

Sponsored by:



E-COMMERCE
EXCHANGE

Introduction to

JavaScript

for non-programmers

TRAINING
T O L S
TRAININGTOOLS.COM

Increase your sales by 30% to 100%!

<http://www.ecx.com> ~ 800-815-6610



E-Commerce
Exchange

Best Reseller Prices.

HE'S GOT A WEBSITE. DO YOU?

E-Commerce Enabled Resellers, We Refer Design Work 30 Day Money Back Guarantee

The Best Support and Pricing in the Industry.

Pricing From \$19.95 per Month

- 100 MB of Web site storage
- Unlimited e-mail accounts for your Web site
- Full e-Commerce solutions, backed by Hewlett Packard
- Toll-free 24x7 technical support and Operations Center
- 99.9% Uptime Guarantee
- UNIX or NT Web servers
- Browser-based control of your Web site and e-mail
- Aggressive Reseller Programs

Call Today
800.552.6123 www.interland.net
 404.586.9999 / sales@interland.net

@ Speed. Reliability. Support.™
interland®
Web Hosting

Share these FREE Courses!

Why stuff your friend's mailbox with a copy of this when we can do it for you!
 Just e-mail them the link info – <http://www.trainingtools.com>

Make sure that you visit the site as well:

- MORE FREE COURSES
- Weekly Tool Tips
- Updated course versions
- New courses added regularly

So don't copy files or photocopy - Share!

End User License Agreement

Use of this package is governed by the following terms:

A. License

TrainingTools.com Inc, ("we", "us" or "our"), provides the Licensee ("you" or "your") with a set of digital files in electronic format (together called "the Package") and grants to you a license to use the Package in accordance with the terms of this Agreement.

B. Intellectual Property

Ownership of the copyright, trademark and all other rights, title and interest in the Package, as well as any copies, derivative works (if any are permitted) or merged portions made from the Package shall at all times remain with us or licensors to us. This Package is protected by local and international intellectual property laws, which apply but are not limited to our copyright and trademark rights, and by international treaty provisions.

C. Single-User License Restrictions

1. You may not make copies of the files provided in the Package
2. You may not translate and/or reproduce the files in digital or print format
3. You may not rent, lease, assign or transfer the Package or any portion thereof
4. You may not modify the courseware

YOUR QUESTION ? +



OUR ANSWER

= YOUR SOLUTION.



FOR ONLY \$10.00^{USD} YOU CAN HAVE 30 DAYS EMAIL SUPPORT FOR THIS COURSE

**CALL: 1-888-644-3444
1-416-675-1881**

XNU.COM **Helping YOU to build a better Web!**

Copyrights and Trademarks

No part of this document may be reproduced, stored in a retrieval system, or transmitted in any form or by any means – electronic, mechanical, recording, or otherwise – without the prior written consent of the publisher.

Netscape Navigator is a trademark of Netscape Communications Corp.

Windows 3.1, Windows 95, Windows NT, and Internet Explorer are trademarks of Microsoft Corporation.

All trademarks and brand names are acknowledged as belonging to their respective owners.

Published by

XtraNet

180 Attwell Dr., Suite 130 Toronto, Ontario, Canada M9W 6A9

Phone: 416-675-1881 Fax: 416-675-9217

E-mail: info@xnu.com

Copyright © 1999 by XtraNet Communications Inc.

All Rights Reserved

Printed in Canada

January 1999

First Edition

1 2 3 4 5 6 7 8

Table of Contents

Chapter 1 - Introduction to JavaScript programming.....	1
JavaScript versus JAVA.....	2
Interpreted programs vs. Compiled programs.....	2
Why Learn JavaScript.....	3
What you can use JavaScript for.....	3
About JavaScript.....	3
Review Questions.....	4
Summary.....	5
Chapter 2 - JavaScript Syntax.....	6
Inserting Client Side JavaScript into an HTML Page.....	7
Syntax and Conventions.....	8
Case-sensitivity.....	8
Semicolons.....	8
White Space.....	8
Strings and Quotation Marks.....	9
Brackets, Opening and Closing.....	10
Comments.....	11
Variable and Function Names.....	12
Reserved Words.....	13
Review Questions.....	14
Summary.....	15
Chapter 3 - Basic Programming Constructs.....	16
Declaring Your Variables.....	17
Types of Variables.....	17
Using Operators.....	18
JavaScript Operators.....	19
Control Structures (Loops and Branches).....	21
Branches.....	21
The if statement.....	21
The switch statement.....	22
Loops.....	23
The while loop.....	23
The for loop.....	23
Functions.....	24
Built-in functions.....	24
Programmer created functions.....	24
Review Questions.....	26
Summary.....	27
Chapter 4 - Objects, Events, and the Document Object Model.....	28
Object.....	29
The new operator.....	29
The Document Object Model (DOM).....	30
Arrays.....	33
Events.....	34
onClick.....	34
onSubmit.....	34
onMouseOver.....	35
onMouseOut.....	35
onFocus.....	36
onChange.....	36
onBlur.....	36
onLoad.....	37
onUnload.....	37
Review Questions.....	39
Summary.....	40
Chapter 5 - Alerts, Prompts, and Confirms.....	41

Table of Contents

Alerts, Prompts, and Confirms	42
window.alert()	42
window.prompt()	42
window.confirm()	43
Review Questions	47
Summary	48
Chapter 6 - Form Validation	49
Creating a FORM with an Active Cursor	50
Using Form Checking	51
Complete Script	56
Review Questions	58
Summary	59
Chapter 7 - Mouse Over Effects	60
Image Object	61
Mouse-over	62
Creating Flexible Functions	63
Pre-loading Images	66
eval()	67
Testing for completion and compatibility	69
Completed Script	70
Review Questions	71
Summary	72
Chapter 8 - Pop-up Windows	73
Pop-up Windows	74
window.open()	74
window.close()	74
Window Features Explained	75
Creating the window contents on the fly	78
Setting the document colors	80
Complete Script	81
Review Questions	82
Summary	83
Glossary	84
Answer Appendix	87
<u>TOOLBOX CAFE</u>	96

1

Introduction to JavaScript Programming

This section will provide you with the basics of what JavaScript is, and why you should use it.

Objectives

1. JavaScript versus JAVA
2. Interpreted programs versus Compiled programs
3. Why JavaScript
4. What you can use JavaScript for
5. About JavaScript

JavaScript versus JAVA

JAVA is a strongly typed, compiled programming language developed by Sun Microsystems. JavaScript, developed originally by Netscape, is a lightweight, interpreted programming language initially called LiveScript. The two languages are not related in any way. All programming languages share a certain amount of similarity.

Interpreted programs versus Compiled programs

Before we start discussing the differences between interpreted and compiled we have to define the term source code or as it is more commonly referred to, the code. The code is the plain text commands that the program is written in. All programming languages start out as source code, it is then either interpreted or compiled. The code that you will create in this course can be considered source code.

Interpreted programming languages tend to be simpler to program but slower to execute in general. Each time a program is run it has to be interpreted (interrogated) line by line, based on the flow of execution (you will see later branches and loops affect the flow of execution).

Compiled programming languages have a more complex syntax, and require more strict programming practices. With a compiled programming language you first write the source code, then you feed it to a compiler (a special computer program) which produces an executable binary program. On the Windows platforms the output of the compiler usually ends in the ".exe" file extension. The program that comes out of the compilation process tends to be platform (operating system) specific. The key benefit for the programmer is that no other programmer can look at the source code once it is compiled. The other key factor is that the language used to write the source code becomes irrelevant once it has been compiled.

JAVA is a compiled language that is platform independent, whereas JavaScript is an interpreted language. The browser provides the platform independence for JAVA through its JAVA Virtual Machine, and the interpreter for JavaScript. As a result, the browser you are writing your scripts for is important.

Why Learn JavaScript

JavaScript is the only scripting language currently supported by the popular web browsers. Netscape Navigator only supports JavaScript, whereas Microsoft Internet Explorer supports both JavaScript and VBScript. JavaScript can also be used on web servers for what's called server side scripting as well. The time you invest into learning the JavaScript language will provide you with what is now considered to be a core skill for web development.

What you can use JavaScript for

JavaScript can extend the usefulness of your web pages beyond what you can do with just HTML. In this course you will use it to ensure that a user is inputting data into your forms in the correct format, to create interesting buttons with mouse rollover effects, and to create pop-up windows. When combined with Cascading Style Sheets, you can create what are called Dynamic HTML pages. By learning JavaScript your needs and imagination will lead you to extend your HTML pages.

About JavaScript

JavaScript is an interpreted programming language that can be embedded into an HTML web page. Interpreted means that the entire web page is downloaded to the browser and the JavaScript code is executed when an event is triggered. When the code is executed it is interpreted one line at a time. There are a number of events that will trigger the execution of a JavaScript, like a click on a form button, or the completion of a web page loading.

Netscape originally created JavaScript; It has since been standardized by the European Computer Manufactures Association (ECMA). Today there are several versions of JavaScript (1.0, 1.1, 1.2,...) and the language is continually developing as both the Internet and the web evolve.

The different versions follow somewhat browser development, and the older browsers do not support newer versions of JavaScript.

Browser Version	Netscape Navigator	Microsoft Internet Explorer
2	JavaScript 1.0	Not Supported
3	JavaScript 1.1	JavaScript 1.0
4	JavaScript 1.2; not full ECMA-262 compliant	JavaScript 1.2; ECMA-262 compliant

Review Questions

1. (True or False) JavaScript is an interpreted language.
2. JAVA is a _____ programming language, and is platform _____.
3. (True or False) JAVA and JavaScript were created by the same company.
4. Microsoft Internet Explorer supports the following scripting languages.
 - a. JavaScript
 - b. JAVA
 - c. BASIC
 - d. VBScript
 - e. C++
 - f. Perl
5. (True or False) JavaScript is supported by a large number of browsers.

Summary

1. JavaScript is not JAVA
2. JavaScript is Interpreted, and JAVA is Compiled
3. Why you would use JavaScript
4. What you can use JavaScript for
5. About the JavaScript Language

2

JavaScript Syntax

In this chapter you will learn about the peculiarities of the JavaScript language. These are the details for writing a script that will help you avoid errors while you are creating your own scripts and learning the basics of the JavaScript programming language.

Objectives

1. Placing JavaScript in an HTML page
2. Case-sensitivity
3. Semicolons
4. Whitespace
5. Brackets
6. Comments
7. Variable and Function Names
8. Reserved Words

Inserting Client Side JavaScript into an HTML Page

JavaScript is added to an HTML page using the SCRIPT tag. The script tags should be placed inside the head tag of the document. If an older browser looks at a page containing script tags it will ignore them, as older browsers are written to ignore tags they can't interpret.

JavaScript code should also be placed inside an HTML Comment tag set.

E.g. `<!-- code -->`

When used with JavaScripts the ending comment tag will also start with two slashes // which is the JavaScript code for comment. This tells the JavaScript interpreter to ignore that statement.

This is a standard way for adding JavaScript to your HTML pages so that it works properly for browsers that are JavaScript enabled and those that do not support JavaScript.

```
<HTML>
<HEAD>
<TITLE>Web Page containing JavaScript</TITLE>

<SCRIPT LANGUAGE="JAVASCRIPT">
<!-- hide JavaScript code from browsers that are not JavaScript enabled
      .
      .      (JavaScript Statements goes here)
      .
//end hiding of JavaScript code -->
</SCRIPT>
</HEAD>

<BODY>
      (HTML document goes here)
</BODY>
</HTML>
```

We may also put in a single line of code attached to an event. Events will be explained later. The general syntax for this structure is:

```
<HTML_TAG Attribute="option" onEvent="JavaScript code statements go here">stuff in between the opening and closing tag</HTML_TAG>
```

Syntax and Conventions

Writing in any language follows rules and conventions. For example, the English language requires that each sentence must contain a subject and verb to be legitimate. You must also capitalize the first letter of a sentence, and end each sentence with punctuation such as a period or question mark. This is the syntax, or grammar of the English language. Each programming language has a similar set of rules commonly referred to as the syntax.

JavaScript has several rules and conventions for its syntax:

Case-sensitivity:

JavaScript is a case-sensitive language, meaning that the language will treat these words differently: example, Example, EXAMPLE

Semicolons:

All statements should end in a semicolon. The semicolon separates one statement from another.

I.e. `var x = 0; var y = 10;`

White Space:

JavaScript, like HTML, ignores spaces, tabs, and newlines that appear in statements. JavaScript does, however recognize spaces, tabs, and newlines that are part of a string. We will talk more about strings later in the course.

`var x=0;` is the same as `var x = 0;`

All of these examples will produce the same results. It is a good idea to put some spacing in your code to make it easier to read. It is not good programming practice to stretch statements over several lines.

You do need a space between a programming command and the thing it is working on. For example, you need a space between `var` and the variable name.

Strings and Quotes:

A string is a sequence of zero or more characters enclosed within single or double quotes ('single', "double").

The double quotation mark can be found within strings that start, and end with (are delimited by) single quotes ('He said, "JavaScript is an interesting language." ').

The single quotation mark can be used within a string delimited by double quotation marks. This syntax will be used quite often through out the book.

For example:

```
<INPUT TYPE="Button" VALUE="Click Me"
onclick="window.alert('You Clicked me');">
```

In the example above we have line of HTML code that uses double quotes to delimit the tag's attributes. So to create a popup window that displays the string "You Clicked me" we need to enclose the string within single quotes. This is done so that the entire JavaScript statement is interpreted and the HTML syntax also remains valid.

The Backslash (\) and Strings:

The backslash character named as such because it leans backwards, and it should not be confused with the forwardslash character (/) that leans forwards. The backslash has a special purpose in JavaScript strings. It will be followed by another character that represents something in a string that cannot be typed on the keyboard.

For example we want the word "You" to appear on one line and the word "Clicked" on a new line and "me" on a third line. The string would look like this:

```
'You\nClicked\nme'.
```

The \n represents a carriage return and a line feed. These are the two operations that take place when you use the return on a typewriter. The results would look like this:

```
You
Clicked
me
```

These backslash and letter combinations are commonly referred to as escape sequences. Some of the common sequences are:

<code>\b</code>	backspace
<code>\f</code>	form feed
<code>\n</code>	new line
<code>\r</code>	carriage return (no linefeed)
<code>\t</code>	tab
<code>\'</code>	single quote (apostrophe)
<code>\"</code>	double quote

The last two are important and can be used like this:

'You didn\'t get that done' or "You didn\'t get that done"

The `\` tells the interpreter that in this case it should print the single quote and not interpret it as a delimiter.

Opening and Closing Brackets:

All brackets you open must be closed! This includes (), [], and { }.
i.e. `window.open('ex1.htm','popup','scrollbars=yes');`

```
if ( x[0] == 10 ) {  
    x[0] = 0;  
    x[1] = 0;  
}
```

The curly brackets { } are used to contain multiple JavaScript statements. In the above example `x[0]=0;` and `x[1]=0;` are two different statements.

The square brackets [] are part of a special data structure called arrays. Arrays will be covered later in the course.

The curved brackets () are used to contain a function or a method's arguments. Functions and methods will be described shortly. Multiple arguments are separated by commas.

i.e. `('ex1.htm','popup','scrollbars=yes')`.

Comments:

You can create a comment using the double forward slashes, like this:

```
// this is a comment
```

Or for multiple line comments you can open the comment with a forward slash and an asterisk “/*”, and close it with an asterisk followed by a forward slash “*/” like this:

```
/* Comments are often used by programmers  
to leave notes about their program logic so that when  
they return to update it, or someone else needs to edit it,  
they can understand what the programmer was doing at the time.  
*/
```


Variable and Function Names

In the next chapter you will be introduced to variables and functions. As the programmer you get to choose and assign the names. The names of the variables and functions must follow these simple rules.

1. The first character must be a letter of the alphabet (lowercase or uppercase), an underscore (_) or a dollar sign (\$). The dollar sign is not recommended as it is not supported prior to JavaScript ver 1.1.
2. You CANNOT use a number as the first character of the name.
3. Names CANNOT contain spaces.
4. Names CANNOT match any of the reserved words.

The following are examples of valid names:

```
x  
add_two_num  
x13  
_whatever  
$money_string
```

We recommend that you use descriptive names for your variables and your functions and that you adopt a standard way of naming things. The two formats that are common are; using the underscore to replace spaces, or capitalizing the first letter of complete words after the first word in the name. For example:

```
add_two_num  
addTwoNumbers
```

JavaScript tends to use the latter for its naming conventions.

Reserved Words

There are a number of words that make up the components of the JavaScript language. These words cannot be used for variable or function names because the program interpreter would be unable to distinguish between a default JavaScript command and your variable or function name.

abstract	delete	innerWidth	Packages	status
alert	do	instanceof	pageXOffset	statusbar
arguments	document	int	pageYOffset	stop
Array	double	interface	parent	String
blur	else	isFinite	parseFloat	super
boolean	enum	isNaN	parseInt	switch
Boolean	escape	java	personalbar	synchronized
break	eval	length	print	this
byte	export	location	private	throw
callee	extends	locationbar	prompt	throws
caller	final	long	protected	toolbar
captureEvents	finally	Math	prototype	top
case	find	menubar	public	toString
catch	float	moveBy	RegExp	transient
char	focus	moveTo	releaseEvents	try
class	for	name	resizeBy	typeof
clearInterval	frames	NaN	resizeTo	unescape
clearTimeout	Function	native	return	unwatch
close	function	netscape	routeEvent	valueOf
closed	goto	new	scroll	var
confirm	history	null	scrollbars	void
const	home	Number	scrollBy	watch
constructor	if	Object	scrollTo	while
continue	implements	open	self	window
Date	import	opener	setInterval	with
debugger	in	outerHeight	setTimeout	FALSE
default	Infinity	outerWidth	short	TRUE
defaultStatus	innerHeight	package	static	

Review Questions

1. Which of the following are valid variable or function names:

- a. y
- b. 100percent
- c. a big number
- d. break
- e. subtractTwoNumbers
- f. First_Name

2. Place the brackets in the correct locations indicated by the underscore.

```
if _ z_0_ == 0 _ _  
    x = 2;  
_
```

3. Complete this sentence: All statements should end in a _____.

4. True or False. JavaScript is a case insensitive language.

5. True or False. It is a good idea to add comments to your program code.

6. Match the brackets in Column A with the use in Column B.

Column A	Answer	Column B
a. { }		used for array index values
b. []		used to contain a functions, arguments
c. ()		used to contain multiple JavaScript statements

Summary

1. JavaScript is placed within the <SCRIPT> tags
2. JavaScript is case-sensitive
3. All JavaScript statements end with a semicolon
4. JavaScript ignores whitespace
5. Which types of brackets to use where
6. How and why you should put comments in your program code
7. What names you can use for variables and function names
8. What words are reserved as part of the JavaScript language

3

Basic Programming Constructs

In this chapter you will learn the basics constructs of programming. These constructs are similar in a number of programming languages, and will be used in a number of our scripts in later chapters.

Objectives

1. Declaring Variables
2. Using Operators
3. Creating Control Structures (Branches and Loops)
4. Functions (Built-in and programmer-created)

Declaring Your Variables

A variable is a name assigned to a location in a computer's memory to store data. Before you can use a variable in a JavaScript program, you must declare its name. Variables are declared with the **var** keyword, like this:

```
var x;  
var y;  
var sum;
```

You can also declare multiple variables with the same **var** keyword.

```
var x, y, sum;
```

To take it one step further you can combine variable declaration with initialization.

```
var x = 1, y = 3, sum = 0;
```

If you don't specify an initial value for a variable when you declare it, the initial value is a special JavaScript undefined value.

Remember: JavaScript is case-sensitive so `x` and `X` are two different variable names.

Types of Variables

A big difference between JavaScript and other languages like JAVA and C is that JavaScript is *untyped*. This means that a JavaScript variable can hold a value of any data type, and its data type does not have to be set when declaring the variable. This allows you to change the data type of a variable during the execution of your program, for example:

```
var x = 10;  
x = "ten";
```

In this example the variable `x` is first assigned the integer value of 10, and then the string value of the word ten.

Using Operators

Operators are the things that act on variables. We have already seen an operator used in the previous example, where we were assigning values to our variables. The example used one of the most common operators, "=" or the assignment operator. Another operator would be the addition operator "+".

```
var x = 1, y = 3, sum = 0;  
sum = x + y;
```

This small chunk of JavaScript code will declare the variables x, y and sum and assign the number 1 to x, 3 to y and 0 to sum. The second line of the script will add x to y and assign it to sum. The value of the sum variable will be 4.

Other operators are used to compare things, i.e. "==" equality, ">" greater than. For example,

```
var x = 1, y = 3, sum = 0;  
if ( sum == 0 ) {  
    sum = x + y;  
}
```

This bit of code first checks to see if sum is equal to zero, and if so then it adds x and y together and assigns the result to sum. The "if" statement is an example of a control structure which we will examine shortly.

JavaScript Operators

Computational

These are probably the most common operators. They are used for common mathematical operations.

- Unary negation (-)
- Increment (++)
- Decrement (--)
- Multiplication (*)
- Division (/)
- Modulo arithmetic (%)
- Addition (+)
- Subtraction (-)

Logical

These operators are very commonly used in conditional statements like “if” and “switch” that you will be learning about shortly.

- Logical NOT (!)
- Less than (<)
- Greater than (>)
- Less than or equal to (<=)
- Greater than or equal to (>=)
- Equality (==)
- Inequality (!=)
- Logical AND (&&)
- Logical OR (||)
- Conditional (ternary) (?:)
- Comma (,)

Bitwise

You have probably heard that computers work with bits and bytes. These operators do work with bits or zeros and ones. These operators are very rarely used.

- Bitwise NOT (~)
- Bitwise Shift Left (<<)
- Bitwise Shift Right (>>)
- Unsigned Shift Right (>>>)
- Bitwise AND (&)
- Bitwise XOR (^)
- Bitwise OR (|)

Assignment

It is important to note that the single equal sign is used for assignment and not for testing equality. The compound assignment operators can combine a couple of programming steps to make your code tighter, and more efficient.

Assignment (=)

Compound assignment operators

Addition (+=)

Subtraction (-=)

Multiplication (*=)

Division (/=)

Modulo Arithmetic (%=)

Left Shift (<<=)

Right Shift (>>=)

Unsigned Right Shift (>>>=)

Bitwise And (&=)

Bitwise Or (|=)

Bitwise Xor (^=)

Control Structures (Loops and Branches)

Branches

if

The "if" statement is a fundamental control statement. It allows your program to perform a test, and act based on the results of that test. For example:

```
if ( (x == 1) && (y == 3) ) {  
    sum = y - x;  
}
```

In this statement the value of the x variable is compared to 1 to see if it is equal, and the value of the y variable is compared with 3 to see if it is equal. The use of the "&&", which is the "and" operator, adds an additional logical condition that says that the first comparison must be true **and** the second comparison must be true for the overall result of the test to be true. If the test results in an overall true condition then the statements that follow the if statement will be executed. If the test results are false nothing will occur.

An additional clause you can add to the "if" statement is the "else", an example:

```
if (sum == 0) {  
    sum = x + y;  
}  
else {  
    subtotal = sum;  
}
```

This statement is read as: if sum equals 0 then sum equals x plus y, or else subtotal equals sum.

switch

The switch statement is handy when a variable may take on a number of values and you want to test for some of those values. The use of "switch" is shorter and easier to read than a number of "if" statements.

```
switch(n) {
    case 1: //start here if n equals 1.
        // place code here
        break;

    case 2: //start here if n equals 2.
        // place code here
        break; // stop here

    .
    .
    .

    default; // if all other conditions fail do this
        // place code here
        break;
}
```

Loops

A loop is a programming structure that forces the statements contained within its delimiters to execute over and over again until a condition is met at which point the loop ends.

while

While a condition is true, execute one or more statements. “While loops” are especially useful when you do not know how many times you have to loop, but you know you should stop when you meet the condition.

```
var x = 1;
while ( x <= 10 ) { // loop until x is greater than 10

    until x is greater than 10

    x++; // add one to the value of x
}
```

for

“For loops” are useful when you know exactly how many times you want the loop to execute.

```
var x;
for ( x = 1; x <= 10; x++ ) { // loop while x is <= 10

    do something ten times

}
```

Functions

Functions are an important part of programming as they allow you to create chunks of code that perform a specific task. Functions in JavaScript are called subroutines or procedures in other programming languages. JavaScript has a number of built-in functions that are part of the language. JavaScript also gives you the ability to create your own functions. Your JavaScript programs may be made up of one function or several functions.

Built-in:

The built in functions are there to perform a number of actions that programmers expect to be part of a programming language. Some of the built in functions include: `parseFloat(string value)`, `parseInt(string value)`, `isNaN(value)`, etc.. We will use these functions later in the course.

Programmer Created:

Functions that you create start with the command “function” and are followed by the name for the function. For example:

```
function function_name( argument1, argument2, ... ) {  
    .  
    .  
    JavaScript statements go here  
    .  
    .  
} // end of function
```

Usually the name is an indicator of the action the function is going to provide such as “checkForm”. The name is followed by a set of parenthesis and inside the parenthesis there can be a number of arguments. Arguments are the variable names that will be used for values (data) being passed to the function.

All functions have the ability to pass back a value through the return statement. Another way of saying this is you can pass data into a function using its arguments, and get the results out with a returned value. Functions can only return **one** value. In the example below the sum of two numbers is returned by the `add_two_num()` function.

```
var globalVar = 1999;

function add_two_num( a, b) {
    var sum = a + b;
    return sum;
} // end of function add_two_num

function mainProgram() {
    var x = 5, y = 3, total = 0;
    total = add_two_num( x , y );
    alert(total); // display the value of total
}
```

In the above example:

1. The function `mainProgram` declares variables and assigns their initial values as follows - “x” equal to 5, “y” equal to 3 and “total” equal to 0
2. Then it calls the `add_two_num` function and passes in the values of x and y.
3. In the `add_two_num` function the values are added together and stored in a variable named `sum`.
4. The value of `sum` is returned back to the `mainProgram` function and stored in the variable named `total`.
5. The value of `total` is then displayed to user in an alert box.

Variables declared in a function are only available while within that function’s braces { }. These are commonly referred to as local variables. Another type of variable is a global variable. Global variables are available to all functions, and should be used with caution, as it is easy to assign the wrong value to a global variable. In the above example `globalVar` is a global variable because it is declared outside of all of the functions.

Review Questions

1. True or False. Variables in JavaScript are strongly typed.
2. True or False. You can declare a number of variables all at once.
3. The _____ keyword is used to declare variables.

4. Match the operator with its function.

Column A	Answer	Column B
a. =		assignment
b. ==		addition
c. +		equality

5. Create an if structure that will make sure the value of the p variable is greater than or equal to 99, and if so set the total equal to the value of p.
6. Create a function that will multiply two numbers together and return the result.

Summary

1. Variables are declared using the keyword var.
2. What operator to use
3. Creating control structures, branches and loops
4. The two types of functions, built-in and programmer-created

4

Objects, Events, and the Document Object Model

In this module you will be introduced to the concepts of Objects, the DOM, and events and how you can use them.

Objectives

1. Objects
2. The "new" operator
3. The Document Object Model
4. Arrays
5. Events
 - i. onClick
 - ii. onSubmit
 - iii. onMouseOver
 - iv. onMouseOut
 - v. onFocus
 - vi. onChange
 - vii. onBlur
 - viii. onLoad
 - ix. onUnload

Objects

An object is a collection of variables (parameters) and functions (methods). The syntax for using an object is: `object.parameter`, or `object.method()`. A string is an object in JavaScript and has several methods and parameters.

```
var StringVar = new String("This is a string of characters.");
```

```
var x = StringVar.length;  
StringVar = StringVar.toUpperCase();
```

If this code is executed `StringVar` is a new variable with the value of "This is a string of characters.". The `x` variable is set to the length of the `StringVar`, in this case 31. Length is a property of the string object. If you count the characters (letters, spaces, and punctuation) between the quotes you will see that it adds up to 31. The `toUpperCase()` method converts all of the alpha characters in the string to upper case i.e. "THIS IS A STRING OF CHARACTERS."

JavaScript has the following objects built into the language: `String`, `Math`, `Date` and `Array`. The string object has a number of methods for manipulating strings as demonstrated above with the `toUpperCase()` method. The math object is a collection of methods and properties for performing mathematical operations like: `min()`, `max()`, `sin()`, `cos()`, etc.. The date object is a collection of methods for working with dates and time. The array object allows programmers to create collections of data. Arrays will be discussed shortly.

The new operator

Objects and arrays **cannot** simply be typed into your JavaScript programs! They must be created. We use the "new" operator to create a new instance of an object or an array. To put that another way the new operator creates a copy of an existing object or an array structure and assigns the name you want to it.

The generic syntax is:

```
o = new Object();
```

Since objects are made up of methods (functions) and parameters (variables), the newly created object "o" in this case has all of the same methods and parameters of the original `Object`. The parameters will all be set to their default value. Arrays will be covered shortly.

The Document Object Model (DOM)

The browser provides us with a series of objects. The browser window that the page is displayed in is known as the window object. The HTML page displayed by your browser is known as the document object. The document object is probably the most commonly used object in client-side JavaScript.

The HTML elements that you add to a page also extend the object hierarchy. An example is the FORM element and the elements that reside inside the form. This means that you can reference these objects, as illustrated in the HTML page below:

`window.document.forms[0]`

Refers to the first form in the document. Forms are implemented as arrays in the DOM. If there is more than one form on page the numbers will start at zero and go up.

`window.document.Form1`

Refers to the form by name Form1

`window.document.Form1.FirstName.value`

Refers to the value typed into the textbox named FirstName by the client, in the form named Form1

```
<HTML>
```

```
<HEAD>
```

```
  <TITLE>Simple Form</TITLE>
```

```
</HEAD>
```

```
<BODY>
```

```
<FORM NAME="Form1">
```

```
Name: <INPUT TYPE="TEXT" NAME="FirstName"><BR>
```

```
<INPUT TYPE="Button" VALUE="Submit Info" >
```

```
</FORM>
```

```
<FORM NAME="Form2">
```

```
Name: <INPUT TYPE="TEXT" NAME="LastName"><BR>
```

```
<INPUT TYPE="Button" VALUE="Submit Info" >
```

```
</FORM>
```

```
</BODY>
```

```
</HTML>
```

Objects located in the current document, in the current window can drop the reference to those two objects. For example:

`forms[0]`

refers to the first form within this document

`Form1`

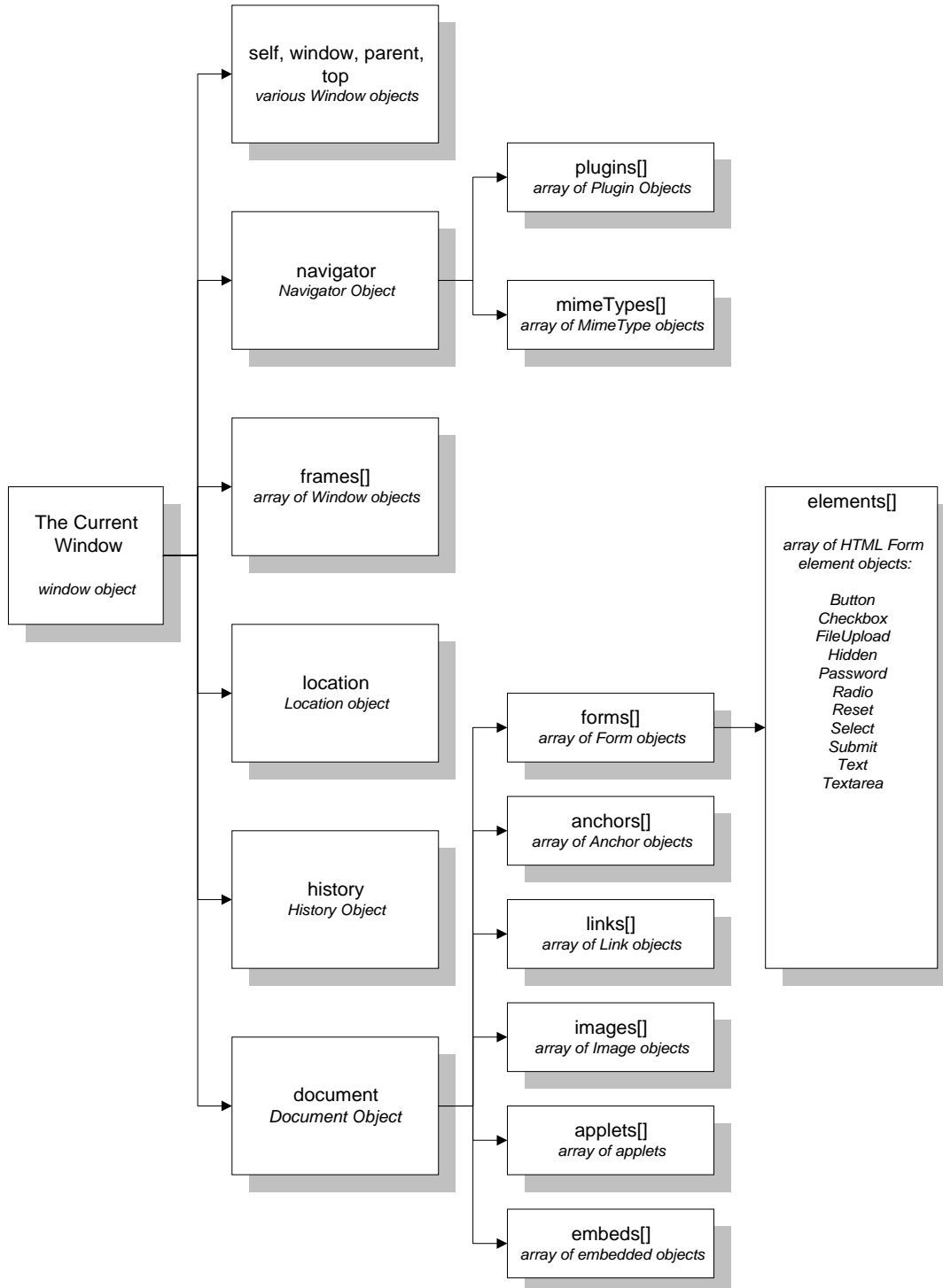
refers to the form named Form1 in this document

`Form1.FullName.value`

refers to the value typed, in the browser by the client, into the textbox named FullName, in the form named Form1, in this document

We recommend that you use the NAME attribute of any HTML tag that you are going to script, as in the above example the form is named Form1. This practice will simplify object naming for you.

The diagram below illustrates the Document Object Model (DOM).



Arrays

JavaScript, similar to other programming languages, has the capabilities to use a data structure called an array. An array is a collection of data where each piece of data is held in a numbered position within the array. A one-dimensional array would be similar to a column in a spreadsheet.

Each position in the array is assigned an index number beginning with 0 for the first position, 1 for the second position, and so on. This allows any position within the array, or “element” of the array, to be referenced by its index number.

The index of the array is indicated by the number contained within the square brackets. An array is implemented as an object in JavaScript and can be created using the “new” statement.

```
var a = new Array(); // creates an array called “a”  
a[0] = 1.2; // sets the first element  
a[1] = "JavaScript"; // sets the second element
```

Arrays are important to understand because a number of components of the Document Object Model (DOM) are implemented as arrays, like forms, images, and elements.

Events

Events are the triggers that call (start) one of your functions. Your client-side JavaScript programs will not execute (run / be interpreted) unless started by an event. An event could be an action such as clicking on a button or placing your mouse over an image. We will use the `onClick` event for starting our form validation scripts, and the `onMouseOver` event for creating graphic images that change when you place your cursor over them.

The available event handlers in JavaScript are: (remember JavaScript is case-sensitive)

`onClick()`

A click event occurs when a button, checkbox, radio button, reset button, or submit button is clicked. This event is regularly used with a button to start script execution.

```
<INPUT TYPE="Button" VALUE="Click Me"
onClick="window.alert('You Clicked me');">
```

In the above example when you click on the button "Click Me" it will execute the JavaScript statement `window.alert('You Clicked me');`. You can call any function or method this way.

`onSubmit()`

A submit event occurs when the user submits a form. This event is regularly used with a form and a submit button to start the form validation script.

```
<FORM action="http://www.xnu.com/formtest.asp"
onSubmit="return checkform();">
```

In the above example when the user clicks on the submit button, it will execute the function `checkform()`. If the form passes all the validation tests, a true value is returned and the data will be passed to the `formtest.asp` CGI program. If the form contents do not pass the validation tests, it will not send the data to the `formtest.asp` CGI program.

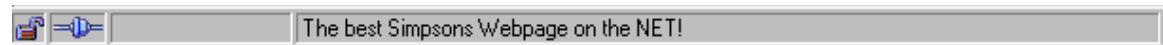
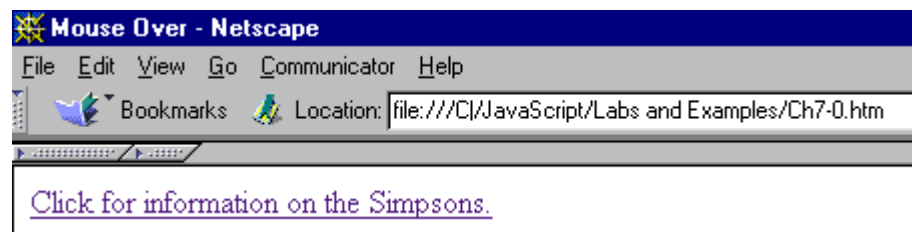
Note: We will be using these events in our discussion of Alerts, Prompts, and Confirmations; as well as, for Form Validation.

onMouseOver()

An onMouseOver event occurs when the user positions their mouse over a hyperlink, or a linked region of a client-side image map.

```
<a href="http://synergy.simplenet.com/simpsons/"  
onMouseOver="window.status='The best Simpsons Webpage on the  
NET!'; return true;">Click for information on the Simpsons.</A>
```

In the above example an HTML link to a web site is created using the anchor tag. The onMouseOver() event is added to the anchor tag, and in this case the script will put a message in the status line of the browser window. Take a look at the results below.



Note the message in the status line of the browser window.

onMouseOut()

An onMouseOut event occurs when the user moves their mouse off of a hyperlink, or a linked region of a client-side image map.

```
<a href="http://synergy.simplenet.com/simpsons/"  
onMouseOut="window.status='The best Simpsons Webpage on the NET!';  
return true;">Click for information on the Simpsons.</A>
```

This example is the similar to the one above, except when the mouse is over the link, the hyperlink URL is shown in the status line, and when the mouse is moved off the hyperlink the message is displayed.

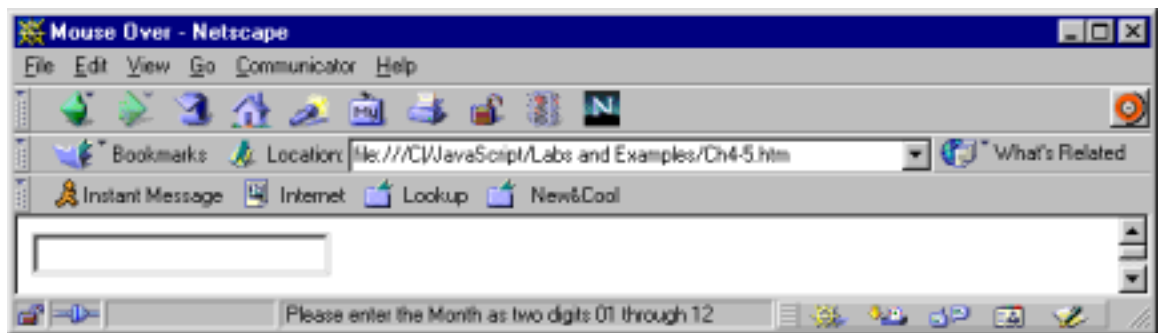
Note: We will be using these events in the Chapter on using Mouse Roll-overs.

onFocus()

This event occurs when a user tabs into or clicks on a password field, a text field, a textarea, or a FileUpload field in an HTML form. If a user clicks on one of these elements in a form, it is receiving the user's focus.

```
<INPUT TYPE="TEXT" NAME="Month"
onFocus="window.status=('Please enter the Month as two digits 01
through 12'); return true;">
```

In this example when the user clicks on the month box or tabs into it a message is displayed in the status line of the browser that indicates what the user should type in.



onChange()

The change event happens when the user leaves a password, text, textarea or FileUpload field in an HTML form, and its value has changed.

```
<INPUT TYPE="TEXT" NAME="Month"
onChange="window.status=('The value of the Month Changed!!!!'); return
true;" >
<INPUT TYPE="TEXT" NAME="Year">
```

onBlur()

The blur event triggers when the user leaves a password, text, textarea or FileUpload field in an HTML form.

```
<INPUT TYPE="TEXT" NAME="Month"
onBlur="window.status=('Do you not care about the value of the Month!');
return true;">
<INPUT TYPE="TEXT" NAME="Year">
```

Note: It is usually a good idea to use either onChange or onBlur, and not both at the same time as they will appear to conflict with one another.

onLoad()

The load event triggers when the browser finishes loading a document or all the frame pages within a <FRAMESET>.

```
<BODY onLoad="alert('Welcome to our website!');">
```

In this example after the page has completed loading into the browser the alert message below is popped up.

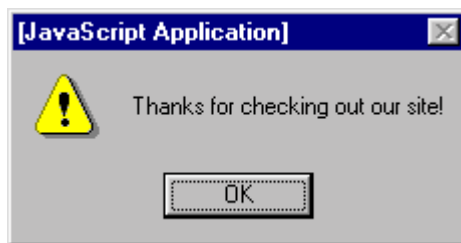


onUnload()

The unload event occurs when you move to a new document. For example if you use the back button, or click on a link to another page the unload event will occur.

```
<BODY onUnload="alert('Thanks for checking out our site!');">
```

This example will popup the following message when someone moves off of the current page.



Note: It is possible to specify multiple events on a single HTML tag. For example,

```
<a href="http://synergy.simplenet.com/simpsons/"  
onMouseOver="mouse1.src='gifs/mouse_over.gif'; "  
onMouseOut="mouse1.src='gifs/mouse_out.gif'; ">
```

We will use this feature in the mouse over section.

The following table shows you what events are supported on the HTML Form Elements.

JavaScript Events and HTML Form Elements

Events / HTML Elements	Blur	Click	Change	Focus	Load	Mouseover	Select	Submit	Unload
Button		X							
Checkbox		X			X				X
Document								X	
Form						X			
Link		X							
Radio		X							
Reset		X							
Selection	X		X	X					
Submit		X							
Text	X		X	X			X		
Textarea	X		X	X			X		

Review Questions

1. An object is a collection of _____ and _____.

2. The four objects that are built into the JavaScript language are:
 - i. _____
 - ii. _____
 - iii. _____
 - iv. _____

3. (True or False) The DOM is a collection of objects added to the JavaScript language by the browser.

4. (True or False) Events are key to beginning the execution of your scripts.

5. Which event would you use to start a script after a page has been fully loaded by the browser?
 - a. onClick()
 - b. onSubmit()
 - c. onLoad()
 - d. onMouseOver()
 - e. onload()

6. (True or False) Events are tied to specific HTML elements.

7. Name four (4) objects within the "document object" of the DOM.

8. What is an array?

Summary

1. Describe an object
2. Use the new operator
3. Understand Document Object Model
4. Create and use an array
5. Use the built-in events
 - i. onClick
 - ii. onSubmit
 - iii. onMouseOver
 - iv. onMouseOut
 - v. onFocus
 - vi. onChange
 - vii. onBlur
 - viii. onLoad
 - ix. onUnload

5

Alerts, Prompts, and Confirms

Objectives

1. Creating Alerts
2. Creating Prompts
3. Creating Confirms

Alerts, Prompts, and Confirms

Alerts, Prompts, and Confirms are all methods of the window object that create predefined, popup style dialogue boxes within a window object.

`window.alert(message)` The “message” is plain text (not HTML) that displays in a dialog box popped up over a window. The alert box will have a button labeled “OK” that will dismiss the alert box when clicked.

```
window.alert("Hello Gary!");
```

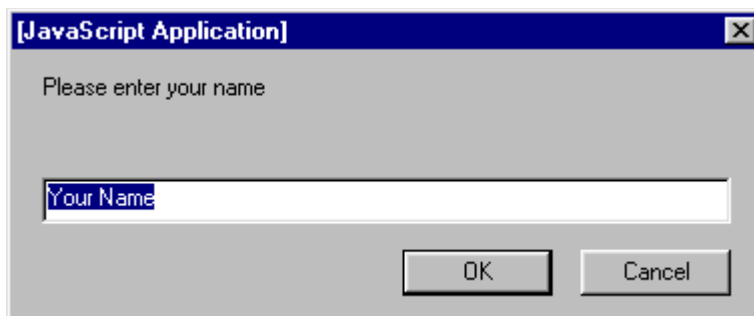
The above line of code would produce the following popup window.



`window.prompt(message, default)` The “message” is again plain text that will be displayed in the dialog box popped up over the window, and the default is a string that is displayed as the default input. When default is set to "", nothing will be in the prompt box. The prompt window will have an input field for the user to enter information and an “OK” button and a “Cancel” button. This method will return the value that the user types in.

```
window.prompt('Please enter your name','Your Name');
```

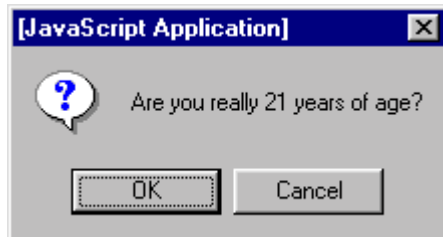
The above line of code would produce the following popup window



`window.confirm(question)` The question is another plain text string. It should be a question that you want answered by the user.

```
window.confirm('Are you really over 21 years of age?');
```

The above line of code would produce the following popup window



If the user clicks on the OK button the method will return a Boolean value of true. If however the user clicks on the cancel button the method will return a Boolean value of false. These return values can be stored in a variable like `ageTest` and used with branching logic.

```
ageTest = window.confirm('Are you really over 21 years of age?');
```

Our lab for this section will demonstrate using all three window methods.

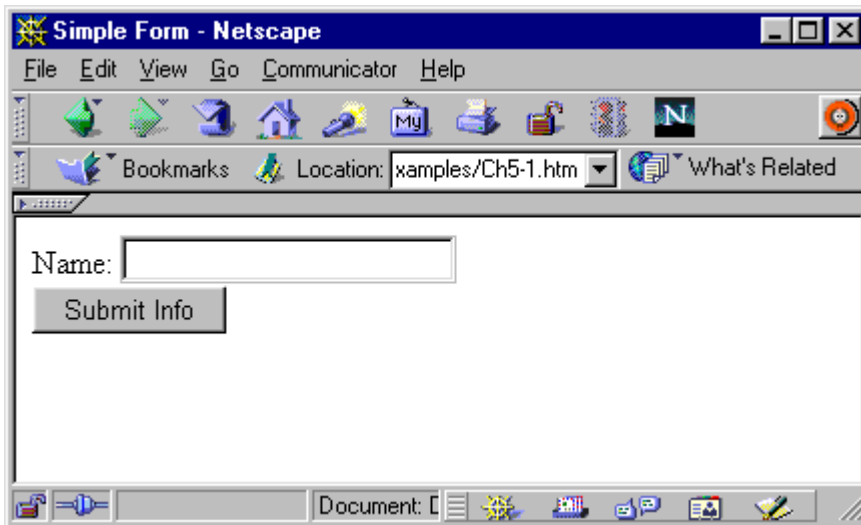
Let's examine the following HTML Form example:

```
<HTML>
<HEAD>
  <TITLE>Simple Form</TITLE>
</HEAD>

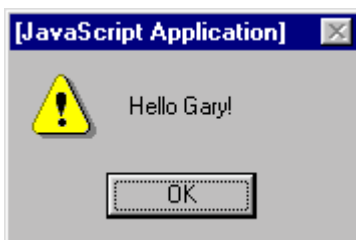
<BODY>

<FORM NAME="Form1">
Name: <INPUT TYPE="TEXT" NAME="Name"><BR>
<INPUT TYPE="Button" VALUE="Submit Info">
</FORM>

</BODY>
</HTML>
```



At this point the form will do nothing when the "Submit Info" button is pressed. What we will first do is use the onClick event on the button to popup an alert message that says Hello "The name typed in" followed by an exclamation mark.



To make this button function we need to add the onClick event to our HTML page.

```
<INPUT TYPE="Button" VALUE="Submit Info"
onClick="window.alert('Hello ' + document.Form1.Name.value + '!');">
```

This says when the user clicks on the button, the browser will create an alert (the popup). Alert is a method of the window object. In this case the string 'Hello ' will be concatenated with the data typed in by the user in the textbox and concatenated with the exclamation mark. Single quotes are used to delimit the strings in this example because the HTML code requires the double quotation marks as delimiters of the attribute onClick. Remember our discussion of syntax; the JavaScript statement must end with a semicolon.

Try out the following example:

```
<HTML>
<HEAD>
```

```
<TITLE>Using Prompts, Confirms, and Alerts</TITLE>
```

```
<SCRIPT LANGUAGE="JAVASCRIPT">
```

```
<!-- hide JavaScript code from browsers that are not JavaScript enabled
```

```
function do_stuff() {
    var flag=false, fullName;

    fullName = prompt("Please enter your name,");

    while (flag == false) {
        // if the user confirms their name is correct the flag gets set to true
        // and the loop test fails and execution continues with the alert line
        flag = confirm("Is your name really ' + fullName);

        // If the user does not confirm that their name is correct the user if
        // the user is given a chance to re-enter it.
        if (flag == false) {
            fullName = prompt("Please re-enter your name,");
        }
    }
    // Display a custom message to the user
    alert(fullName + ', welcome to our website!');
}

//end hiding of JavaScript code -->
</SCRIPT>
```

```
</HEAD>  
  <BODY onload="do_stuff()">  
  
    <H1>Using Prompts, Confirms, and Alerts</H1>  
  
  </BODY>  
</HTML>
```

Review Questions

1. What are the predefined, popup style, dialogue boxes we can use with JavaScript?
2. Which type of dialogue box would you use if you want the user to enter some information?
3. What is concatenation?

Summary

1. Able to create Alerts
2. Able to create Prompts
3. Able to create Confirms

6

Creating Effective Forms

Objectives

1. Creating a FORM with an Active Cursor
2. Using FORM Checking
3. The required Logic
4. Creating a function
5. Adding the checks
6. Testing

Creating a FORM with an Active Cursor

When a web page, containing a FORM element, loads in the browser there is no default condition that places the cursor inside the first input element. If the user just starts to type, nothing happens because the cursor is not active, anywhere in the document in the browser.

Many authors that create forms for use on the web would like to have the cursor placed and active in the first input field. It would mean that a person browsing wouldn't have to click in the field before entering data; they could simply begin to type.

FORM elements of the type text, password, textarea and file upload support, not only events such as onClick and onBlur but they also support the methods of "focus" and "blur". This means that you can tell the cursor to be placed in the input field or that it can't be placed in specified fields. If you combine this feature with the page's onLoad event you can place the cursor in a designated field when the page loads.

```
<HTML>
<HEAD>
  <TITLE>Simple Form</TITLE>
</HEAD>
<BODY onLoad="document.CardForm.HoldersName.focus()">

<FORM NAME="CardForm" action="http://www.xnu.com/formtest.asp">
Enter Cardholders Name: <INPUT TYPE="TEXT" NAME="HoldersName"><BR>
<INPUT TYPE="Submit" VALUE="Submit Info">
</FORM>

</BODY>
</HTML>
```

Using Form Checking

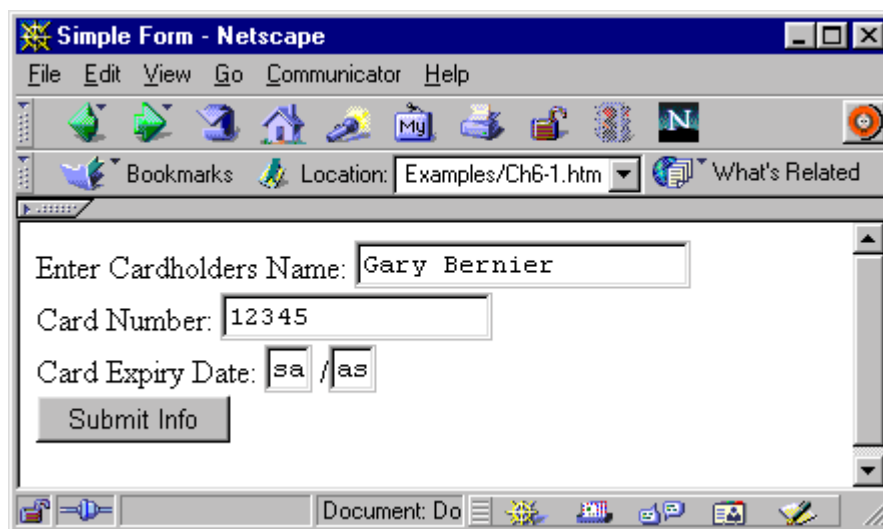
Form validation is accomplished by using JavaScript to preprocess the information the user types into a form before the data is sent to a CGI based server application. This practice is more efficient than allowing incorrectly formatted data to be sent to the server. If the information is incorrectly formatted you can alert the user with a pop-up and force them to correct the mistake. Form validation is one of the most common uses of JavaScript.

To examine this process we expand on our FORM in our web page:

```
<HTML>
<HEAD>
  <TITLE>Simple Form</TITLE>
</HEAD>
<BODY onLoad="document.CardForm.HoldersName.focus()">

<FORM NAME="CardForm" action="http://www.xnu.com/formtest.asp">
Enter Cardholders Name: <INPUT TYPE="TEXT" NAME="HoldersName"><BR>
Card Number: <INPUT TYPE="TEXT" NAME="CardNumber" SIZE="16"
MAXLENGTH="16"><BR>
Card Expiry Date: <INPUT TYPE="TEXT" NAME="CardMonth" SIZE="2"
MAXLENGTH="2">
/<INPUT TYPE="TEXT" NAME="CardYear" SIZE="2" MAXLENGTH="2"><BR>
<INPUT TYPE="Submit" VALUE="Submit Info">
</FORM>

</BODY>
</HTML>
```



This form has, as much “checking” as there is available with HTML. The card number field is restricted to 16 characters, but does not require 16 at this point. Our month and year fields are broken into two text areas that will only allow two characters each, but do not require two.

Our script must check:

1. To make sure a name is entered into the Cardholder's Name field. We will test this by making sure the user has typed in at least one character.
2. To make sure the card number is 16 characters long, and is made up of only numbers. We will do this by testing the length of the string typed in by user, and testing to make sure that it is only made up of numbers.
3. To make sure the month and year are entered as numbers, and that they are valid. This means the month must be between 1 and 12 and the year must be greater than or equal to this current year.
4. If any of these criteria are not met then an alert should notify the user of what is wrong, and the information should not be sent to the server for processing.

Lets start by adding our required script tags and comments; as well as, creating a new function called checkform.

```
<SCRIPT LANGUAGE="JAVASCRIPT">
<!-- hide JavaScript code from browsers that are not JavaScript enabled
function checkform() {

}
//end hiding of JavaScript code -->
</SCRIPT>
```

The function checkform() does not currently have any instructions to execute. We will add these shortly. In order to call this function we will add the following onSubmit event to the FORM HTML tag.

```
onSubmit="return checkform();"
```

Now when someone clicks on the Submit Info button it will first execute the checkform() function before sending the information to the web server for processing. Note the we are using the “return” property, associated with the “checkform” function, which looks for a value of true or false from the associated function. If checkform() returns a value of true, then the form contents will be submitted to the CGI script, on the server, for processing. If checkform() returns a value of false the contents will not be submitted and some other action may be initiated such as an alert.

The next piece of code we will add is a string variable named "message" that will hold the contents of the error message to be displayed to the user as an alert. We will also add an "if" structure that will test for the length of the error message. If the "message" variable's length is greater than zero then an alert message should be displayed, if it is zero then the information is correct and should be sent to the server for processing.

```
<SCRIPT LANGUAGE="JAVASCRIPT">
<!-- hide JavaScript code from browsers that are not JavaScript enabled
function checkform() {
var message = "";

if ( message.length > 0 ) {
    alert( message );
    return false;
}
else {
    return true;
}

} // end of the function checkform()
//end hiding of JavaScript code -->
</SCRIPT>
```

Now that we have this structure in place we can create our first real test of our form. Lets check to see if the length of the card holder's name is greater than zero.

```
if (document.CardForm.HoldersName.value.length == 0) {
    message = message + "Please enter the name on the credit card!\n";
}
```

This statement appears complex at first, but lets step through it to see how it works. The "if" is testing the value typed into the HoldersName field of our HTML FORM named CardForm that is in the document. This accurately references the object's value, the .length will return the length of the number of characters in the string. If the length is equal to zero we know that the user has typed nothing in, and we should create an error message. Creating the error message is the purpose of the next line. The error message we want to display is "Please enter the name on the credit card!" The \n at the end of the line will cause a carriage return and a linefeed so that the next error message will display on a new line inside the same alert box. We know that we will be testing each field for a correct value so we may have a different error message associated with each test. We add the current value of the message variable using the (+) concatenation operator and store the result back in the message variable using the (=) assignment operator.

We can now add the next three tests that are all the same structure.

```
<SCRIPT LANGUAGE="JAVASCRIPT">
<!-- hide JavaScript code from browsers that are not JavaScript enabled

function checkform() {

var message = ""; // create error message string variable, with nothing in it

if (document.CardForm.HoldersName.value.length == 0) {
    message = message + "Please enter the name on the credit card!\n";
}

if (document.CardForm.CardNumber.value.length < 16) {
    message = message + "Please type in all 16 digits of the complete credit
card number!\n";
}

if (document.CardForm.CardMonth.value.length < 2) {
    message = message + "Please type in both digits of the expiry month!\n";
}

if (document.CardForm.CardYear.value.length < 2) {
    message = message + "Please type in both digits of the expiry year!\n";
}

if ( message.length > 0 ) { // is there an error message?
    alert( message ); // display error message
    return false; // return false, not ok to process
}
else {
    return true; // no error message to display, return ok to process
}

} // end of the function checkform()

//end hiding of JavaScript code -->
</SCRIPT>
```

Note: You will notice that we have added comments to some of the code. This is great for remembering what something is doing down the road. Commenting also makes it easier for others to understand your code.

```

if (document.CardForm.CardMonth.value.length < 2) {
    // test to make sure two characters were entered
    message = message + "Please type in both digits of the expiry month!\n";
}
else if (isNaN(document.CardForm.CardMonth.value)) {
    // test for numeric digits only
    message = message + "The Month value needs to be between 1
and 12\n";
}
else if (document.CardForm.CardMonth.value < 1 ||
document.CardForm.CardMonth.value > 12 ) {
    // test the values to make sure they are not less than 1 or
// greater than 12
    message = message + "The Month value needs to be
between 1 and 12\n";
}

```

The above code segment is built upon our length test. Now that we know we have two characters we must check to see that they are numeric. Once that has been determined we need to check to make sure the numeric values are within the range we are interested in.

We use a new function in the code segment; `isNaN()`, which means "is Not a Number". `isNaN()` will return a value of true if the string being evaluated is not a number, or it will return a value of false if the string being evaluated is made up of only numeric digits.

We use the operators `<` "less than", `>` "greater than", and `||` "or". So the last expression is read as follows; "if the value is less than one or greater than 12 then display an error message."

Once you have added this code to your web page, you can test all of the conditions with the following inputs: blank, aa, 1, -1, 13, and 06.

Then we add similar lines to setup the test for a valid year. Use 99, 98, 00 for test values. As you can tell, your script is not year 2000 compliant.

Add the necessary lines to test for 16 digits in the card number. Test with the following inputs: 123456789abcdefg, 123456789, 1234567890123456.

Complete Script

```
<HTML>
<HEAD>
  <TITLE>Simple Form</TITLE>
  <SCRIPT LANGUAGE="JAVASCRIPT">
    <!-- hide JavaScript code from browsers that are not JavaScript enabled

    function checkform() {

      var message = ""; // create error message string variable, with nothing in it

      // Test the Card Holder's Name Length
      if (document.CardForm.HoldersName.value.length == 0) {
        message = message + "Please enter the name on the credit card!\n";
      }

      // Test the Card Number
      if (document.CardForm.CardNumber.value.length < 16) {
        message = message + "Please type in all 16 digits of the credit card number!\n";
      }
      else if (isNaN(document.CardForm.CardNumber.value)) {
        // test for numeric digits only
        message = message + "The Card Number must contain only numbers.\n";
      }

      // Test the Card Expiry Month
      if (document.CardForm.CardMonth.value.length < 2) {
        // test to make sure two characters where entered
        message = message + "Please type in both digits of the expiry month!\n";
      }
      else if (isNaN(document.CardForm.CardMonth.value)) {
        // test for numeric digits only
        message = message + "The Month value needs to be between 1 and 12\n";
      }
      else if (document.CardForm.CardMonth.value < 1 ||
        document.CardForm.CardMonth.value > 12 ) {
        // test the values to make sure they are between 1 and 12
        message = message + "The Month value needs to be between 1 and 12\n";
      }

      // Test the Card Expiry Year
      if (document.CardForm.CardYear.value.length < 2) {
        message = message + "Please type in both digits of the expiry year!\n";
      }
      else if (isNaN(document.CardForm.CardYear.value)) {
        // test for numeric digits only
        message = message + "The Year value needs to be only digits.\n";
      }
      else if (document.CardForm.CardYear.value < 99) {
        // test the values to make sure they are greater than 99
        message = message + "The Year value needs to be greater than 99\n";
      }

      if ( message.length > 0 ) { // is there an error message?
        alert( message ); // display error message
        return false; // return bad, not ok to process
      }
      else {
        return true; // no error message to display, return ok to process
      }

    } // end of the function checkform()

    //end hiding of JavaScript code -->
  </SCRIPT>
</HEAD>
```

```
<BODY onLoad="document.CardForm.HoldersName.focus()">

<FORM NAME="CardForm" action="http://www.xnu.com/formtest.asp" onSubmit="return checkform();">
Enter Cardholders Name: <INPUT TYPE="TEXT" NAME="HoldersName"><BR>
Card Number: <INPUT TYPE="TEXT" NAME="CardNumber" SIZE="16" MAXLENGTH="16"><BR>
Card Expiry Date: <INPUT TYPE="TEXT" NAME="CardMonth" SIZE="2" MAXLENGTH="2">
/<INPUT TYPE="TEXT" NAME="CardYear" SIZE="2" MAXLENGTH="2"><BR>
<INPUT TYPE="Submit" VALUE="Submit Info">
</FORM>

</BODY>
</HTML>
```

Review Questions

1. What event was used to place the cursor in the first field of our FORM?
2. What did we test to see if the user had entered any information into any of the fields?
3. What does the “return” operator do?
4. How do we get the next error message to appear on the next line down?
5. What function is used to verify that certain entries are numbers?

Summary

1. Able to create a FORM with an Active Cursor
2. Use FORM Checking
3. Create the required Logic
4. Create a function
5. Add checks
6. Test all input cases

7

Mouse Over Effects

In this module you will create a very flexible and robust script that you can use over and over again to create mouse roll over effects.

Objectives

1. The image object
2. Mouse-over
3. Creating flexible functions
4. Pre-loading images
5. The eval function
6. Testing for completion and compatibility

Image Object

When you add the HTML tag to your web page you are creating an image object. For example:

```
<IMG SRC="gifs/mouse_out.gif">
```

The image objects on your page can be referenced as an array, where document.image[0] refers to the first image on the page, and document.image[1] the second and so on. If you add the NAME attribute to the tag we can use the value of the NAME attribute to reference the image object. For example:

```
<IMG NAME="mouse1" src="gifs/mouse_out.gif" border=0>
```

This image object can be referenced as document.image['mouse1']. Remember that JavaScript is case sensitive and "mouse1" is different from "Mouse1" or "MOUSE1".

The generic syntax we will be using in this section is:

```
document.images['imagenam'].property = newvalue;
```

The image object has the following properties:

border

This value can only be read, and is the width of the border around the image specified in pixels by the BORDER attribute.

complete

This value can only be read, and is a Boolean value that will let you know if the image has completed downloading (true) or not (false).

height

This is a read only value that specifies the height in pixels of image set by the HEIGHT attribute.

hspace

This is a read-only value, and refers to the space on the right and left of the image specified in pixels by the HSPACE attribute.

lowsrc

This is a **read/write** string that specifies a URL of an alternate image. The image should be suitable for display at low resolutions; it is specified by the LOWSRC attribute.

name

This is a read-only value, specified by the NAME attribute. As you will see it is a good idea to name your images.

src

This is a **read/write** string that specifies a URL of the image.

vspace

This is a read-only value, and refers to the space on the top and bottom of the image specified in pixels by the VSPACE attribute.

width

This is a read only value that specifies the width in pixels of image set by the WIDTH attribute.

Since we can read and write to the src property, we can change the image that is displayed in the browser. This is what allows us to create mouse roll-over, sometimes called hover buttons, when used in combination with the <A> element in creating links.

Mouse-over

In the mouse rollover script that we will create, we will be changing the src property of the appropriate image object when the onMouseOver and onMouseOut events are triggered. We will start with a very basic and simple implementation and then add to it to make it more flexible and powerful.

To create the effect we need two graphic images that are generally the same with a little bit of difference for example mouse_out.gif and mouse_over.gif, both located in the \gifs subdirectory of your student files.



The following HTML page codes with the addition of the JavaScript events will change the image when the user positions their mouse over the image and change it back again when they move their mouse off the image. The image is originally set to the mouse off image using the tag.

```
<HTML>
<HEAD>
<TITLE>Mouse Over and Mouse Out</TITLE>
</HEAD>
<BODY>
<a href="http://synergy.simplenet.com/simpsons/"
  onMouseOver="mouse1.src='gifs/mouse_over.gif'; "
  onMouseOut="mouse1.src='gifs/mouse_out.gif'; ">
  <IMG name="mouse1" src="gifs/mouse_out.gif" border=0></A>
</BODY>
</HTML>
```

Creating Flexible Functions

To improve our mouse over script we want to start by creating two function. One that will be called when the mouse is on the image, and one that is called when the mouse is off the image.

```
<HTML>
<HEAD>
<TITLE>Web Page containing JavaScript</TITLE>

<SCRIPT LANGUAGE="JAVASCRIPT">
<!-- hide JavaScript code from browsers that are not JavaScript enabled
    function MouseOn() {
        document.images['mouse1'].src = "mouse_over.gif";
    }

    function MouseOff() {
        document.images['mouse1'].src = "mouse_out.gif";
    }

//end hiding of JavaScript code -->
</SCRIPT>
</HEAD>

<BODY>
<a href="http://synergy.simplenet.com/simpsons/"
    onMouseOver="MouseOn();"
    onMouseOut="MouseOff();" >
    <IMG name="mouse1" src="gifs/mouse_out.gif" border=0></A>
</BODY>
</HTML>
```

As you can see in the above code we have added two functions that do the same thing as the inline script did in the previous example, and we have changed the event triggers to call our new functions. This provides us with the flexibility to apply our function to different images.

Next we will want to add an argument to our functions so that they will accept the name of the image we want to change. We will then combine the image name with a standard file extension to make our function work for any number of buttons.

```
<HTML>
<HEAD>
<TITLE>Web Page containing JavaScript</TITLE>

<SCRIPT LANGUAGE="JAVASCRIPT">
<!-- hide JavaScript code from browsers that are not JavaScript enabled

    function MouseOn(imageName) {
        document.images[imageName].src = "gifs/mouse_over.gif";
    }

    function MouseOff(imageName) {
        document.images[imageName].src = "gifs/mouse_out.gif";
    }

//end hiding of JavaScript code -->
</SCRIPT>
</HEAD>

<BODY>
<a href="http://synergy.simplenet.com/simpsons/"
    onMouseOver="MouseOn('mouse1'); "
    onMouseOut="MouseOff('mouse1'); ">
    <IMG name="mouse1" src="gifs/mouse_out.gif" border=0></A>
<BR>
<a href="http://synergy.simplenet.com/simpsons/"
    onMouseOver="MouseOn('mouse2'); "
    onMouseOut="MouseOff('mouse2'); ">
    <IMG name="mouse2" src="gifs/mouse_out.gif" border=0></A>
</BODY>
</HTML>
```

In this example we pass the value "mouse1" or "mouse2" to our functions. This value in turn gets stored in the variable "imageName". Next we want to focus our attention on the string that is equated to the image name. The string "gifs/mouse_out.gif" is not very flexible. We need to express that as:

```
"gifs/" + imageName + ".gif"
```

The line above will create the correct string value for each set of images to change.

At this point we will need four (4) graphic images; mouse1_over.gif, mouse1_out.gif, mouse2_out.gif and mouse2_over.gif.

```
<HTML>
<HEAD>
<TITLE>Web Page containing JavaScript</TITLE>

<SCRIPT LANGUAGE="JAVASCRIPT">
<!-- hide JavaScript code from browsers that are not JavaScript enabled

    function MouseOn(imageName) {
        document.images[imageName].src = "gifs/" + imageName +
"_over.gif";
    }

    function MouseOff(imageName) {
        document.images[imageName].src = "gifs/" + imageName +
"_out.gif";
    }

//end hiding of JavaScript code -->
</SCRIPT>
</HEAD>

<BODY>
<a href="http://synergy.simplenet.com/simpsons/"
    onMouseOver="MouseOn('mouse1'); "
    onMouseOut="MouseOff('mouse1'); ">
    <IMG name="mouse1" src="gifs/mouse1_out.gif" border=0></A>
<BR>
<a href="http://synergy.simplenet.com/simpsons/"
    onMouseOver="MouseOn('mouse2'); "
    onMouseOut="MouseOff('mouse2'); ">
    <IMG name="mouse2" src="gifs/mouse2_out.gif" border=0></A>
</BODY>
</HTML>
```

Although the above example will work and create the flexibility that we require, it is not optimized. When the browser loads this page it only loads the two mouse out images. When the user positions the mouse over the image for the first time the browser must download the mouse over image. This can be slow and your effect will not display properly. The solution is to pre-load all of our images into the browser's cache so that they can be accessed quickly.

Pre-loading Images

In our previous example we only had two image objects mouse1, and mouse2 respectively. We are going to create four more, and pre-load them into our browser's cache.

We can pre-load all four of our images files in to the browser's cache by creating four new objects mouse1_on, mouse1_off, mouse2_on, mouse2_off. These image objects are not displayed in the browser. We will be using them to hold the images so that we can quickly switch between them. Let's look at an example of how to do this:

```
var mouse1_off = new Image();
mouse1_off.src = "gifs/mouse1_out.gif";

var mouse1_on = new Image();
mouse1_on.src = "gifs/mouse1_over.gif"
```

For each image file, we create a new image object, and set its src attribute to the appropriate filename. In setting this property, the browser decides it needs to download the image file, at which point it is stored in the cache for quick retrieval, when it is called for display.

Our page will now contain six image objects: mouse1, mouse1_on, mouse1_off, mouse2, mouse2_on, and mouse2_off.

To make the image change in the browser we need to make the src attribute of the image object created by the tag in our case mouse1 or mouse2 equal to on objects src or the off objects src.

We can reference the two image objects created by the HTML tag as follows:

```
document.images['mouse1'] or document.images['mouse2']
```

We want to be able to set those images src attribute equal to one of the other objects src attributes.

```
document.images['mouse1'].src = mouse1_on.src  
or  
document.images['mouse1'].src = mouse1_off.src
```

Notice in the above example, to make it work for mouse2 we would just have to substitute mouse2 for mouse1. So for our function we will substitute the mouse1 reference with a variable called imageName.

```
document.images[imageName].src = imageName + "_on.src"  
or  
document.images[imageName].src = imageName + "_off.src"
```

From earlier in the course you will know that JavaScript is going to try to concatenate (join) the string value of the imageName variable with the other string component to form the object name mouse1_on.src or mouse1_off.src.

If we tried to do it this way it wouldn't work because what the JavaScript interpreter would be trying to do would be to make the src parameter equal to mouse1_on.src for example. This appears to sound ok, except the src parameter is expecting a URL for the image, NOT a string. To get around this we will use a special function in JavaScript called eval().

eval()

We place the eval function around the concatenation like so:

```
document.images[imageName].src = eval( imageName + "_on.src" );
```

The order of operations that is followed this time is that the value of the imageName variable is concatenated with the string, to form one string like mouse1_on.src. The eval function then converts that to JavaScript code, meaning that it converts to the reference to the object we created in our pre-loading phase and it's src parameter. This way we are making the src parameter of one object equal to the src parameter of another image object. This effectively flips the object in the browser.

Our new script looks like this:

```
<HTML>
<HEAD>
<TITLE>Web Page containing JavaScript</TITLE>

<SCRIPT LANGUAGE="JAVASCRIPT">
<!-- hide JavaScript code from browsers that are not JavaScript enabled

    var mouse1_off = new Image();
    mouse1_off.src = "gifs/mouse1_out.gif";
    var mouse1_on = new Image();
    mouse1_on.src = "gifs/mouse1_over.gif"

    var mouse2_off = new Image();
    mouse2_off.src = "gifs/mouse2_out.gif";
    var mouse2_on = new Image();
    mouse2_on.src = "gifs/mouse2_over.gif"

    function MouseOn(imageName) {
        document.images[imageName].src = eval( imageName + "_on.src" );
    }

    function MouseOff(imageName) {
        document.images[imageName].src = eval( imageName + "_off.src" );
    }

//end hiding of JavaScript code -->
</SCRIPT>
</HEAD>

<BODY>
<a href="http://synergy.simplenet.com/simpsons/"
    onMouseOver="MouseOn('mouse1'); "
    onMouseOut="MouseOff('mouse1'); ">
    <IMG name="mouse1" src="gifs/mouse_out.gif" border=0></A>
<BR>
<a href="http://synergy.simplenet.com/simpsons/"
    onMouseOver="MouseOn('mouse2'); "
    onMouseOut="MouseOff('mouse2'); ">
    <IMG name="mouse2" src="gifs/mouse_out.gif" border=0></A>
</BODY>
</HTML>
```

Testing for completion and compatibility

We should add two more pieces to our script, a test to make sure the image is completely downloaded and to make sure that the browser supports the image object. We will start by testing for completion.

```
function MouseOn(imageName) {  
    if( eval(imageName + "_on.complete") ) {  
        document.images[imageName].src = eval( imageName + "_on.src" );  
    }  
} //end function MouseOn
```

We want to use the image object's complete property to test to see that the image has been completely downloaded before we try to change to it so the user will never see a partially loaded image. Similar to the use of the eval in the previous section, here we want to combine the value of the imageName variable with the string "_on.complete" to create the complete image name and property, and have JavaScript act on the object instead of the string.

Next we will test for compatibility. We can do this very simply by testing the browser to see whether it supports the image object. Earlier browsers like Microsoft Internet Explorer 3.0, and earlier did not support this object. To make your code browser proof we will add this conditional statement to our code.

```
if (document.images) { // if browser supports images  
    ...do stuff...  
}
```

We need to add this to both of our functions, and to the pre-loading piece of our code.

Completed Script

```
<HTML>
<HEAD>
<TITLE>Web Page containing JavaScript</TITLE>

<SCRIPT LANGUAGE="JAVASCRIPT">
<!-- hide JavaScript code from browsers that are not JavaScript enabled

    if (document.images) { // if browser supports images
        //Preload our images
        var mouse1_off = new Image();
        mouse1_off.src = "gifs/mouse1_out.gif";
        var mouse1_on = new Image();
        mouse1_on.src = "gifs/mouse1_over.gif"

        var mouse2_off = new Image();
        mouse2_off.src = "gifs/mouse2_out.gif";
        var mouse2_on = new Image();
        mouse2_on.src = "gifs/mouse2_over.gif"
    }

    function MouseOn(imageName) {
        if (document.images) { // if browser supports images
            if( eval(imageName + "_on.complete") ) { //is the image completely downloaded
                document.images[imageName].src = eval( imageName + "_on.src" );
            }
        }
    } //end function MouseOn

    function MouseOff(imageName) {
        if (document.images) { // if browser supports images
            if( eval(imageName + "_on.complete") ) { //is the image completely downloaded
                document.images[imageName].src = eval( imageName + "_off.src" );
            }
        }
    } // end function MouseOff

//end hiding of JavaScript code -->
</SCRIPT>
</HEAD>

<BODY>
<a href="http://synergy.simplenet.com/simpsons/"
    onMouseOver="MouseOn('mouse1');"
    onMouseOut="MouseOff('mouse1');" >
    <IMG name="mouse1" src="gifs/mouse_out.gif" border=0></A>
<BR>
<a href="http://synergy.simplenet.com/simpsons/"
    onMouseOver="MouseOn('mouse2');"
    onMouseOut="MouseOff('mouse2');" >
    <IMG name="mouse2" src="gifs/mouse_out.gif" border=0></A>
</BODY>
</HTML>
```

Review Questions

1. What two events are used to create the mouse roll-over effect?
2. What properties of the image object can you read and write?
3. Which property is a Boolean value you can test for?
4. Why do we pre-load our images?
5. Why is it important to test to see if the images have completely downloaded?
6. What would you need to add to our script to get it to work with more than two links?

Summary

1. Use the image object
2. Use the onmouseover, and onmouseout events
3. Create a flexible mouse rollover function
4. Pre-load your images
5. Use the eval function
6. Test for image download completion and browser compatibility

8

Pop-up Windows

Objectives

- 1) Pop-up windows
 - a) `window.open` method
 - b) `window.close` method
- 2) The window features explained
- 3) Creating windows on the fly
- 4) Setting the document colors

Pop-up Windows

In this chapter we are going to be using the *window* object and the *open* and *close* methods. This particular programming technique can be put to good use or it can be misused. A good use is to highlight a product a customer may be interested in.

```
window.open( URL, Name, Features, Replace )
```

Arguments:

URL

the URL of the document to be displayed in the new window.

Name

an optional string that specifies the name property that can be used with the target attribute.

Features

is a comma-separated list of features about the window; alwaysLowered, alwaysRaised, channelmode, dependent, directories, fullscreen, height, hotkeys, innerHeight, innerWidth, left, location, menubar, outerHeight, innerWidth, resizable, screenX, screenY, scrollbars, status, toolbar, top, width, z-lock

Replace

Optional boolean argument, that states weather the page that gets loaded gets added to the page history in the browser. This argument was intended for use when changing the contents of a window.

```
window.close()
```

This method allows the user to close a window.

Window Features Explained

When you create a new pop-up window you will need to turn off any window features that you do not want displayed in the window. As well, you will need to set the size of your new window.

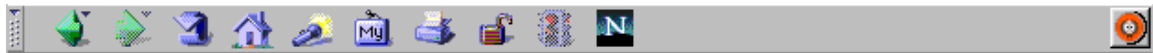
status (status bar): the line at the bottom of the browser that displays information. To display a window with no status bar, set `status=no`.



location: shows the URL of the page you are looking at. To turn the location bar off, `location=no`.



toolbar: contains the buttons you use to move through history (back, and forward) to stop web page loading, etc. To display a window with no toolbar, set `toolbar=no`



menubar: is the bar that has all of the drop down menus. To turn the menubar off, `menubar=no`.



scrollbars: are the sliders on the right hand side and bottom of a window that let you move up or down and left or right. To turn off the scrollbars, `scrollbars=no`.

left, and screenX: control the window's position from the left of your screen in pixels. *Left* is used by Internet Explorer and *screenX* is used by Navigator.

top, and screenY: control the windows position from the top of your screen in pixels. *Top* is used by Internet Explorer and *screenY* is used by Navigator.

height: is the height of the window in pixels.

width: is the width of the window in pixels.

The following feature string sets the height to 500 pixels, width to 400, and turns off the menubar, scrollbars, status line, and the toolbars, and positions the window at the top of the screen one hundred pixels from the left of the screen.

```
window.open('height=500,width=400,menubar=no,scrollbars=no,status=no,toolbar=no,screenX=100,screenY=0,left=100,top=0')
```

It is very common to see popup windows displayed with all of the features turned off.

We will start with our fairly standard HTML page:

```
<HTML>
<HEAD>
  <TITLE>JavaScript Popup Window</TITLE>
  <SCRIPT LANGUAGE="JAVASCRIPT">
    <!-- hide JavaScript code from browsers that are not JavaScript enabled

    function popupwin() {

    }

    //end hiding of JavaScript code -->
  </SCRIPT>
</HEAD>
<BODY>
  <FORM NAME="FORM1">
    <INPUT TYPE="BUTTON" VALUE="Popup Window" ONCLICK="popupwin();">
  </FORM>
</BODY>
</HTML>
```

To the function popupwin() we will add the following line of code:

```
winpopup = window.open('Ch8-0.htm', 'popup',
'height=500,width=400,menubar=no,scrollbars=no,status=no,toolbar=no,screenX
=100,screenY=0,left=100,top=0');
```

This line will create a new window, and load the file Ch8-0.htm into that window. The name of the window is set to "popup". The window will have the following features:

- 500 pixels high
- 400 pixels wide
- no menubars
- the scrollbars
- a status line
- the toolbars

The window will also be located zero pixels from the top of the screen and 100 pixels from the left of the screen. Remember, Microsoft Internet Explorer uses the left and top features, while Netscape Navigator uses screenX and screenY.

If you open up the file Ch8-0.htm you will notice that we used a similar structure to create a close window button.

```
<HTML>
<HEAD>
  <TITLE>JavaScript Popup Window</TITLE>
  <SCRIPT LANGUAGE="JAVASCRIPT">
    <!-- hide JavaScript code from browsers that are not JavaScript enabled

    function closewin() {
      window.close();
    }

    //end hiding of JavaScript code -->
  </SCRIPT>
</HEAD>
<BODY>
  <H1>You called!</H2>
  <IMG SRC="./gifs/teddy.jpg">
  <FORM NAME="FORM1">
    <INPUT TYPE="BUTTON" VALUE="Close Window"
ONCLICK="closewin();">
  </FORM>
</BODY>
</HTML>
```

Creating the windows contents on the fly

We can modify our script to create the document that will appear in our new window instead of displaying a pre-made document as we did previously.

The first thing we need to modify is the `window.open` statement so that it does not load a particular page. We can accomplish this by replacing the file 'Ch8-0.htm' with a single double quote mark, as illustrated:

```
winpopup = window.open('', 'popup', 'height=500,width=400,
                        menubar=no,scrollbars=no,status=no,toolbar=no,
                        screenX=100,screenY=0,left=100,top=0');
```

To create the actual HTML page that will appear in our pop-up window we need to use the `document.write()` method. This method allows us to write text to the specified document. For example:

```
winpopup.document.write('<HTML>\n<HEAD>\n');
```

This line will start the creation of our HTML document in the window identified by the `winpopup` window object. We will need to create the whole HTML document like this. For example:

```
winpopup.document.write('<HTML>\n<HEAD>\n');
winpopup.document.write('<TITLE>This is a popup</TITLE>\n');
winpopup.document.write('</HEAD>\n');
winpopup.document.write('<BODY>\n');
winpopup.document.write('</BODY>\n</HTML>\n');
```

This will create an empty document in the new pop-up window. When you are finished writing information to the new window we should let the browser know that nothing more is coming, we can do that with the `document.close()` method which would be added after the lines above. Our code should now look like this:

```
function popupwin() {
    winpopup =
window.open('', 'popup', 'height=500,width=400,menubar=no,scrollbars=no,status=no,toolb
ar=no,screenX=100,screenY=0,left=100,top=0');
    winpopup.document.write('<HTML>\n<HEAD>\n');
    winpopup.document.write('<TITLE>This is a popup</TITLE>\n');
    winpopup.document.write('</HEAD>\n');
    winpopup.document.write('<BODY>\n');

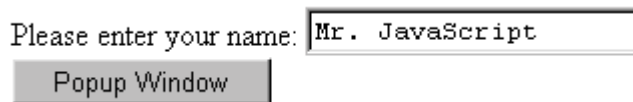
    winpopup.document.write('</BODY>\n</HTML>\n');

    winpopup.document.close(); //Close the Window to additional writes
} //end function popupwin()
```

Now that we have the window opening and displaying an empty HTML page we should make it do something a little more creative. We want our pop-up window to thank our customer for purchasing a product. To start we need to add a text input field to our form.

```
<BODY>
  <FORM NAME="FORM1">
    Please enter your name: <INPUT TYPE="TEXT" NAME="YourName"><BR>
    <INPUT TYPE="BUTTON" VALUE="Popup Window" ONCLICK="popupwin();">
  </FORM>
</BODY>
```

This will create a page that looks like this:



To get "Mr. JavaScript" to display in our window we need to add the following line to our script:

```
winpopup.document.write('<H3>Thank you, ' + document.FORM1>YourName.value + ' for
choosing XYZ Product</H3>\n');
```

This line will combine the strings with the value typed in by the user, and should be located in your script between the lines that write the BODY tags.

We should also give our user the ability to close this newly created pop-up window. To accomplish this we will add the following lines after the one above.

```
winpopup.document.write('<FORM NAME="FORM1">\n');
winpopup.document.write('<INPUT TYPE="BUTTON" VALUE="Close Window"
ONCLICK="window.close();">\n');
winpopup.document.write('</FORM>\n');
```

This will create a button called "Close Window" that when clicked will close the current browser window.

Setting the document colors

When you use popup windows there are some additional properties you may want to control. The document object has properties that allow you to set the document's background color and the default color for text in the document.

The background color can be changed at any time. This allows you to create some interesting effects in your popup window. The text color can only be set once after the body tag has been written to the document (see the completed code). An example of using these properties is below.

```
//set the documents background color  
winpopup.document.backgroundColor = "#0000FF"; RRGGBB
```

```
//set the documents text color RRGGBB  
winpopup.document.fgColor = "#FFC814";
```

Complete Script

```
<HTML>
<HEAD>
  <TITLE>JavaScript Popup Window</TITLE>
  <SCRIPT LANGUAGE="JAVASCRIPT">
    <!-- hide JavaScript code from browsers that are not JavaScript enabled

    function popupwin() {

      winpopup = window.open('', 'popup', 'height=500,width=400,menubar=no,
      scrollbars=no,status=no,toolbar=no,screenX=100,screenY=0,left=100,top=0');

      winpopup.document.write('<HTML>\n<HEAD>\n');
      winpopup.document.write('<TITLE>A Dynamic Popup Window</TITLE>\n');
      winpopup.document.write('</HEAD>\n');
      winpopup.document.write('<BODY>\n');

      //set the documents background color RRGGBB
      winpopup.document.bgColor = "#0000FF";

      //set the documents text color RRGGBB
      winpopup.document.fgColor = "#FFC814";

      winpopup.document.write('<H1>Thank you for choosing XYZ Product</H1>\n');
      winpopup.document.write('<H2>' + document.FORM1.YourName.value +
'</H2>\n');
      winpopup.document.write('<FORM NAME="FORM1">\n');
      winpopup.document.write('<INPUT TYPE="BUTTON" VALUE="Close Window"
ONCLICK="window.close();">\n');
      winpopup.document.write('</FORM>\n');
      winpopup.document.write('</BODY>\n</HTML>\n');

      winpopup.document.close(); //Close the Window to additional writes    }

    //end hiding of JavaScript code -->
  </SCRIPT>
</HEAD>
<BODY>
  <FORM NAME="FORM1">
    <INPUT TYPE="BUTTON" VALUE="Popup Window" ONCLICK="popupwin();">
  </FORM>

</BODY>
</HTML>
```

Review Questions

1. What are the two methods used with popup windows?
2. What are some good uses of popup windows?
3. (True or False) You can not control the size of a popup window?
4. (True or False) It is not common to see popup windows displayed with all of the features turned off.

Summary

- 1) Create pop-up windows
 - a) use the `window.open` method
 - b) use the `window.close` method
- 2) Create a window using the features
- 3) Create windows on the fly
- 4) Set the document colors

G

Glossary

Array(s):

An array is a collection of data that is associated with a single variable name. Each piece of data is considered a value and is indexed numerically within the array.

Comment(s):

Comments are often used by programmers as a means of documenting their program logic so that when they return to update it, or someone else needs to edit it, they can understand what the programmer was doing at the time.

Delimiter(s):

These are the brackets, quotations etc. that surround pieces of code. Examples of delimiters are “ ”, { }, (), []. A start delimiter such as (must be accompanied by a closing delimiter such as).

Event(s):

These are the ‘triggers’ that initiate (start) one of your functions. Client-Side JavaScript will not run unless it is ‘called on’ by an event. Examples of events can include, clicking on a form’s submit button or moving your cursor over an image that is a hyperlink.

Expressions:

Expressions are a set of statements that, as a group, evaluate to a single value. JavaScript then categorizes this resulting value as one of the five data types: number, string, logical, function, or object.

Function(s):

Functions are an important part of programming as they enable you to create chunks of code that perform specific tasks. In JavaScript, functions are referred to as ‘subroutines’ or ‘procedures.’ JavaScript has several built-in functions, but you can also create your own functions for your programs.

Literals:

Literals are data comprised of numbers or strings used to represent fixed values in JavaScript. They are values that do not change during the execution of your scripts.

Method(s):

A method is a function that has been encapsulated in an object.

Object(s):

An object is a collection of variables and functions. JavaScript has four built-in objects including String, Date, Math and Array.

Operator(s):

These are the things that act on variables. There are three types of operators used in the JavaScript language: Computational, Logical and Bitwise. Computational operators perform mathematical tasks such as addition, subtraction, multiplication etc. and are the most commonly used. Logical operators are often used in condition statements like “if” and “switch statements. Bitwise operators work with bits and bytes (or 1s and 0s) and are rarely used.

Statement(s):

A statement is a complete line of JavaScript code.

Syntax:

A set of rules and guidelines used for writing in any language. Each programming language has its own set of rules and conventions. If you do not comply with the language’s syntax, your programs or scripts will not work correctly – if at all.

Variable(s):

A variable is a name assigned to a location in a computer’s memory to store data. Variables must be declared in your programs and can be identified by the fact that they are preceded by the keyword ‘var’. JavaScript, unlike other programming languages, can hold different data types as variables without declaring the type.

Local Variable(s):

These are variables that are only available within a specific function’s braces.

Global Variable(s):

These are variables that are available to all functions. Global variables should be used with caution, as it’s easy to assign the wrong value to a global variable.

A

Answer Appendix

Chapter 1 - Answers

1. (**True** or False) JavaScript is an interpreted language.
2. JAVA is a **compiled** programming language, and is platform **independent**.
3. (True or **False**) JAVA and JavaScript were created by the same company.
4. Microsoft Internet Explorer supports the following scripting languages.
 - a. **JavaScript**
 - b. JAVA
 - c. BASIC
 - d. **VBScript**
 - e. C++
 - f. Perl
6. (**True** or False) JavaScript is supported by a large number of browsers.

Chapter 2 - Answers

1. Which of the following are valid variable or function names:

- a. **y**
- b. 100percent
- c. a big number
- d. break
- e. **subtractTwoNumbers**
- f. **First_Name**

7. Place the brackets in the correct locations indicated by the underscore.

```
if ( z[0] == 0 ) {  
    x = 2;  
}
```

8. Complete this sentence: All statements should end in a semicolon.

9. True or **False**. JavaScript is a case insensitive language.

10. **True** or False. It is a good idea to add comments to your program code.

11. Match the brackets in Column A with the use in Column B.

Column A	Answer	Column B
a. { }	b	used for array index values
b. []	c	used to contain a functions, arguments
c. ()	a	used to contain multiple JavaScript statements

Chapter 3 - Answers

1. True or **False**. Variables in JavaScript are strongly typed.
2. **True** or False. You can declare a number of variables all at once.
3. The **var** keyword is used to declare variables.
4. Match the operator with its function.

Column A	Answer	Column B
a. =	a	assignment
b. ==	c	addition
c. +	b	equality

5. Create an if structure that will make sure the value of the p variable is greater than or equal to 99, and if so set the total equal to the value of p.

```
if ( p >= 99 ) {  
    total = p;  
}
```

6. Create a function that will multiply two numbers together and return the result.

```
function multiplyTwoNum( m, n ) {  
    var total;  
    total = m * n;  
    return total;  
}
```

or

```
function multiplyTwoNum( m, n ) { return m * n; }
```

Chapter 4 - Answers

1. An object is a collection of **methods** and **parameters**.

2. The four objects that are built into the JavaScript language are:
 - i. String**
 - ii. Date**
 - iii. Array**
 - iv. Math**

3. (**True** or False) The DOM is a collection of objects added to the JavaScript language by the browser.

4. (**True** or False) Events are key to beginning the execution of your scripts.

5. Which event would you use to start a script after a page has been fully loaded by the browser?
 - a. onClick()
 - b. onSubmit()
 - c. onLoad()**
 - d. onMouseOver()
 - e. onload()

6. (**True** or False) Events are tied to specific HTML elements.

Chapter 5 - Answers

1. What are the predefined, popup style, dialogue boxes we can use with JavaScript?

- a. **window.popup()**
- b. **window.confirm()**
- c. **window.prompt()**

2. Which type of dialogue box would you use if you want the user to enter some information?

window.prompt()

3. What is concatenation?

Concatenation is the combination (or addition) of two or more strings into one string.

Chapter 6 - Answers

1. What event was used to place the cursor in the first field of our FORM?

onFocus event is used to put the cursor in the first field of the form. It is used in our example in conjunction with the onLoad event.

2. What did we test to see if the user had entered any information into any of the fields?

We tested to see if the length of the string typed in by the user was greater than one character.

3. What does the “return” statement do?

The return sends a value back to the statement that invoked it. When a function is done it can send only one value back, this is done through the return statement.

4. How do we get the next error message to appear on the next line down?

We use the \n (newline literal) in the string.

5. What function is used to verify that certain entries are numbers?

The isNaN() (is not a number) function is used to test a string to see if it only contains numbers.

Chapter 7 - Answers

1. What two events are used to create the mouse roll-over effect?

**onMouseOver
onMouseOut**

2. What properties of the image object can you read and write?

**src (source)
lowsrc (low res source)**

3. Which property is a Boolean value you can test for?

complete

4. Why do we pre-load our images?

So that we can quickly switch between the two image objects when the user is moving their mouse pointer over and off the image.

5. Why is it important to test to see if the images have completely downloaded?

So that we don't try to show a user an image that is partially loaded into their browser.

6. What would you need to add to our script to get it to work with more than two links?

The only thing you need to add to the script is some new image objects for the additional links. The logic section does not need to be updated.

Chapter 8 - Answers

1. What are the two methods used with popup windows?

window.open()

window.close()

2. What are some good uses of popup windows?

Display a picture, or more details about a product that a potential customer is interested in.

3. (True or **False**) You can not control the size of a popup window?
4. (**True** or False) It is not common to see popup windows displayed with all of the features turned off.

Toolbox Café

Web Hosting

NetNation Communications Inc.

Reliable and Affordable Web Site Hosting

www.netnation.com
1-888-277-0000

Online Marketing

Quality Web Hosting

www.olm.net
1-877-265-6638

Interland Web Hosting

Speed, Reliability, Support

www.interland.net
1-800-214-1460

CI Host

Web Hosting Solutions Provider

www.cihost.com
1-888-868-9931

Access

Connect America

International Dial Up Services

www.connectamerica.com
909-778-3640

FlashNet Marketing, Inc.

Internet access products and services available for resale and promotion

www.flashopportunity.com
1-877-225-5364 x480

E-commerce

E-Commerce Exchange

The online entrepreneur's choice for e-commerce since 1989

www.ecx.com
1-800-815-6610

Associations

Webgrrls International

The women's tech knowledge connection

www.webgrrls.com

The HTML Writers Guide

International organization of Web authors

www.hwg.org

Training

TrainingTools.com

Free Courses for Download

www.trainingtools.com
info@trainingtools.com

XNU.COM

Web development training

www.xnu.com
info@xnu.com
1-877-644-3444

Thomas Brunt's OutFront.net

Microsoft Frontpage learning community

www.outfront.net
thomas@outfront.net

Hardware

D-Link Networks

Link for less!

www.dlink.ca
1-800-354-5265

Software

Elettro Inc.

Virtual Tradeshow CD

100TopDownloads.com
info@elettro.com

Let THEM All Know YOU! ONLY **\$110.00** /YEAR
IN THIS TOOLBOX CAFE (PRINT VERSION)  **CLICK HERE**
TRAININGTOOLS.COM INFO@TRAININGTOOLS.COM

**BUILD IT
HOST IT
SELL IT**

CyberSynth.com
Bringing IT All Together

A variety of packages to give your web site maximum impact for minimal expense. Contact Us about our many hosting, commerce, and promotional solutions.



CyberSynth.com
www.cybersynth.com
1.800.342.7076

Index

A

alert 42
Arrays 33

B

Brackets 10
Branches, if 21
Branches, Switch 22

C

Case-sensitivity 8
Checking, forms 51
Closing windows 74
Comments 11
Compiled 2
complete 69
confirm 43
Cursor, positioning 50

D

document.bgColor 80
document.fgColor 80
document.write() 78
DOM 30

E

eval() 67
Events 34
Events, onBlur 36
Events, onChange 36
Events, onClick 34
Events, onFocus 36
Events, onLoad 37
Events, onMouseOut 35, 62
Events, onMouseOver 35, 62
Events, onSubmit 34
Events, onUnload 37

F

for loops 23
Form checking 51
Forms 50
Function Names 12
Functions 24
Functions, built-in 24
Functions, eval() 67
Functions, Flexible 63
Functions, programmer created 24

I

if 21
Image, Object 61
Images, Pre-loading 66

Interpreted 2

J

JavaScript vs. JAVA 2
JavaScript, about 3
JavaScript, inserting into an HTML Page 7
JavaScript, Syntax 8

L

Loops, for 23
Loops, while 23

M

Mouse over script 70
Mouse-over 62

N

new 29

O

Object, Array 33
Objects 29
Objects, document 78, 80
Objects, Image 61
onBlur 36
onChange 36
onClick 34
onFocus 36
onLoad 37
onMouseOut 35
onMouseOver 35
onSubmit 34
Opening windows 74
Operators, a list 19
Operators, new 29
Operators, Using 18

P

parameters, complete 69
Pre-loading images 66
prompt 42

Q

Quotes 9

R

Reserverd Words 13

S

Scripts, Dynamic Window 81
Scripts, form validation 56
Semicolons 8
Strings, quotation marks 9
Switch 22

Index

Syntax	8
Syntax, Brackets	10
Syntax, Case-sensitivity	8
Syntax, Comments	11
Syntax, Reserved Words	13
Syntax, Semicolons	8
Syntax, Variable and Function Names	12
Syntax, white space	8

U

unLoad	37
--------------	----

V

Variable Names	12
Variables, Declaring	17
Variables, Typing	17

W

while loops	23
white space	8
window.alert	42
window.close	74
window.confirm	43
window.open	74
window.prompt	42