Imperative programming languages, such as JavaScript, C/C++, and Java, use something called LOGIC to make simple decisions and perform repetitive actions in a program. In addition, these languages use OPERATIONS to perform mathematical calculations, comparisons, transfers of data, and other activities within the computer itself. In combination, logic and operations do all of the actual work inside of a program.

How logic operates, and what sort of operations JavaScript is capable of, are topics for discussion in later sections. All that you have to remember right now is that computers are really very stupid creatures; they can't do anything for themselves unless you tell them how to do it, which means that you have to spell EVERYTHING out for them. In addition, the operations and decisions computers can make are really very simple ones compared to the ones humans can make; sophisticated programs create an ILLUSION of complex behavior by layering many simple operations together and performing them at unbelievably high speeds. At the core of any program, however, are basic logic structures and basic operations which we will introduce in these modules.

Computer programming is mostly comprised of problem solving: taking large complex problems, breaking them down into the smallest possible pieces, and building programs out of these simple bits. Whenever students ask me how to tackle a problem, I always respond that they should WRITE DOWN the things that they are sure about, even the stupidest-seeming things. Once you have written down things you KNOW must happen or problems you have already solved, it makes it much clearer what remains to be done. In addition, you must start breaking down your problems into the simplest bits that you can think of, until you have the entire problem broken down into single tasks which the computer can perform. I will attempt to explain a little more about this process in the next module.

Summary

In JavaScript, user-initiated EVENTS trigger JavaScript FUNCTIONS. These functions contain LOGIC which makes decisions or performs repetitions, using OPERATIONS to calculate, to compare, or to solve various problems. Once all logic and operations are finished, the function accesses and/or manipulates RESOURCES to generate user FEEDBACK based on the decisions and operations it has made. Obviously, this is an artificially simplified statement (since logic, operations, resources, and functions would actually be a part of ALL of these processes), but it gives you a GENERAL idea of the process involved in the execution of a JavaScript program.

You, as a programmer, will design every aspect of the JavaScript code itself, as well as the USER INTERFACE. You will be taking complex problems, breaking them down into single tasks which the computer can execute, and reassembling these simple bits into larger-scale programs which can then perform the complete task.

3 Variables: An Introduction

A computer stores data, among other things. The term "data" sounds intimidating, but it really isn't. Data is simply information. On the computer, data can be numbers, or words, or pictures, or sounds, or movies, or anything.

When you program in any imperative programming language, you will need to instruct the computer when to store data, and when to transfer data, and when to delete data, etc. Data, however, can be much more than mere information; it can also be references to actual elements of the web page or the web browser itself. You'll see what I mean as we go along.

In imperative programming languages, such as JavaScript, data is stored in VARIABLES. A variable is really just a container for information, for data.

In JavaScript, ANYTHING can be stored in a variable. A number or a word could be stored in a variable. A picture or a sound file could be stored in a variable. Even a reference to an actual browser window or an entire HTML page could be stored in a variable in JavaScript, allowing you to open or close (or otherwise manipulate) that browser window (for instance), or even alter an individual HTML element in that browser window, via the appropriate JavaScript variable. After we have explored some more basic JavaScript programming concepts, I will show you how variables and actual browser elements work together. Until I have given you some more theoretical information about programming, however, none of my explanations of the practical applications would make any sense, so please bear with me.

Think of a variable as if it were a box. You take data and you put it into a box (a variable). You can also replace the data in one of these variable boxes with new data.

A variable MUST have a name; otherwise, you'll have no way to refer to that variable in a manner which the computer (or YOU) can understand. In JavaScript, names ARE case-sensitive (along with everything else), so SPELLING COUNTS! Variable names follow the standard naming conventions, and should usually begin with a lower-case letter.

From http://www.w3schools.com/js/js_variables.asp

- Variable names must begin with a letter
- Variable names can also begin with $ and _ (but we will not use it)
- Variable names are case sensitive (y and Y are different variables)

In the following example, I will create a variable called "george". I am going to fill george with a number, the number 1.

Here is a conceptual picture of george:

Variable George

As you can see, george, the variable, is a box with a name, containing the number 1. In JavaScript, you will use variables to store all data, whether it be numbers, or words, or pictures, or whatever.

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). Again, these technical words sound intimidating, but they're not.

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.

Computers are stupid. If you don't tell them that something exists, they can't figure it out for themselves; that's why you have to declare a variable.

In JavaScript, you would declare "george" (from the above example) like this:

```
var george;
```

To declare a variable in JavaScript, you must first state the JavaScript keyword, "var" (which tells the computer that you want to create something of type "var" or variable), followed by a space, followed by the name that you want to call that variable (in this case, the name is george); the command is finished off with a semi-colon to end the line of code.

Once you have declared a variable, you may refer and make changes to that variable simply by stating its name and performing OPERATIONS upon it. Most operations in JavaScript (including addition (+), subtraction (-), multiplication (*), and division (/), among others) are performed with a handful of symbols, many of which will be already familiar to you; I will talk more about these OPERATORS a little later.

In JavaScript, I could initialize the already-declared variable, george, using the following syntax:

```
george = 1;
```

In the above example, I have initialized the variable, george, setting it equal to the number 1; I have done this using the "gets" operator, which is represented by the equals (=) sign.

The "gets" operator (=) replaces the contents of whatever is on its left-hand side with whatever is on its right-hand side. In the above example, george "gets" 1; this means that the "gets" operator takes the number 1 from its right-hand side and sticks that value into george on its left-hand side, replacing

george's contents; since george has never contained any data, this process INITIALIZES george, setting him to a starting value of 1. Again, I have ended my command, my line of code, with the semi-colon end-of-line marker; every complete command in JavaScript must end its line with a semi-colon.

A COMMAND is an instruction or set of instructions for the computer, telling it to do something (or multiple somethings). Every time you come to the end of a command in JavaScript (which will end a line of code), you must mark the end of that line with a semi-colon, as in the above examples.

I could also declare and initialize the variable, george, in a single step using the following syntax:

```
var george = 1;
```

First, I declare george to exist by using the var keyword, then I use the "gets" operator to initialize george to contain the number 1. Although it is possible to combine these two processes (declaration and initialization) together into one command as in the above example, it is important for you to understand that these are two separate actions, and that there will be times when you will NEED to separate variable declaration and initialization for one reason or another.

A Note on Semi-Colons

JavaScript is derived from the programming language called "C". All C-based languages end lines of code with semi-colons; a semi-colon (;) is the end-of-line or end-of-command marker. You have already seen a similar use of the semi-colon in CSS, where a semi-colon marks the end of a property definition (i.e. `font-size:24px;`).

Note: Older code written in JavaScript does NOT always use a semi-colon to mark the end of a line of code. Theoretically, JavaScript, under many circumstances, may count carriage-return characters as a replacement for semi-colon characters; this does not always work properly, however. I cannot recommend the omission of semi-colon characters when coding commands in JavaScript, as it opens you up to unexplained errors due (I assume) to bugs in the JavaScript interpreters.

4 Variables: Data Types

As mentioned earlier, variables can contain many kinds of data or information. There are several official TYPES of data which JavaScript can understand. In brief, I am going to outline a few of the most important of these data types, since I will be referring to them when we start coding a little later.

In JavaScript, there are three essential "primitive" data types: numbers, strings, and booleans.

Here are some numbers:

10
-147
.0035
1.5
-520.0003

As far as JavaScript is concerned, numbers may be positive or negative, and may have decimal places or be without decimal places.

As I mentioned earlier, however, JavaScript is a C-based language; the C-based languages include C++, Java, and JavaScript, among others. In C-based languages, numbers are divided into two separate varieties: integers (int) and floats (float).

Integers (int) are whole numbers with NO decimal places. Here are some integers:

1
-100
247
-68000
0

Floats (float) are numbers WITH decimal places. Here are some floats:

.0003
146.5
-2.575
10000.1

Again, when you are programming in JavaScript, you will not usually differentiate between integers and floats. However, 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.

STRINGS are collections of alphanumeric characters. Anything you can type on a computer keyboard may potentially be part of a string. Strings, in JavaScript, are always marked with quote marks. Here are some strings:

"Hi"
"Wow, you have 2.5 children!"
"Michael"
"Hey, dude, I like your green hair..."

As you can see, strings are ALWAYS enclosed in double-quote marks, and may contain a variety of characters, whether text, numbers, spaces, or punctuation.

In HTML, you have already dealt with strings frequently. Every time you set an attribute for a tag, you are actually setting that attribute equal to some string value (i.e. src="capitalA.gif" or width="54").

Even though strings may have numbers contained within them, there is a distinct difference between string-based numbers and actual numbers. For instance, these two values are NOT the same:

1
"1"

The first example is the actual number, 1. The second example is the CHARACTER "1", which is NOT a number; rather, it is merely a symbol that you type on the computer keyboard. You can NOT perform mathematical computations on a STRING "1", whereas you CAN perform mathematical computations on a NUMBER 1.

Having said this, JavaScript has on-the-fly conversion capabilities which automatically convert string numerical characters to numbers, as well as numbers to strings, under appropriate conditions. Although this sort of automatic conversion is a tremendous convenience, it does NOT keep variables from being EITHER numbers OR strings at any given time; you will frequently be called upon to differentiate between these two data types in your code. I myself have been tripped up more than once by these on-the-fly conversions while programming in JavaScript, so I just wanted to make sure that I mentioned this feature.

The third "primitive" data type in JavaScript is BOOLEAN. Booleans sound very technical, but they are quite simple. Invented by George Boole in a previous century, boolean numbers, in JavaScript and other traditional programming languages, have just two states: 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. In JavaScript, booleans will always be either false or true (not off/on or 0/1).

Booleans are everywhere in computer-based or electronics-based equipment. Have you ever noticed the | and 0 characters printed on the on/off switches for your computer, stereo, or other pieces of electronics equipment? Those characters represent on and off for the switch, with | or 1 being ON, and 0 being OFF. That's a BOOLEAN! Switches themselves are firmly embedded in the history of the first computers.

Once upon a time, computers were just gigantic, room-sized banks of switches, and, later, switches and lightbulbs, which turned on and off mechanically; when a lightbulb or switch was on, that was a 1, whereas when a lightbulb or switch was off, that counted as a 0. Using these 0's and 1's, you could create BINARY code (binary means numbers in base 2, which is all 0's and 1's). Just like the secret codes you may have played with as a child, you could say that the number 0 represents the letter "a", and that the number 1 represents the letter "b", and that the number 10 (10 in binary is 2 in our regular counting system) represents the letter "c", and that the number 11 represents the letter "d", etc. With big enough binary numbers, you could extend your binary "secret code" to represent not only letters, but also colors, or sound waves, or picture information, or whatever you like. Once you have decided which binary numbers represent which characters or sounds or pictures, you could store this binary information in your computer. With the "key" to your binary "secret code", you could then take this binary information out of your computer, manipulate it in some fashion, convert it back into a semblance of its original form, and display, play, or printout this media.

Using switches and lightbulbs, of course, computers could only store and manipulate a VERY limited amount of binary information (it took roomfuls of switches just to get enough numbers together to add up your checkbook, let alone represent millions of colors!); those early computers also weren't very fast. Now, with silicon microchips and advanced magnetic memory, computers can store and manipulate

mind-boggling quantities of information at magical speeds. All of this data, however, is still encoded, stored, and manipulated in the computer using on/off, binary, BOOLEAN states.

Although you will probably never have to get deep enough into the computer to be forced to look at actual binary conversion of data (thank goodness!), JavaScript, naturally enough, is still heavily reliant on booleans and boolean states. You will frequently be testing conditions in code to see whether something is true or false, a boolean state. Has the user clicked on a particular button? That information would be either true (the user HAS clicked on the button) or false (the user has NOT clicked on the button), a boolean state. Has the user visited this page three times? If they HAVE visited the page three times, the condition would be TRUE, whereas if they have visited the page FEWER than three times or MORE than three times, the condition would be FALSE, another boolean state. In some kind of game code, you might have a boolean variable which remains false until the user has collected all of the "magic wands" in the game (for instance), whereupon the boolean variable would be changed to true, allowing new levels of the game to open up. Boolean variables are everywhere in JavaScript.

Besides the three "primitive" data types, there is also the OBJECT data type in JavaScript. Objects include a whole host of structures and entities which are beyond the scope of this introductory lecture. In the next section, however, we will discuss the basic features of generic objects, as you will need to know a little bit about objects in order to use JavaScript effectively (since almost everything in JavaScript is built using objects).

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.

In JavaScript, the 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.

You will find that there are many times when you will need to check for the null or undefined variable data types, especially when handling user feedback or input. For instance, if you program JavaScript to bring up a prompt window (which allows a user to type in some information and click either the OK or Cancel buttons), you will need to test for the undefined or null variable if a user has not given any input or has pressed the "Cancel" button on the prompt dialog box; this is just one place where you might need to test for this data type.

The null variable and the undefined variable may be treated interchangeably in JavaScript, but can be detected individually in the most modern browsers, if needed.

5 Typed and Untyped Languages and Simple Variable Operations

In JavaScript, variables may contain ANY type of data. When you declare a variable, you only need to state the keyword var and the name of the variable; this creates a generic variable. Once you have a variable declared, you can stick a number or a string or a boolean or an object into the variable, and change the data type of the variable at will.

Example:

```
var george = 1;
george = "Hi";
```

In the above example, I have set george equal to the number 1. In the second line of code, I have replaced the contents of george with a string, "Hi". I have changed george's data type from a number to a string. This kind of action 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.

Most programming languages, like C++ and Java, are "typed" languages. In a typed language, a given variable may only ever be ONE data type. For instance, in C/C++, you would create an "int" variable type, and that variable could only contain an integer, or you would create a "string" variable type, and that variable could only ever contain a string.

Example (C/C++):

```
int george = 1;
string fred = "Hi";
float ethel = 1.052;
```

In the above example in C/C++, I have declared and initialized three different variables. The variable george is an integer (int) and may only ever contain an integer. The variable fred is a string, and may only ever contain a string. The variable ethel is a float, and may only ever contain a number with decimal places. In typed languages like C/C++, variable data types remain fixed for the duration of a program.

As I mentioned earlier, even though JavaScript is an untyped language, and even though you can change a variable's data type on-the-fly in JavaScript, variables can only be a SINGLE data type at any given time, whether that type be a number, a string, a boolean, an object, or null/undefined.

Here are some more JavaScript examples of variables being declared, initialized, and otherwise operated upon:

```
var kate = "hi";
var louie = 10.25;
var josie = louie;
var sam = 200;
var answer = louie + josie + sam;
kate = answer;
```

In order of the examples: the variable kate now contains a string value, "hi". The variable louie now contains a number. The variable josie has been set equal to louie, which means that a copy of louie's contents have been placed inside of josie; both josie and louie, then, are equal to 10.25. The variable sam

now contains a number as well. The variable called answer has been set to the sum of louie and josie and sam, which would be 10.25 + 10.25 + 200, or the number 220.5. Note: answer is just a random name, picked because it makes sense in English; I could have named it anything. The variable kate (which has already been declared), then, has had its string content of "hi" replaced with the numerical content of the variable answer, which changes kate's data type from a string to a number.

The following example is WRONG:

```
louie + josie + sam = answer;
```

Even though the above example looks like math operations that you might perform on paper, the above syntax is backwards. The variable answer needs to contain the sum of louie + josie + sam, which is only possible if the variable answer is on the LEFT-HAND side of the "gets" (=) operator (since the "gets" operator always replaces the contents of the left-hand variable with whatever is on the right-hand side). Left-to-right orientation, then, is important in JavaScript, as well as case-sensitivity!

Example (CORRECT):

```
var answer = louie + josie + sam;
```

The following example is ALSO WRONG:

```
var answer = louie + JOSIE + Sam;
```

Earlier, I initialized the variables josie and sam using all lower-case letters in their names. Since JavaScript is case-sensitive, JOSIE and Sam represent different, non-existent, undefined variables. The variable answer in the above example, then, would end up containing non-valid information, producing a bug and a JavaScript error in your code.

Again, I apologize for the abstractness of these examples, but you must understand the basics of JavaScript variable syntax before you can do anything at all with the language. I hope that you can see, though, that handling these variables and performing simple mathematical operations upon them is fairly straightforward as long as you understand the basic logic behind their syntax.

6 Alert Method of the Window Object

Alert Method of the Window Object

The Window Object

When you open a JavaScript-enabled web browser, such as Firefox or Internet Explorer, certain built-in objects in JavaScript are declared and initialized automatically. One of these built-in objects is an instance of the Window object called window (with a lower-case "w").

Again, you do NOT need to declare the window instance of the Window object; this is done for you automatically when a web browser window is opened. The window instance refers to the current browser window, and has many properties and methods associated with it. A complete list of properties and methods of the Window object are printed in *"JavaScript: The Definitive Guide"*, if you are interested. For the moment, I am only concerned with the alert() method of the Window object.

The alert() method of the Window object causes an alert dialog box to open. Here is an example of the alert() method in action:

Alert Me

The alert() method REQUIRES one argument, a string, which will become the text displayed in the alert dialog box.

Technically, the alert() method is invoked with the following complete syntax:

```
window.alert("Hi");
```

Note that using single quotes in this case is also acceptable:

```
window.alert('Hi');
```

Because everything on an HTML page is displayed within the web browser window, however, it is NOT necessary to state the window instance name directly; window is automatically implied as the parent object for everything on an XHTML page. It is perfectly safe to state the alert() method by itself, without using dot syntax to explicitly state the parent/child relationship between it and the window instance. The following syntax for calling the alert() method is perfectly safe and correct:

```
alert("Hi");
```

You can break up the string within an alerted method with the newline character \n. This character essentially causes a line break within the string. For example:

```
alert("Hi there\nYou are a brilliant web designer");
```

Here it is in action:

Alert Me

Note: \n must always be enclosed within quotes.

You can also include a variable in an alert statement using a + to concatenate it with a string. For

example: Assume a variable x takes on the value 12 prior to the alert. Then you could alert the following:

```
alert("The number of months in a year is " + x);
```

Displayed

Note that the text displayed by alert dialog boxes can only be plain text, not XHTML formatted text.

You can also use the javascript absolute URL as the value for the HREF attribute in order to trigger JavaScript commands, such as an alert.

Example:

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

In the above example, I have stated the SCHEME for the absolute URL (javascript), followed by the colon character, followed by the "scheme_specific_part", the actual JavaScript command desired. 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.

Here is an example of the alert() method placed into an onClick event handler:

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

Although in the previous example, the string value passed as the argument to the alert() method was enclosed in DOUBLE-quotes, you can see that I have marked the string value in the above example using SINGLE-quote marks. 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.

The following examples would be WRONG:

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

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

7 The SCRIPT Tag

It must be evident to you now that event handlers (such as onClick) are inadequate for holding any substantive quantity of code. Most JavaScript code in a JavaScript-enhanced page is placed within a SCRIPT tag. 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. There are many different languages that can be coded within a SCRIPT tag; there are even several different versions of JavaScript itself. For this class, all that you are concerned with is the main version of JavaScript.

Tag: **SCRIPT**
Attribute: **TYPE**
Value: **text/javascript**
Description: The TYPE attribute of the SCRIPT tag sets the programming language type to be coded within the tag.

Example (abbreviated):

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

Example (in context):

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">

<title>Sample JavaScript Page </title>

<script type="text/javascript">
// Here's a couple of lines of JavaScript code:
var george = 1;
alert("Howdy");
</script>
</head>
<body> <
p>Some content...</p>
</body>
</html>
```

In the above example, I have placed my SCRIPT tag in the HEAD of my HTML document. Between the opening and closing SCRIPT tags, I have programmed my JavaScript code, creating a variable called george containing a numerical value of 1, and calling the alert() method which is receiving one argument,

the string "Howdy".

Notice how I am able to use DOUBLE-QUOTE marks again to mark my string values. This is possible because I am NOT inside an attribute.

There is an older, deprecated attribute of the SCRIPT tag: LANGUAGE. You will still see many JavaScript-enhanced web pages using the LANGUAGE attribute of the SCRIPT tag rather than the TYPE attribute.

Example:

```
<script language="javascript">

<!--

// Here's a couple of lines of JavaScript code:

var george = 1;

alert("Howdy");

//-->

</script>
```

It is no longer necessary to use the LANGUAGE attribute.

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. For the most part, they may be omitted. Some of my sample pages in the coding exercises may include these statements.

As a page loads, the SCRIPT tag is executed, declaring and initializing variables, and executing methods.

Example Code (abbreviated):

```
<script type="text/javascript">

var george = 1;

alert("Howdy");

alert(george);

</script>
```

Example Code (in context):

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">

<title>Sample JavaScript Page</title>
```

```
<script type="text/javascript">

var george = 1; alert("Howdy");

alert(george);

</script>

</head>

<body>

<p>Some content...</p>

</body>

</html>
```

In the above example, I first declare and initialize a variable, george. I then call the alert method, passing the string value "Howdy" as an argument to the alert method. Last, I call the alert method again, passing the contents of the variable, george, to the alert method. Note: even though the contents of george are a NUMBER, not a STRING, JavaScript automatically converts the number INTO a string for use by the alert method. Cool, huh?

Here is the above example displayed.

Notice how the alert dialog boxes pop up as the page loads in the above example. As I said, the code inside the SCRIPT tag is executed as the page loads.

Of course, you're not usually going to want to have EVERYTHING in your script tag execute all at once; that would be complete chaos. To reserve some actions, and to call those actions at will rather than just as the page loads, requires the use of a FUNCTION. We'll talk about functions later in the course.

Final Notes

A page may, in fact, have several SCRIPT tags on it, both in the HEAD and in the BODY of the document. Because SCRIPT tags are executed as a page loads, you may have a SCRIPT tag in the BODY of a page which lays on-the-fly HTML (built from JavaScript code) into the BODY of the HTML page as the page is parsing.

JavaScript code may ALSO be coded into an external text file, called a ".js" file, which is very much like an external ".css" document, only filled with JavaScript code rather than CSS code. A ".js" file, or even several ".js" files, would be attached to an HTML page using the SCRIPT tag, and many HTML pages could share the same ".js" files. For example:

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

A way to enhance script performance is to cache the entire script, by including it in a .js file. The technique causes the browser to load the script in question only once, and recall it from cache should the page be reloaded or revisited.

Do not put the script tags in your external script files!

8 Comments

Single line comments are marked at the beginning of the line with "//", while multiple line comments are marked "/* comment... */". JavaScript, and all of the C-based languages, use the SAME commenting style! Unlike CSS, however, single line comment syntax IS supported in JavaScript, so you may use it with impunity here.

Note: It is no longer necessary need to place HTML comments inside my opening and closing SCRIPT tags, to hide the JavaScript code from non-JavaScript-enabled web browsers.

Example:

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

For all homework assignments in this course, I suggest inserting comments.

This will help you in remembering what you have done in creating the script and thus should be helpful in debugging your script. It will also help me in understanding your code.

Before the script tag you should describe, in 1 or 2 sentences, what is the purpose of the script. Within the script itself, you may wish to use comments that begin with // for a one line comment, or /*...comments...*/ for multiple line comments.

For Example:

<head>

/* The purpose of the following script is to welcome the user through an alert */

```
<script type="text/javascript">
```

window.alert ("Welcome to my CNIT 133 List Homework Page") // alert to user

</script>

</head>

9 Operators

There are operators, and there are operands.

Example:

```
1 + 1
```

The plus (+) sign is an OPERATOR, because it performs some action. The 1's are OPERANDS, because they are the things being acted, or operated, upon.

You already know several JavaScript operators:

+ (plus)
- (minus)
* (multiply)
/ (divide)

Here are some more operators, called COMPARISON operators, which you are probably also familiar with:

< (less than)
> (greater than)
<= (less than or equal to)
>= (greater than or equal to)

You have already encountered the "gets" (=) operator, also called the "assignment" operator. As you know, this operator is NOT used for comparisons; rather, it takes whatever is on its right side, and places that information into whatever is on its left side.

Example:

```
var george = 1;
```

To make comparisons for equality, you need to use the EQUALITY operator, which is represented by two equals signs together:

== (is equal to)

If you want to test whether something is NOT equal, you would use the INEQUALITY operator, which is represented by an exclamation point and an equals sign together:

!= (is not equal to)

Example:

```
if (theSky == "gray")
alert("Put on your overcoat");
```

If the sky is gray, then alert "Put on your overcoat".

Example:

```
if (theWater != "warm")
alert("Turn on the hot water");
```

If the water is not warm, then alert "Turn on the hot water".

Both of the above JavaScript examples use something called BRANCH logic; we'll talk more about JavaScript logic a little later in this module. I would also like to point out that I have created variable and function names which reflect PURPOSE, and, when used, sound very much like an English sentence; you should strive for this sort of clarity when writing JavaScript code.

Here are some more operators:

Logical AND:

&& (and)

Logical OR:

|| (or)

Example:

```
if (theSky == "blue" && theTemperature >= 65)
alert("Take off your coat and take a walk outside");
```

If the sky is blue AND the temperature is greater than or equal to 65 degrees Fahrenheit, then alert "Take off your coat and take a walk outside". The condition above will only be met if BOTH the sky is blue AND the temperature is greater than or equal to 65; otherwise, nothing will happen.

Example:

```
if (theSky == "gray" || theRain == "falling")
alert("Stay home with a book);
```

If the sky is gray OR the rain is falling, then alert "Stay home with a book". The condition above will be met if EITHER the sky is gray OR the rain is falling, which allows you more potential opportunities to stay home with a good book.

JavaScript also provides something called the INCREMENT and DECREMENT operators, which either add or subtract 1 from something.

++ (increment)
-- (decrement)

Example:

```
var myCounter = 0;
myCounter++;
// myCounter is now equal to 1.
myCounter--;
// myCounter is now equal to 0.
myCounter--;
// myCounter is now equal to -1.
```

Without the increment and decrement operators, we would be forced to say:

```
var myCounter = 0;
myCounter = myCounter + 1;
// myCounter is now equal to 1.
myCounter = myCounter - 1;
// myCounter is now equal to 0.
myCounter = myCounter - 1;
// myCounter is now equal to -1.
```

The increment and decrement counters provide a shorthand for adding 1 to a variable or subtracting 1 from a variable. This is extremely useful when counting mechanical repetitions using LOOP logic, another form of JavaScript logic which we will introduce in the next module.

The last operator we will discuss is called the "assignment with addition" operator, or the "add by value" operator.

+= (add by value)

This is another shorthand operator, like the increment/decrement operators. If we say "a += b", it is the same as saying "a = a + b". With the "by value" or "assignment with operation" operators, you get two actions for the price of one! The first operation is performed between the two operands, then the result of that operation is put into the first operand, leaving the second (right-hand) operand unchanged.

Example:

```
var fred = 1;
var ethel = 2;
fred += ethel;
```

In the above example, the variable fred starts out equal to 1, and the variable ethel starts out equal to 2. After the "add by value" operator works on the two variables, fred is now equal to 3, while ethel is still equal to 2. Again, this is because:

```
fred += ethel;
```

is the same as saying:

```
fred = fred + ethel;
```

This operator is VERY important in JavaScript because it is used to CONCATENATE, or join, strings together. You'll need to do this in order to create XHTML text on-the-fly using JavaScript.

Example:

```
var myName = "Michael";
var myText = "<p>Hi there, ";
myText += myName;
myText += "! How nice to see you!</p>";
```

In the above example, I am using the "add by value" operator to concatenate my XHTML string together, adding in the user name stored in the variable myName, to create one whole paragraph of XHTML code. If this were a real piece of JavaScript code, I would probably have gotten the user's name from a prompt or out of a form text INPUT field, and stored that information, placing it into XHTML pages (using

SCRIPT tags and the document.write() method) at key points to personalize the site for the user.

There are a lot of other operators, but these are most (though not all) of the essential ones. For more information about operators, see *"JavaScript: The Definitive Guide"*, Chapter 5.

Summary of Operators

+ (plus)
- (minus)
* (multiply)
/ (divide)

= (gets, assignment)
== (equality, is equal to)
!= (inequality, is not equal to)

< (less than)
> (greater than)
<= (less than or equal to)
>= (greater than or equal to)

&& (and, logical AND)
|| (or, logical OR)

++ (increment, adds 1 to operand)
-- (decrement, subtracts 1 from operand)

+= (assignment with addition, add by value)

10 Write Method of the Document Object

You use the write method of the document object to write content to a webpage. This includes strings and HTML tags, in which both must be enclosed in quotes. For example:

```
document.write( "The sky is blue." );
document.write( "The sky is <i>blue.</i>" );
document.write( "The <b>sky</b> is blue.<br>" );
document.write( "The sky is blue." );
document.write( "<h3>The sky is blue.</h3>" );
```

Displayed:
The sky is blue.The sky is *blue.*The **sky** is blue.
The sky is blue.

The sky is blue.

Variables can also be included using concatenation. For example, suppose the variable x has the value of 5:

```
var x = 5;
document.write( "The number " + x + " is the number you chose" );
```

Displayed:
The number 5 is the number you chose

Another write statement that can be useful is document.writeln. Writeln is short for "write line". For example:

```
document.writeln( "The sky is blue." );
document.writeln( "The water is clear." );
```

Displayed :
The sky is blue. The water is clear.

document.writeln is the same as document.write, except that essentially a space is written on the page right after the argument's content.

You can include inline styles within document.write statements. For example:

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

Notice that the escape character \" is needed because we need quotes around the style's values and we need quotes around the entire argument of the document.write statement. An alternative is to replace \" with a single quote. For example:

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

Displayed:

XYZ

You can use document.write to write XHTML tables. For example:

```
document.write( "<table border = '1'><tr> " + "<th>Number</th> <th>Absolute Value</th></tr>" );
document.write( "<tr align=\"right\"><td>" + 1 + "</td>" + "<td>" + Math.abs(1) + "</td></tr>" );
document.write( "<tr align=\"right\"><td>" + 2 + "</td>" + "<td>" + Math.abs(2) + "</td></tr>" );
document.write( "<tr align=\"right\"><td>" + 0 + "</td>" + "<td>" + Math.abs(0) + "</td></tr>" );
document.write( "<tr align=\"right\"><td>" + -4 + "</td>" + "<td>" + Math.abs(-4) + "</td></tr>" );
document.write( "</table>" );
```

Displayed: Sample JS Write Method 04

11 Prompts

You can obtain user input by using the prompt method of the window object. The problem with using prompts is that IE has basically disabled prompts for so-called security reasons. For this reason, I suggest avoiding prompts. If you do use IE and get a message to allow scripts and accept it, then you should refresh the page and the prompt will be normally be displayed.

For example:

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

This will open a popup dialog box that allows the user to enter a string. The dialog box will contain the message "Enter a number" (without the quotes), a blank input text area (represented by ""), and an ok and cancel button. The user positions the mouse into the input text area and types in characters and then clicks the ok button to return the string to the script. This will assign the user input to the variable 'number'. Note that the text displayed by prompts is plain text, not XHTML formatted text.

Displayed: Sample JS Prompt 01

Similar to the alert method, you can can omit window from window.prompt. For example:

```
number = prompt( "Enter a number", "" );
```

You can also initalize the blank input text area. For example:

```
number = window.prompt( "Enter a number", "0" );
```

Displayed: Sample JS Prompt 02

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

If you need to use the variable 'number' in an arithmetic statement you should convert it from a string to an arithmetic integer, using the parseInt function. For example:

```
var number, numb;

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

numb = parseInt( number );
```

If number is to include a decimal point and decimal positions then you would use the parseFloat function. For example:

```
var number, numb;

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

numb = parseFloat( number );
```

Now let's take a simple example. Suppose you wanted to write a script that prompts the user for 2 numbers, adds them together, and displays the results using a document.write statement.

```
<script type="text/javascript">
```

```
var number1, number2, sum;

number1 = prompt( "Enter the first number", "" );

number2 = prompt( "Enter the second number", "" );

sum = number1 + number2;

document.write("<h2>The sum of the numbers is " + sum + "</h2>");

</script>
```

Displayed: Sample JS Prompt 03

Notice that the displayed sum is INCORRECT. Instead of adding the numbers, it concatenated them because the prompts returned strings. The reason why the displayed sum is incorrect is because we didn't include the parseInt statements.

Corrected Example Displayed: Sample JS Prompt 04

```
<script type="text/javascript">

var number1, number2, numb1, numb2, sum;

number1 = prompt( "Enter the first number", "" );

number2 = prompt( "Enter the second number", "" );

numb1 = parseInt( number1 );

numb2 = parseInt( number2 );

sum = numb1 + numb2;

document.write("<h2>The sum of the numbers is " + sum + "</h2>");

</script>
```

Notice that if the user enters a non-numeric character to either of the above prompts then the result will display 'The sum of the numbers is NaN', meaning Not a Number. This will also occur if the user clicks the cancel button when responding to the prompt.

You can also combine the parseInt and prompt statements. For example:

```
<script type="text/javascript">

var number1, number2, sum;

number1 = parseInt( prompt( "Enter the first number", "" ) );

number2 = parseInt( prompt( "Enter the second number", "" ) );

sum = number1 + number2;

document.write("<h2>The sum of the numbers is " + sum + "</h2>");

</script>
```

Displayed: Sample JS Prompt 05

Notice that if the user enters a floating point number to either prompt then the sum will be incorrect

because parseFloat statements would be needed instead of parseInt statements.

Final thoughts about prompts:

JavaScript does not provide for a simple method of input that is analogous to writing a line of text with document.write. So prompts can provide a way to allow user input. The problem with using prompts is that you can only obtain one value per prompt. Another problem is that IE have basically disabled prompts, and because of this, you should avoid using them. Finally and most importantly, most users find prompts to be relatively annoying. A better way to obtain input from the user is to use form input text boxes. This method will be discussed later in this book.

12 Introduction to Functions

Since the understanding of functions is very important to programming languages, including JavaScript, functions will be introduced right from the start in this course. Some of the following can be found on W3Schools.

To keep the browser from executing a script when the page loads, you can put your script into a function.

A function contains code that will be executed by an event or by a call to the function.

You may call a function from anywhere within a page (or even from other pages if the function is embedded in an external .js file).

Functions can be defined both in the <head> and in the <body> section of a document. However, to assure that a function is read/loaded by the browser before it is called, it could be wise to put functions in the <head> section.

Syntax

function *functionname*(*var1,var2,...,varX*)
{
some code
}

The parameters var1, var2, etc. are variables or values passed into the function. The { and the } defines the start and end of the function.

Note: A function with no parameters must include the parentheses () after the function name.

function *functionname*()
{
some code
}

Note: Do not forget about the importance of capitals in JavaScript! The word *function* must be written in lowercase letters, otherwise a JavaScript error occurs! Also note that you must call a function with the exact same capitals as in the function name.

Example 1

```
<head>
<title>Example 1</title>
<script type="text/javascript">

function displaymessage()
{
```

```
alert("Hello World!");
}

</script>
</head>

<body>
<form>
<input type="button" value="Click me!" onclick="displaymessage()">
</form>
</body>
</html>
```

If the line: alert("Hello world!!") in the example above had not been put within a function, it would have been executed as soon as the page was loaded. Now, the script is not executed before a user hits the input button. The function displaymessage() will be executed if the input button is clicked.

Displayed:

Note that the following relevant code will be equivalent to the above:

```
<form>
<input type="button" value="Click me!" onclick="alert('Hello World!');">
</form>
```

Using a function is much better for organization and maintenance techniques than just listing the JavaScript statements as a string after the event handler. This is especially true when there are several statements to be executed.

Example 2

```
<head>
<title>Example 2</title>
<script type="text/javascript">

function writeMessages()
{
document.write("Hi there<br>");
document.write("Have a nice day<br>");
document.write("Are you having fun yet?");
document.close();
alert("bye");
}

</script>
</head>

<body>
```

```
<form>
<input type="button" value="Click me!" onclick="writeMessages();">
</form>
</body>
</html>
```

Displayed:

Note that the following relevant code will be equivalent to the above, but it is harder to follow and maintain:

```
<form>
<input type="button" value="Click me!" onclick="document.write('Hi there');
document.write('Have a nice day');
document.write('Are you having fun yet?'); document.close(); alert('bye'); ">
</form>
```

Note that Firefox will sometimes show a spinning wheel after a series of document.write statements, and thus you may need append these statements with a document.close() statement.

13 Form Input and Output

Form Input

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. Instead of invoking multiple prompts, you can obtain the input from the form, using dot syntax. Notice the use of a table for best input text alignment and CSS for right-justifying the input numbers.

For example:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Form example 1</title>
<style type="text/css">
input {text-align: right;}
body {background-color: #f5f5dc;}
.body { width: 550px; margin: 0px auto;}
</style>
<script type="text/javascript">
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>
<section class="body">
<form name="myform">
<table>
<tr>
<td>First Number:</td>
<td><input type="text" name="num1" size="10"></td>
</tr>
<tr>
<td>Second Number:</td>
<td><input type="text" name="num2" size="10"></td>
</tr>
</table>
<br>
<input type="button" onclick="process()" value="SUBMIT">
<input type="reset" value="RESET">
</form>
</section>
</body>
```

Displayed.

Explanations:

BODY SECTION:

The form statement is given a name, in this case '**myform**', so that the script can access the values of the form's elements. The input text statements also have distinct names so that their values can be accessed by the script. The input button functions to invoke the script's function, named 'process', via the onclick event handler.

THE SCRIPT:

The script contains the user-declared function **process()**, whose content is required to be enclosed within braces. After declaring the variables to be used in the function, we retrieve the value of the input text boxes and assign them to variables. Namely:

```
number1 = document.myform.num1.value;
number2 = document.myform.num2.value;
```

This statement means to assign the value of the field whose name is 'num1', in the form whose name is 'myform', to the variable 'number1'.

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;

After the string values are converted to integers via parseInt statements, n1 + n2 is calculated and assigned to the variable 'sum'. Finally an alert is used to display the desired result.

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">
```

Notice that an 'id' attribute replaces the 'name' atribute. In the script's function, we assign the variable 'number1' to the value of the input text field whose name is 'num1'.

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

This is the same as: number1 = document.myform.num1.value;

So given that our form contains:

```
<input type="text" id="num1" size="10">
<input type="text" id="num2" size="10">
```

Our entire script may look like this:

```
<script type="text/javascript">
```

```
function process()
{
var number1, number2, n1, n2, sum;
number1 = document.getElementById("num1").value;
number2 = document.getElementById("num2").value;
n1 = parseInt(number1);
n2 = parseInt(number2);
sum = n1 + n2; alert("The sum of the numbers is " + sum);}
</script>
```

Note that when using the getElementById method as in the above example, you should not include the name of the form. The following would cause a syntax error:

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

Form Output

One way to output the results from a script is to use document.write statements within the body tags. The problem with this method is that it usually leads to writing to a new page. A better approach is to output the results, using either form input text boxes or a form textarea.

For example:

```
<!DOCTYPE html>
<html lang="en">

<head>
<meta charset="utf-8">

<title>Form example 2</title>

<style type="text/css">
input {text-align: right;}
body {background-color: #f5f5dc;}
.body { width: 550px; margin: 0px auto;}
</style>

<script type="text/javascript">

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;
document.myform.result.value = sum;
}
</script>
</head>

<body>
```

```
<section class="body">

<form name="myform">
<table>
<tr>
<td>First Number:</td>
<td><input type="text" name="num1" size="10"></td>
</tr>

<tr>
<td>Second Number:</td>
<td><input type="text" name="num2" size="10"></td>
</tr>

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

</table>
<br><br>

<input type="button" onclick="process()" value="SUBMIT">
<input type="reset" value="RESET">
</form>
</section>
</body>
```

Displayed.

Notice that:
document.myform.result.value = sum;
actually outputs the value of 'sum' into the form text box whose name is 'result'. Alternatively, you may choose to outptut the results into a form textarea. For example:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Form example 3</title>
<style type="text/css">
input {text-align: right;}
body {background-color: #f5f5dc;}
.body { width: 550px; margin: 0px auto;}
</style>
<script type="text/javascript">
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;
document.myform.result.value = ("The inputted numbers were " + n1 + ", " + n2 + "\nThe sum of the integers was " + sum); }
</script>
</head>
<body>
<section class="body">
<form name="myform">
<table>
<tr>
<td>First Number:</td>
<td><input type="text" name="num1" size="10"></td>
</tr>
<tr>
<td>Second Number:</td>
<td><input type="text" name="num2" size="10"></td>
</tr>
</table>
<br> Results:
<textarea rows="4" cols="40" name="result">
</textarea>
<br><br>
<input type="button" onclick="process()" value="SUBMIT">
<input type="reset" value="RESET">
</form>
</section>
</body>
```

Displayed.

The advantage of using a textarea is that you can display multiple lines of output. For example, using the last example, suppose I change:

document.myform.result.value = ("The sum of the numbers is " + sum);
to the following:

```
document.myform.result.value = ("The inputted numbers were " + n1 + ", " + n2 + "\nThe sum of the integers was " + sum);
```

Displayed.

Notice the use of \n to cause a line break in the outputted string. Also notice that when you break up a long string over two or more lines that the concatenation symbol + must end all lines, except the last. Notice that the following can also be used:

```
document.getElementById("result").value =
("The inputted numbers were " + n1 +
", " + n2 + "\nThe sum of the integers was " + sum);
```

You can also add CSS style rules to form elements for an interesting visual effect. For example:

```
<style type="text/css">
textarea {font-family: verdana; font-size: 22px;}
</style>
```

Displayed.

Focusing the cursor on a form field

The easiest way to focus the cursor on a form field is to use the HTML5 attribute of autofocus

For ex:

```
<form>
<input type="text" name="name" size="20" autofocus>
.
.
.
</form>
```

Another way to do it is by using the JavaScript method of getElementById and the event handler of onload.

```
<body onload="document.getElementById('name').focus();">

<form>
<input type="text" id="name" size="20">
.
.
</form>
```

14 Input Alignment

1. When you use form input text boxes, I suggest using a table for better alignment of the descriptions and the input text boxes. For example:

```
<form>
.
.
<table>
<tr>
<td>First Number:</td>
<td><input type="text" id="num1" size="6"></td>
</tr>
<tr>
<td>Second Number:</td>
<td><input type="text" id="num2" size="6"></td>
</tr>
</table>
.
.

</form>
```

2. When you want to display numbers in form input text boxes (especially dollars and cents), it is best to align the numbers to the right so that it is easier for the user to verify any totals. You can do this with CSS, for example:

```
<style type="text/css">

input {text-align: right;}

</style>
```

or

```
<td><input type="text" name="Value" style="text-align: right;"></td>
```

Here is a similar example to the first example in the previous chapter. The difference is that better alignment is used.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Form example 1</title>
<script type="text/javascript">
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>
<body bgcolor="#f5f5dc">
<div align="center">
<form name="myform">
<table>
<tr>
<td>First Number:</td>
<td><input type="text" name="num1" size="10"
style="text-align: right;"></td>
</tr>
<tr>
<td>Second Number:</td>
<td><input type="text" name="num2" size="10"
style="text-align: right;"></td>
</tr>
</table>
<br>
<input type="button" onclick="process()" value="SUBMIT">
<input type="reset" value="RESET">
</form>
</body>
</html>
```

Displayed.

15 InnerHTML

The innerHTML Property

The easiest way to get or modify the content of an element is by using the innerHTML property.

innerHTML is not a part of the W3C DOM specification. However, it is supported by all major browsers.

The innerHTML property is useful for returning or replacing the content of HTML elements.

Whereas using document.write statements will display output on a new page, you can use innerHTML and getElementById to display output on the same page.

What you do is to set a variable to a string that will contain HTML. This string is then created using concatenation. For example:

```
<!DOCTYPE html>
<html lang="en">

<head>
<meta charset="utf-8">
<title>innerHTML</title>

<style type="text/css">
input {text-align: right;}
body {background-color: #f5f5dc;}
.body { width: 360px; margin: 0px auto;}
</style>

<script type="text/javascript">

function process() {

// Set the htmlOutput variable to a string that contains an HTML table

var htmlOutput = "";

htmlOutput += "<table border = '1'>";
htmlOutput += "<tr><th>State</th><th>Capital</th></tr>";
htmlOutput += "<tr><td>California</td><td>Sacramento</tr>";
htmlOutput += "<tr><td>New York</td><td>Albany</tr>";
htmlOutput += "<tr><td>Texas</td><td>Austin</tr>";
htmlOutput += "</table>";

// Set the contents of the <div> block to the contents of htmlOutput

document.getElementById("theTable").innerHTML = htmlOutput;

}
```

```
</script>

<style type="text/css">
body {background-color: #f5f5dc;"}
</style>

</head>

<body>
<section class="body">
<h3>Create and Output Table on this Page</h3>

<form>
<input type="button" value="Create and Output Table" onclick="process();">
</form>

<br><br>

<!-- id of theTable is where the output will be written by the document.getElementById statement -->

<div id="theTable"></div>

</section>
</body>
</html>
```

Displayed

16 If Statement

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

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!).

For example, let's say the CONDITION that I am testing is the color of the sky. Depending on what color the sky is, I am going to take off my hat, or put on my overcoat, or take out my umbrella.

IF the sky is blue, then I am going to take off my hat. IF the sky is gray, then I am putting on my overcoat. IF the sky is dark gray OR black, then I am taking out my umbrella.

I am about to go out of the house, and have come to a decision point. Depending upon the color of the sky, I am going to take different actions. My path, at this point, will BRANCH, depending upon the condition that I am testing (in this case, the color of the sky). This is an example of branching logic: I come to a decision point, and my path may BRANCH in any number of different ways, depending upon the result of the condition test that I am making.

In JavaScript, this sort of logic is expressed using an IF STATEMENT.

Example:

```
if (theSky == "blue")
document.write( "The sky is blue" );
```

First, I state the keyword, if, which makes this an IF STATEMENT, followed by a space. Then I put opening and closing parentheses characters as a container for the CONDITION that I am testing. If whatever is being tested inside the parentheses is true, then I will perform whatever action follows the parentheses; if whatever is being tested inside the parentheses is false, then I will go on to the next line of JavaScript code, ignoring the statement following the parentheses.

I may perform MULTIPLE lines of code if my condition is true by using curly-brace characters to define space, much as I do with the function declaration.

Example:

```
if (theSky == "blue")
{
document.write( "The sky is blue<br>" );
document.write( "Open the curtains<br>" );
document.write( "Wave to the neighbors" );
}
```

Now consider the following example:

```
if (theSky == "blue")
```

```
{
document.write( "The sky is blue<br>" );
document.write( "Open the curtains<br>" );
document.write( "Wave to the neighbors<br>" );
}
document.write( "Go to work" );
```

Note that Go to work will appear on the web page whether or not the IF condition is true or false. If the If condition is true, then Go to work will appear after 3 statements are displayed, and if the If condition is false, then Go to work will be the only statement displayed on the web page.

17 If...else if Statement

Textbook References

Flanagan book: some pages in Ch.5 and 6

Gosselin book: p. 149-165

McDuffie book: p. 162-164, 174-175

Deitel book: Ch. 7

If I want to branch in more than two directions, I can use ELSE IF statements after my initial IF statement.

Example:

```
if (theSky == "blue")
{
     document.write( "Put on my sunglasses" );
}
 else if (theSky == "gray")
{
     document.write( "Put on my coat" );
}
else if (theSky == "black")
{
     document.write( "Put on my coat" );
     document.write( "Get out my umbrella" );
}
```

Note: I may include a final ELSE statement at the end of my branch if I want a catch-all statement for everything that doesn't meet the previous conditions.

With sequential if/else-if statements in a set, as soon as one of the conditions is met in the branch, that part of the code is executed and the branch is ended. You can't get TWO separate portions of the branch to execute if TWO of the conditions are true; to do that, you need to use TWO separate IF statements rather than an if/else-if branch like the one above.

Examples:

Ex1:

```
if (x >= 6 )
    if (y < 9 )
       alert("A")
```

```
    else
       alert("B")
else
    alert("C")
```

Notice that the 2nd else belongs with the 1st if.
Since the 1st if statement is false when x=5 and y=8, C will be alerted.

Ex2:

```
if (x >= 6 )
    if (y < 9 )
        alert("A")
    else
        alert("B")
```

When x=5 and y=8, nothing will get alerted.

When x=6 and y=8, both if statements are true and A will be alerted.

When x=6 and y=11, the 1st if statement is true but the 2nd if statement is false, and so B will be alerted.

Note that a very common mistake is using only one equal sign in conditions that test for equality.

Ex:

x=0;

```
if (x = 0)
    alert("x equals " + x);
else
    alert("x not equal to " + x);
```

Corrected:

```
if (x == 0)
    alert("x equals " + x);
else
    alert("x not equal to " + x);
```

18 Debugging Syntax Errors

You are going to run frequently into JavaScript code that doesn't work. In fact, MOST code fails to operate correctly the first time you test it, for one reason or another.

Debugging JavaScript syntax errors is much harder, in general, than debugging HTML errors. JavaScript is VERY strict with regards to syntax.

There are two types of syntax errors, viz., load-time errors and run-time errors.

Load-time errors:

An error that violates the grammatical rules of the language when the script is loaded. These errors are the major mistakes that prevent the script from functioning before it has a chance to start. It is during the loading process that JavaScript spots any serious errors that will cause your script to fail right off the bat. The script cannot be run until the page has been successfully loaded.

The error may be that the code has an extra parenthesis, or is missing a quote, brace, semi-colon, etc. Or you may declare a variable that is a JavaScript reserve word, such as class. Or you may enter parseint, instead of parseInt. JavaScript is case-sensitive.

Run-time errors:

A run-time error occurs when the JavaScript interpreter encounters a problem during the execution of a program. This usually happens when the interpreter reads code it can't handle.

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

For example:

```
<script type="text/javascript">
document.write(greeting);
</script>
```

In IE, you would get:

```
Line: 8
Char: 1
Error: 'greeting' is undefined
```

You get a similar error message in Firefox.

If you then changed the script to:

```
<script type="text/javascript">
var greeting = "Hi there";
document.write(greeeting);
```

</script>

You would get an error message that 'greeeting' is undefined.

Another type of run-time error occurs when you misapply one of JavaScript's objects. For example, document="test"; results in an error ("document cannot be set by assignment.") because you cannot assign a value directly to the document object.

Firefox:

I have found that debugging is fairly easy in 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.

Suppose you have a script containing the following:

var x = window.prompt("enter a value" "");

By clicking Tools/Error Console/Errors, you will see something similar to:

Error: Missing) after argument list
Line: 11
var x = window.prompt("enter a value" "");

Most Firefox versions also show an arrow pointing to approximately where the error occurred. In this case, the arrow will point to the first " after value.

I suggest getting Firefox's add-on called Web Developer.

You may wish to also download Firefox's add-on debugger called Firebug.

Internet Explorer:

If you are testing your code in older versions of Internet Explorer, and the 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.
Suppose you have a script containing the following:

var x = window.prompt("enter a value" "");

When it is run you will see the yellow error icon indicating a syntax error. After clicking it, the error message box will say something like:

Line: 11
Char: 39
Error: Expected ')'

Code: 0

This is not all that enlightening. Besides the trouble of finding line 11 and character 39, the actual error may be on a different line. After inspecting the code, you should eventually see that the error is that "enter a value" should be followed by a comma.

Starting with IE8, there is a JavaScript debugger as part of its development tools. To use it, you must ensure that it's enabled. Again you need to have "Disable script debugging" as UNCHECKED, under Tools/Internet Options/Advanced. The script debugger is accessed by clicking Tools/Developer Tools, which opens the debugger window. Now click the Script tab at the top left and the Start Debugging button.

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. The console displays the error message, the URL of the error, and the line number for the error.

There is a freeware editor called Crimson Editor for Windows. It has line numbering that can be turned on, and line numbers can also be included when printing hard copy. It's a pretty nice editor with syntax coloring for various language files such as html and css.

It's available for download at http://www.crimsoneditor.com/

A free and good editor for both HTML and for locating line numbers is HTML-Kit. It has plugins for JavaScript coding.

You can also locate line numbers in Notepad++.

Here is a list of some of the most common errors that students make:

Spelling: JavaScript is CASE-SENSITIVE. Check your spellings!

Incorrectly-placed Carriage Returns: Placing carriage returns in the middle of parentheses for methods or functions may not work.

Spaces: Names may have NO SPACES in them. Some arguments may also not have spaces in them, depending on the method in question. Don't forget to follow the naming conventions that I outlined in the earlier modules!

Missing Quote Marks: One quote mark missing, and your JavaScript program will fail. The same holds true for missing parentheses or curly-braces, etc. I get caught by this myself surprisingly often.

Here is a site for checking for javascript syntax errors online. I tried it and it looks useful.

http://jshint.com/

19 Debugging Logic Errors

If running your script does not induce a syntax error then either it worked as you intended it or it contains a logic error, i.e., the script still may not work properly while running: it may be mis-calculating a value, or perhaps an **if** condition is using the logically wrong (but syntactically valid) operator. These are logic errors.

To debug a logic error can be very time consuming as you would need to inspect every line in your script. This is known as tracing.

Some common logic errors:

1. Using the 'less than' symbol < when you meant to use the 'greater than' symbol > or vice versa

2. In an 'if' statement, using the 'and' symbol && when you meant to use the 'or' symbol || or vice-versa. This happens often when using negation.

Some advice if you can't find your logic error within a reasonable amount of time:

1. insert window.alert statements into your script to let you know where you are and what's happening. You add a call to the alert() method into a sequence of lines of code. Test your code, and see if the alert dialog box you inserted shows up or not. Remove that alert(), and put another one someplace else. Keep testing with the alert() method in various locations until you pinpoint the precise point in the code where everything breaks down.

2. Comment out unessential code and try to run the block you're having trouble with. Once you've solved any problems in that section, allow a few more lines in by removing the comments on those lines. Now debug those lines of code. Continue the process until you've got a clean working script.

3. Print out your code since it's easier to spot errors on printed paper than reading it on the screen.

4. Take a break. Persistence doesn't mean you have to solve every problem in one session. The longer you leave the problem, the more likely you'll be able to see it with fresh eyes and spot your error immediately. Perhaps, sleep on it.

The biggest secret to debugging code is NOT to let yourself get frustrated. Mistakes and bugs in code are NORMAL, and you should ALWAYS expect them. If code works right the first time, it's usually a fluke and a cause for celebration; I dance around the house, patting myself on the back, on the rare occasions when that happens!

Methodical coding procedures are also of paramount importance. Once you've gotten a grasp of what good coding form is all about, you will need to develop certain habits, producing code in a very regular and regimented fashion.

Here are some good sites for debugging tips on load-time errors, run-time errors, and logic errors:

http://www.cross-browser.com/talk/debug_tips.php
http://www.javaworld.com/javaworld/jw-07-1996/jw-07-javascript.html

http://javascript.about.com/od/reference/a/error.htm

20 Introduction to jQuery

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 is a web designer's secret weapon in the battle of JavaScript programming. With jQuery, you can accomplish tasks in a single line of code that would otherwise take hundreds of lines of programming and many hours of browser testing to achieve with your own JavaScript code.

I believe that the best tutorial for an introduction to jQuery is found at W3Schools.

Tutorial: http://www.w3schools.com/jquery/default.asp

For your convenience, here is a partial summary from this tutorial.

jQuery is a JavaScript Library.
jQuery greatly simplifies JavaScript programming.
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

URL for jQuery:

http://code.jquery.com/jquery-1.9.1.js

You may wish to download the above script, upload it to hills, and refer to your hill's server filename, e.g., jquery.js, instead of using the above URL.

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

Alternatively you can use the following script statement to access jQuery:

<script type="text/javascript" src="http://code.jquery.com/jquery-1.9.1.js"></script>

<script type="text/javascript" src="http://code.jquery.com/jquery-1.10.1.js"></script>

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)

http://www.w3schools.com/jquery/jquery_ref_selectors.asp

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).

Coding example: Display a p tag and clicking it will make it disappear

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8" />
<title>jQuery Ex 1</title>
<script type="text/javascript" src="http://code.jquery.com/jquery-1.9.1.js"></script>
<script type="text/javascript">
$(document).ready(function(){
$("p").click(function(){
$(this).hide();
});
});
</script>
</head>
<body>
<p>If you click on me, I will disappear.</p>

</body>
</html>
```

Displayed

jQuery Event Functions

The jQuery event handling methods are core functions in jQuery.

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

It is common to put jQuery code into event handler methods in the <head> section:

Coding example: Display p tags and clicking button will make them disappear

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8" />
<title>jQuery Ex 2</title>
<script type="text/javascript" src="http://code.jquery.com/jquery-1.9.1.js"></script>
<script type="text/javascript">
$(document).ready(function(){
$("button").click(function(){
$("p").hide();
});
});
</script>
</head>

<body>
<h2>This is a heading</h2>
<p>This is a paragraph.</p>
<p>This is another paragraph.</p>
<button type="button">Click me</button>
</body>

</html>
```

Displayed

jQuery Hide and Show

With jQuery, you can hide and show HTML elements with the hide() and show() methods:

Example

```
$("#hide").click(function(){
```

```
$("p").hide();
});
$("#show").click(function(){
$("p").show();
});
```

Coding example: Display p tags and clicking button will make them disappear slowly

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8" />
<title>jQuery Ex 3</title>
<script type="text/javascript" src="http://code.jquery.com/jquery-1.9.1.js"></script>
<script type="text/javascript">
$(document).ready(function(){
$("button").click(function(){
$("p").hide(1000);
});
});
</script>
</head>
<body>
<button type="button">Hide</button>
<p>This is a paragraph with little content.</p>
<p>This is another small paragraph.</p>
</body>
</html>
```

Displayed

Query Slide - slideDown, slideUp, slideToggle

The jQuery slide methods gradually change the height for selected elements.

Coding example: Display a panel of text and clicking it will then display another panel of text

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8" />
<title>jQuery Ex 4</title>
<script type="text/javascript" src="http://code.jquery.com/jquery-1.9.1.js"></script>
<script type="text/javascript">
$(document).ready(function(){
$(".flip").click(function(){
```

```
$(".panel").slideDown("slow");
});
});
</script>

<style type="text/css">
div.panel,p.flip
{
margin:0px;
padding:5px;
text-align:center;
background:#e5eecc;
border:solid 1px #c3c3c3;
}
div.panel
{
height:120px;
display:none;
}
</style>
</head>

<body>

<div class="panel">
<p>Because time is valuable, we deliver quick and easy learning.</p>
<p>At W3Schools, you can study everything you need to learn, in an accessible and handy format.</p>
</div>

<p class="flip">Show Panel</p>

</body>
</html>
```

Displayed

Coding example: Clicking a button will fade a colored box and the button

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8" />
<title>jQuery Ex 5</title>
<script type="text/javascript" src="<script type="text/javascript" src="http://code.jquery.com/jquery-1.9.1.js"></script>
<script type="text/javascript">
$(document).ready(function(){
```

```
$("button").click(function(){
$("div").fadeTo("slow",0.25);
});
});
</script>
</head>

<body>
<div style="background:yellow;width:300px;height:300px">
<button type="button">Click to Fade</button>
</div>
</body>
</html>
```

Displayed

See another Tutorial about jQuery:

http://docs.jquery.com/Tutorials:How_jQuery_Works#jQuery:_The_Basics

Note: I have found that the button tag does not always work in jQuery unless type="button" is included.

Final note: You are required to use jQuery in some homework assignments in this course. Wherever applicable, feel free to use jQuery in any homework assignment in this course.

21 Using Events the jQuery Way

The following is an excerpt from **Chapter 5 of the book, JavaScipt and jQuery, the Missing Manual**.

Mouse Events

Ever since Steve Jobs introduced the Macintosh in 1984, the mouse has been a critical device for all personal computers. Folks use it to open applications, drag files into folders, select items from menus, and even to draw. Naturally, web browsers providemlots of ways of tracking how a visitor uses a mouse to interact with a web page:

• **click.** The click event fires after you click and release the mouse button. You'll commonly assign a click event to a link: For example, a link on a thumbnail image can display a larger version of that image when clicked. However, you're not limited to just links. You can also make any tag on a page respond to an event—even just clicking anywhere on the page.
Note: The click event can also be triggered on links via the keyboard. If you tab to a link, then press the Enter (Return) key, the click event fires.

• **dblclick.** When you press and release the mouse button twice, a double-click (dblclick) event fires. It's the same action you use to open a folder or file on your desktop. Double-clicking a web page isn't a usual web-surfer action, so if you use this event, you should make clear to visitors where they can double-click and what will happen after they do. Also note that a double-click event is the same thing as two click events, so don't assign click and double-click events to the same tag. Otherwise, the function for the click will run twice before the double-click function runs.

• **mousedown.** The mousedown event is the first half of a click—the moment when you click the button before releasing it. This event is handy for dragging elements around a page. You can let visitors drag items around your web page just like they drag icons around their desktop—by clicking on them (without releasing the button)and moving them, and then releasing the button to drop them.

• **mouseup.** The mouseup event is the second half of a click—the moment when you release the button. This event is handy for responding to the moment when you drop an item that has been dragged.

• **mouseover.** When you move your mouse over an element on a page, a mouseover event fires. You can assign an event handler to a navigation button using this event and have a submenu pop up when a visitor mouses over the button. (If you're used to the CSS :hover pseudo-class, then you know how this event works.)

• **mouseout.** Moving a mouse off an element triggers the mouseout event. You canmuse this event to signal when a visitor has moved her mouse off the page, or to hide a pop-up menu when the mouse travels outside the menu.

• **mousemove.** Logically enough, the mousemove event fires when the mouse moves—which means this event fires all of the time. You use this event to track the current position of the cursor on the screen. In addition, you can assign this event to a particular tag on the page—a , for example—and respond only to movements within that tag.

For Document/Window Events, Form Events, and Keyboard Events, see **Chapter 5 of JavaScript and jQuery, the Missing Manual**.

Using Events the jQuery Way

Traditionally, programming with events has been tricky. For a long time, InternetExplorer had a completely different way of handling events than other browsers, requiring two sets of code (one for IE and one for all other browsers) to get your code to work. Fortunately, IE9 now uses the same method for handling events as other browsers, so programming is a lot easier. However, there are still a lot of people using IE8 and earlier, so a good solution that makes programming with events easy and cross-browser compatible is needed. Fortunately, you have jQuery.

JavaScript libraries like jQuery solve a lot of the problems with JavaScript programming—including pesky browser incompatibilities.

In addition, libraries often simplify basic JavaScript-related tasks. jQuery makes assigning events and event helpers (the functions that deal with events) a breeze. jQuery programming involves (a) selecting a page element and then (b) doing something with that element.

In fact, since events are so integral to JavaScript programming, it's better to think of jQuery programming as a threestep process:

1. Select one or more elements.

The last chapter explained how jQuery lets you use CSS selectors to choose the parts of the page you want to manipulate. When assigning events, you want to select the elements that the visitor will interact with. For example, what will a visitor click—a link, a table cell, an image? If you're assigning a mouseover event, what page element does a visitor mouse over to make the action happen?

2. Assign an event.

In jQuery, most DOM events have an equivalent jQuery function. So to assign an event to an element, you just add a period, the event name, and a set of parentheses. So, for example, if you want to add a mouseover event to every link on a page, you can do this:

```
$('a').mouseover();
```

To add a click event to an element with an ID of menu, you'd write this:

```
$('#menu').click();
```

After adding the event, you still have some work to do. In order for something to happen when the event fires, you must provide a function for the event.

3. Pass a function to the event.

Finally, you need to define what happens when the event fires. To do so, you pass a function to the event. The function contains the commands that will run when the event fires: for example, making a hidden tag visible or highlighting a moused-over

element.

You can pass a previously defined function's name like this:
$('#start').click(startSlideShow);

When you assign a function to an event, you omit the () that you normally add to the end of a function's name to call it. In other words, the following won't work:

$('#start').click(startSlideShow())

However, the most common way to add a function to an event is to pass an anonymous function to the event. The basic structure of an anonymous function looks like this:

```
function() {
// your code here
}
```

Here is a jQuery example from **Chapter 5 of JavaScript and jQuery, the Missing Manual**. It involves double clicking to display an alert, clicking a button to change the button's text, a mouseover on a link to display some text, and clicking in a form textbox to change the color to red. View source code for a full explanation of the code.

jQuery UI

jQuery UI provides a comprehensive set of core interaction plugins, UI widgets and visual effects that use a jQuery-style, event-driven architecture and a focus on web standards, accessiblity, flexible styling, and user-friendly design. All plugins are tested for compatibility in IE 6.0+, Firefox 3+, Safari 3.1+, Opera 9.6+, and Google Chrome.

22 Web Tutorials and Examples - Basic Operations

Introduction to JavaScript

- http://www.webteacher.com/javascript/
- http://www.w3schools.com/js/js_intro.asp
- http://www.javascriptkit.com/javatutors/primer1.shtml
- http://javascriptkit.com/javatutors/learnjs.shtml

script tag

- http://www.w3schools.com/js/js_whereto.asp

document.write and document.writeln

- http://www.w3schools.com/jsref/met_doc_write.asp
- http://www.w3schools.com/jsref/met_doc_writeln.asp

variables

- http://www.w3schools.com/js/js_variables.asp

parseInt() and parseFloat()

- http://www.w3schools.com/jsref/jsref_parseInt.asp
- http://www.w3schools.com/jsref/jsref_parseFloat.asp

operators

- http://www.w3schools.com/js/js_operators.asp
- http://www.tutorialspoint.com/javascript/javascript_operators.htm

if statement

- http://www.w3schools.com/js/js_if_else.asp

alert and prompt

- http://javascriptkit.com/javatutors/alert1.shtml
- http://www.pageresource.com/jscript/jalert.htm
- http://www.pageresource.com/jscript/jprompt.htm

Math object

- http://www.w3schools.com/jsref/jsref_obj_math.asp

innerHTML Property and getElementbyId

- http://www.w3schools.com/htmldom/dom_methods.asp
- http://www.tizag.com/javascriptT/javascript-innerHTML.php
- http://msdn2.microsoft.com/en-us/library/ms536437.aspx
- http://www.mozilla.org/docs/dom/domref/dom_doc_ref48.html

23 Short Exercises - Basic Operations

1. Identify and correct the errors in each of the following statements:

a) if (c < 7);
window.alert("c is less than 7");

Ans: Error: There should not be a semicolon after the right parenthesis of the condition in the if statement.

Correction: Remove the semicolon after the right parenthesis.
[Note: The result of this error is that the output statement is executed whether or not the condition in the if statement is true. The semicolon after the right parenthesis is considered an empty statement, i.e., a statement that does nothing.]

b) if (c => 7)
window.alert("c is equal to or greater than 7");

Ans: Error: The relational operator => is incorrect.

Correction: Change => to >=.

2. Write a statement (or comment) to accomplish each of the following tasks:

a) State that a program will calculate the product of three integers.

Ans: // Calculate the product of three integers

b) Declare the variables x, y, z and result.

Ans: var x, y, z, result;

c) Declare the variables xVal, yVal and zVal.

Ans: var xVal, yVal, zVal;

d) Prompt the user to enter the first value, read the value from the user and store it in the variable xVal.

Ans: xVal = window.prompt("Enter first integer:", "0");

e) Prompt the user to enter the second value, read the value from the user and store it in the variable yVal.

Ans: yVal = window.prompt("Enter second integer:", "0");

f) Prompt the user to enter the third value, read the value from

the user and store it in the variable zVal.

Ans: zVal = window.prompt("Enter third integer:", "0");

g) Convert xVal to an integer, and store the result in the variable x.

Ans: x = parseInt(xVal);

h) Convert yVal to an integer, and store the result in the variable y.

Ans: y = parseInt(yVal);

i) Convert zVal to an integer, and store the result in the variable z.

Ans: z = parseInt(zVal);

j) Compute the product of the three integers contained in variables x, y and z, and assign the result to the variable result.

Ans: result = x * y * z;

k) Write a line of text containing the string "The product is " followed by the value of the variable result.

Ans:
document.writeln("<h1>The product is " + result + "</h1>");

3. Fill in the blanks in each of the following statements:

a) ____ are used to document a program and improve its readability.

Ans: Comments.

b) A dialog capable of receiving input from the user is displayed with method ____ of object ____.

Ans: prompt, window

c) A JavaScript statement that makes a decision is the ___ statement.

Ans: if.

d) Calculations are normally performed by ____ statements.

Ans: assignment.

e) A dialog capable of showing a message to the user is displayed
with method ____ of object ____.

Ans: alert, window

4. Write JavaScript statements that accomplish each of
the following tasks:

a) Display the message "Enter two numbers" using the window object.

Ans: window.prompt("Enter two numbers");

b) Assign the product of variables b and c to variable a.

Ans: a = b * c;

c) State that a program performs a sample payroll calculation.
[Hint: Use text that helps to document a program.]

Ans: // This program does a sample payroll calculation

5. State whether each of the following is true or false.
If false, explain why.

a) JavaScript operators are evaluated from left to right.

Ans: False. Some operators associate from right to left.

b) The following are all valid variable names:
_under_bar_, m928134, t5, j7, her_sales$, his_$account_total,
a, b$, c, z, z2.

Ans: True.

c) A valid JavaScript arithmetic expression with no parentheses
is evaluated from left to right.

Ans: False. All operators do not have the same precedence and
associativity. Some operations
are performed before others (e.g., multiplication before addition).

d) The following are all invalid variable names:
3g, 87, 67h2, h22, 2h.

Ans: False. h22 is a valid variable name because it begins with
a letter. Variable names in JavaScript can begin with a letter,

an underscore (_) or a dollar sign ($).

6. Fill in the blanks in each of the following statements:

a) What arithmetic operations have the same precedence as multiplication?

Ans: division (/) and remainder (%).

b) When parentheses are nested, which set of parentheses is evaluated first in an arithmetic expression?

Ans: The innermost set.

c) A location in the computer's memory that may contain different values at various times throughout the execution of a program is called a.

Ans: variable.

7. What displays in the message dialog when each of the given JavaScript statements is performed? Assume that x = 2 and y = 3.

a) window.alert("x = " + x);

Ans: x = 2

b) window.alert("The value of x + x is " + (x + x));

Ans: The value of x + x is 4

c) window.alert("x =");

Ans: x =

d) window.alert((x + y) + " = " + (y + x));
Ans: 5 = 5

8. Which of the following JavaScript statements contain variables whose values are destroyed (i.e., changed or replaced)?

a) p = i + j + k + 7;
b) window.alert("variables whose values are destroyed");
c) window.alert("a = 5");
d) stringVal = window.prompt("Enter string:");

Ans: a, d.

9. Given $y = ax^3 + 7$, which of the following are correct JavaScript statements for this equation?

a) y = a * x * x * x + 7;
b) y = a * x * x * (x + 7);
c) y = (a * x) * x * (x + 7);
d) y = (a * x) * x * x + 7;
e) y = a * (x * x * x) + 7;
f) y = a * x * (x * x + 7);

Ans: a, d, e.

10. State the order of evaluation of the operators in each of the following JavaScript statements, and show the value of x after each statement is performed.

a) x = 7 + 3 * 6 / 2 - 1;

Ans: * is first, / is second, + is third, and - is fourth. Value of x is 15.

b) x = 2 % 2 + 2 * 2 - 2 / 2;

Ans: % is first, * is second, / is third, + is fourth, - is fifth. Value of x is 3.

c) x = (3 * 9 * (3 + (9 * 3 / (3))));

Ans: Value of x is 324.

24 Coding Exercises - Basic Operations

Exercise 1a: document.write

Create a webpage that contains a script that outputs the italicized letters A, B, and C on the same line, with each letter separated by a comma and one space. Write the script using three document.write statements and a text color of orange.

Your output should look like this:

A, B, C

Click here for the solution and then view source.

Exercise 1b: document.write

Create a script using only document.write statements, that displays a table that has form text boxes with 3 preset employee names. Input their annual salaries. Calculate and display their salary with a $100 bonus. Display the total of all salaries and the total salaries with bonuses.

Disable the bonus fields and the total fields. Right-align the numeric fields.

To run the example, click here. Then view source for the solution. Note that the form and table are within the script.

Here is another version of the example without document.write statements.

Exercise 2a: prompt, if statement, and alert

Create a webpage that contains a script that uses prompt statements to prompt the user for four integers and uses an alert statement to display the prompted numbers, how many integers were less than 6, how many were greater or equal to 6, and the product of the 4 integers. Note: prompts do not automatically work for IE.

Click here for the solution and then view source.

Exercise 2b: Same as previous problem except using form text boxes as input and a form textarea as output

Create a webpage that contains a script that uses form text boxes to input four integers and uses a form textarea to display the inputted numbers, how many integers were less than 6, how many were greater or equal to 6, and the product of the 4 integers.

Click here for the solution and then view source.

Here is the same solution in which the textarea has no border or scrolling using CSS. Also I have used the autofocus attribute on the first input text box, right-justified the input text boxes, and have used onkeyup events for automatic tabbing of the input text boxes. Note that the first 3 fields requires 3 digits before the auto tabbing is induced. Here is a similar page that uses jQuery to hide the instructions of the

problem and the validation icons when a button is clicked.

Exercise 2c: Similar to the previous problem except use an array and the 'for' looping statement. Also define a 'Click to Fade Results' button on the form. Use jQuery to define a ready handler that defines a click event handler for the button, such that clicking the button fades the results in the textarea.

Click here for the solution and then view source. Note for the jQuery fade to work properly you may need <button type="button">

Exercise 3: form input, if statement, and alert

Create a webpage that contains a script that inputs two integers from form input fields and uses alert statements to display the inputted numbers and whether the integers are equal, or which integer is greater. Display the sum of the two integers in another form input field.

Click here for the solution and then view source.

Exercise 4a: document.write of a table

Create a webpage that contains a script that outputs the following table, which includes using the Math object method: Math.sqrt()

Number	Square Root
4	2
9	3
16	4
25	5

Click here for the solution and then view source. Here is a similar example using jquery to fade out the page after it displays.

Exercise 4b: Write same table (Part 2)

Use getElementById and innerHTML to create and write the table on the same page.

Click here for the solution and then view source.

Exercise 5: parseFloat and toFixed

Create a webpage that contains a script that will calculate how many hours it will take 2 people to finish a job working together, given that you know how many hours it will take each person to finish the job doing it alone. Assume that the people will work at the same rate together as they do alone. Use form text boxes for the hours to complete a job for each person alone. Use a form text box to display how many hours it will take them working together. Accept floating-point numbers for input and round the output to two decimal places. Verify that the input is numeric by using the isNAN function for testing a

non-numeric value. The isNaN function stands for is NOT a Number and returns true if the value is NaN.

Click here for the solution and then view source.

Exercise 6: modulo operator %

Create a webpage that contains a script that inputs two integers from form input fields and then determines and outputs to a form field whether the first integer is a multiple of the second.

Click here for the solution and then view source.

Exercise 7: using if....else if statement:

Use form text boxes for an employee's name and yearly gross income. Using if...else if statements, display a relevant comment in a form text box as described here:

Yearly gross income	Salary comment
0-19999	poor
20000-49999	lower middle
50000-99999	upper middle
100000 or over	wealthy

Check for valid input for yearly gross.

Note that I use jQuery to color the 2nd input text box when the user clicks in that box.

To run the example, click here