JAVASCRIPT

JavaScript is the scripting language of the Web.

JavaScript is used in millions of Web pages to add functionality, validate forms, detect browsers, and much more.

Introduction to JavaScript

JavaScript is used in millions of Web pages to improve the design, validate forms, detect browsers, create cookies, and much more.

JavaScript is the most popular scripting language on the Internet, and works in all major browsers, such as Internet Explorer, Mozilla Firefox, and Opera.

What is JavaScript?

- JavaScript was designed to add interactivity to HTML pages
- JavaScript is a scripting language
- A scripting language is a lightweight programming language
- JavaScript is usually embedded directly into HTML pages
- JavaScript is an interpreted language (means that scripts execute without preliminary compilation)
- Everyone can use JavaScript without purchasing a license

Java and JavaScript are two completely different languages in both concept and design!

Java (developed by Sun Microsystems) is a powerful and much more complex programming language - in the same category as C and C++.

What can a JavaScript Do ?

- **JavaScript gives HTML designers a programming tool -** HTML authors are normally not programmers, but JavaScript is a scripting language with a very simple syntax! Almost anyone can put small "snippets" of code into their HTML pages
- **JavaScript can put dynamic text into an HTML page -** A JavaScript statement like this: document.write("<h1>" + name + "</h1>") can write a variable text into an HTML page
- **JavaScript can react to events -** A JavaScript can be set to execute when something happens, like when a page has finished loading or when a user clicks on an HTML element
- **JavaScript can read and write HTML elements -** A JavaScript can read and change the content of an HTML element
- **JavaScript can be used to validate data -** A JavaScript can be used to validate form data before it is submitted to a server. This saves the server from extra processing
- **JavaScript can be used to detect the visitor's browser -** A JavaScript can be used to detect the visitor's browser, and - depending on the browser - load another page specifically designed for that browser

- **JavaScript can be used to create cookies** - A JavaScript can be used to store and retrieve information on the visitor's computer.

JavaScript Variables

Variables are "containers" for storing information.

JavaScript variables are used to hold values or expressions.

A variable can have a short name, like x, or a more descriptive name, like carname.

Rules for JavaScript variable names:

- Variable names are case sensitive (y and Y are two different variables)
- Variable names must begin with a letter or the underscore character

Note: Because JavaScript is case-sensitive, variable names are case-sensitive.

Example

A variable's value can change during the execution of a script. You can refer to a variable by its name to display or change its value.

```
<html>
<body>
<script type="text/javascript">
var firstname;
firstname="Welcome";
document.write(firstname);
document.write("<br />");
firstname="XYZ";
document.write(firstname);
</script>

<p>The script above declares a variable,
assigns a value to it, displays the value, change the value,
and displays the value again.</p>

</body>
</html>
```

Output :
Welcome
XYZ

The script above declares a variable, assigns a value to it, displays the value, change the value, and displays the value again.

Declaring (Creating) JavaScript Variables

Creating variables in JavaScript is most often referred to as "declaring" variables.

You can declare JavaScript variables with the **var statement**:

```
var x;
var carname;
```

After the declaration shown above, the variables are empty (they have no values yet).

However, you can also assign values to the variables when you declare them:

```
var x=5;
var carname="Scorpio";
```

After the execution of the statements above, the variable **x** will hold the value **5**, and **carname** will hold the value **Scorpio**.

Note: When you assign a text value to a variable, use quotes around the value.

Assigning Values to Undeclared JavaScript Variables

If you assign values to variables that have not yet been declared, the variables will automatically be declared.

These statements:

```
x=5;
carname="Scorpio";
```

have the same effect as:

```
var x=5;
var carname="Scorpio";
```

Redeclaring JavaScript Variables

If you redeclare a JavaScript variable, it will not lose its original value.

```
var x=5;
var x;
```

DataTypes

- Numbers - are values that can be processed and calculated. You don't enclose them in quotation marks. The numbers can be either positive or negative.
- Strings - are a series of letters and numbers enclosed in quotation marks. JavaScript uses the string literally; it doesn't process it. You'll use strings for text you want displayed or values you want passed along.
- Boolean (**true**/**false**) - lets you evaluate whether a condition meets or does not meet specified criteria.
- Null - is an empty value. **null** is not the same as 0 -- 0 is a real, calculable number, whereas **null** is the **absence of any value**.

Data Types

TYPE	EXAMPLE
Numbers	Any number, such as 17, 21, or 54e7
Strings	"Greetings!" or "Fun"
Boolean	Either true or false
Null	A special keyword for exactly that – the null value (that is, nothing)

JavaScript Arithmetic

As with algebra, you can do arithmetic operations with JavaScript variables:

```
y=x-5;
z=y+5;
```

JavaScript Operators

The operator = is used to assign values.

The operator + is used to add values.

The assignment operator = is used to assign values to JavaScript variables.

The arithmetic operator + is used to add values together.

```
y=5;
z=2;
x=y+z;
```

The value of x, after the execution of the statements above is 7.

JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic between variables and/or values.

Given that **y=5**, the table below explains the arithmetic operators:

Operator	Description	Example	Result
+	Addition	x=y+2	x=7
-	Subtraction	x=y-2	x=3
*	Multiplication	x=y*2	x=10
/	Division	x=y/2	x=2.5
%	Modulus (division remainder)	x=y%2	x=1
++	Increment	x=++y	x=6
--	Decrement	x=--y	x=4

JavaScript Assignment Operators

Assignment operators are used to assign values to JavaScript variables.

Given that **x=10** and **y=5**, the table below explains the assignment operators:

Operator	Example	Same As	Result
=	x=y		x=5
+=	x+=y	x=x+y	x=15
-=	x-=y	x=x-y	x=5
*=	x*=y	x=x*y	x=50
/=	x/=y	x=x/y	x=2
%=	x%=y	x=x%y	x=0

The + Operator Used on Strings

The + operator can also be used to add string variables or text values together.

To add two or more string variables together, use the + operator.

```
txt1="What a very";
txt2="nice day";
txt3=txt1+txt2;
```

After the execution of the statements above, the variable txt3 contains "What a verynice day".

To add a space between the two strings, insert a space into one of the strings:

```
txt1="What a very ";
txt2="nice day";
txt3=txt1+txt2;
```

or insert a space into the expression:

```
txt1="What a very";
txt2="nice day";
txt3=txt1+" "+txt2;
```

After the execution of the statements above, the variable txt3 contains:

"What a very nice day"

Adding Strings and Numbers

Look at these examples:

```
x=5+5;
document.write(x);

x="5"+"5";
document.write(x);

x=5+"5";
document.write(x);

x="5"+5;
document.write(x);
```

The rule is:

If you add a number and a string, the result will be a string.

JavaScript Comparison and Logical Operators

Comparison and Logical operators are used to test for true or false.

Comparison Operators

Comparison operators are used in logical statements to determine equality or difference between variables or values.

Given that **x=5**, the table below explains the comparison operators:

Operator	Description	Example
==	is equal to	x==8 is false
===	is exactly equal to (value and type)	x===5 is true x==="5" is false
!=	is not equal	x!=8 is true
>	is greater than	x>8 is false
<	is less than	x<8 is true
>=	is greater than or equal to	x>=8 is false
<=	is less than or equal to	x<=8 is true

How Can it be Used

Comparison operators can be used in conditional statements to compare values and take action depending on the result:

```
if (age<18) document.write("Too young");
```

You will learn more about the use of conditional statements in the next chapter of this tutorial.

Logical Operators

Logical operators are used to determine the logic between variables or values.

Given that **x=6 and y=3**, the table below explains the logical operators:

Operator	Description	Example
&&	and	(x < 10 && y > 1) is true
||	or	(x==5 || y==5) is false
!	not	!(x==y) is true

Conditional Operator

JavaScript also contains a conditional operator that assigns a value to a variable based on some condition.

Syntax

```
variablename=(condition)?value1:value2
```

Example

```
greeting=(visitor=="PRES")?"Dear President ":"Dear ";
```

If the variable **visitor** has the value of "PRES", then the variable **greeting** will be assigned the value "Dear President " else it will be assigned "Dear".

Conditional Statements

Very often when you write code, you want to perform different actions for different decisions. You can use conditional statements in your code to do this.

In JavaScript we have the following conditional statements:

- **if statement** - use this statement if you want to execute some code only if a specified condition is true
- **if...else statement** - use this statement if you want to execute some code if the condition is true and another code if the condition is false
- **if...else if....else statement** - use this statement if you want to select one of many blocks of code to be executed
- **switch statement** - use this statement if you want to select one of many blocks of code to be executed

If Statement

You should use the if statement if you want to execute some code only if a specified condition is true.

Syntax

```
if (condition)
{
code to be executed if condition is true
}
```

Note that if is written in lowercase letters. Using uppercase letters (IF) will generate a JavaScript error!

Example 1

```
<script type="text/javascript">
//Write a "Good morning" greeting if
//the time is less than 10
var d=new Date();
var time=d.getHours();
```

```
if (time<10)
{
document.write("<b>Good morning</b>");
}
</script>
```

Example 2

```
<script type="text/javascript">
//Write "Lunch-time!" if the time is 11
var d=new Date();
var time=d.getHours();

if (time==11)
{
document.write("<b>Lunch-time!</b>");
}
</script>
```

Note: When **comparing** variables you must always use two equals signs next to each other (==)!

Notice that there is no ..else.. in this syntax. You just tell the code to execute some code **only if the specified condition is true**.

If...else Statement

If you want to execute some code if a condition is true and another code if the condition is not true, use the if....else statement.

Syntax

```
if (condition)
{
code to be executed if condition is true
}
else
{
code to be executed if condition is not true
}
```

Example

```
<script type="text/javascript">
//If the time is less than 10,
//you will get a "Good morning" greeting.
//Otherwise you will get a "Good day" greeting.
var d = new Date();
```

```
var time = d.getHours();

if (time < 10)
{
document.write("Good morning!");
}
else
{
document.write("Good day!");
}
</script>
```

If...else if...else Statement

You should use the if....else if...else statement if you want to select one of many sets of lines to execute.

Syntax

```
if (condition1)
{
code to be executed if condition1 is true
}
else if (condition2)
{
code to be executed if condition2 is true
}
else
{
code to be executed if condition1 and
condition2 are not true
}
```

Example

```
<script type="text/javascript">
var d = new Date()
var time = d.getHours()
if (time<10)
{
document.write("<b>Good morning</b>");
}
else if (time>10 && time<16)
{
document.write("<b>Good day</b>");
}
else
```

```
{
document.write("<b>Hello World!</b>");
}
</script>
```

The JavaScript Switch Statement

You should use the switch statement if you want to select one of many blocks of code to be executed.

Syntax

```
switch(n)
{
case 1:
  execute code block 1
  break;
case 2:
  execute code block 2
  break;
default:
  code to be executed if n is
  different from case 1 and 2
}
```

This is how it works: First we have a single expression *n* (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed. Use **break** to prevent the code from running into the next case automatically.

Example

```
<script type="text/javascript">
//You will receive a different greeting based
//on what day it is. Note that Sunday=0,
//Monday=1, Tuesday=2, etc.
var d=new Date();
theDay=d.getDay();
switch (theDay)
{
case 5:
  document.write("Finally Friday");
  break;
case 6:
  document.write("Super Saturday");
  break;
case 0:
  document.write("Sleepy Sunday");
```

```
  break;
default:
  document.write("I'm looking forward to this weekend!");
}
</script>
```

JavaScript Controlling(Looping) Statements

Loops in JavaScript are used to execute the same block of code a specified number of times or while a specified condition is true.

JavaScript Loops

Very often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal lines in a script we can use loops to perform a task like this.

In JavaScript there are two different kind of loops:

- **for** - loops through a block of code a specified number of times
- **while** - loops through a block of code while a specified condition is true

The for Loop

The for loop is used when you know in advance how many times the script should run.

Syntax

```
for (var=startvalue;var<=endvalue;var=var+increment)
{
   code to be executed
}
```

Example

Explanation: The example below defines a loop that starts with i=0. The loop will continue to run as long as **i** is less than, or equal to 10. **i** will increase by 1 each time the loop runs.

Note: The increment parameter could also be negative, and the <= could be any comparing statement.

```
<html>
<body>
<script type="text/javascript">
var i=0;
for (i=0;i<=10;i++)
{
```

```
document.write("The number is " + i);
document.write("<br />");
}
</script>
</body>
</html>
```

Result

```
The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
The number is 6
The number is 7
The number is 8
The number is 9
The number is 10
```

JavaScript While Loop

Loops in JavaScript are used to execute the same block of code a specified number of times or while a specified condition is true.

The while loop

The while loop is used when you want the loop to execute and continue executing while the specified condition is true.

```
while (var<=endvalue)
{
   code to be executed
}
```

Note: The <= could be any comparing statement.

Example

Explanation: The example below defines a loop that starts with i=0. The loop will continue to run as long as **i** is less than, or equal to 10. **i** will increase by 1 each time the loop runs.

```
<html>
```

```
<body>
<script type="text/javascript">
var i=0;
while (i<=10)
{
document.write("The number is " + i);
document.write("<br />");
i=i+1;
}
</script>
</body>
</html>
```

Result

```
The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
The number is 6
The number is 7
The number is 8
The number is 9
The number is 10
```

The do...while Loop

The do...while loop is a variant of the while loop. This loop will always execute a block of code ONCE, and then it will repeat the loop as long as the specified condition is true. This loop will always be executed at least once, even if the condition is false, because the code is executed before the condition is tested.

```
do
{
   code to be executed
}
while (var<=endvalue);
```

Example

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

```
var i=0;
do
{
document.write("The number is " + i);
document.write("<br />");
i=i+1;
}
while (i<0);
</script>
</body>
</html>
```

Result

```
The number is 0
```

JavaScript Break and Continue

There are two special statements that can be used inside loops: break and continue.

JavaScript break and continue Statements

There are two special statements that can be used inside loops: break and continue.

Break

The break command will break the loop and continue executing the code that follows after the loop (if any).

Example

```
<html>
<body>
<script type="text/javascript">
var i=0;
for (i=0;i<=10;i++)
{
if (i==3)
{
break;
}
document.write("The number is " + i);
document.write("<br />");
}
</script>
</body>
```

```
</html>
```

Result

```
The number is 0
The number is 1
The number is 2
```

Continue

The continue command will break the current loop and continue with the next value.

Example

```
<html>
<body>
<script type="text/javascript">
var i=0
for (i=0;i<=10;i++)
{
if (i==3)
{
continue;
}
document.write("The number is " + i);
document.write("<br />");
}
</script>
</body>
</html>
```

Result

```
The number is 0
The number is 1
The number is 2
The number is 4
The number is 5
The number is 6
The number is 7
The number is 8
The number is 9
The number is 10
```

JavaScript Functions

A function (also known as a *method*) is a self-contained piece of code that performs a particular "function". You can recognise a function by its format - it's a piece of descriptive text, followed by open and close brackets.A function is a reusable code-block that will be executed by an event, or when the function is called.

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

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

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

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

Example

```
<html>
<head>
<script type="text/javascript">
function displaymessage()
{
alert("Hello World!");
}
</script>
</head>
<body>
<form>
<input type="button" value="Click me!"
onclick="displaymessage()" >
</form>
</body>
</html>
```

If the line: alert("Hello world!!") in the example above had not been put within a function, it would have been executed as soon as the line was loaded. Now, the script is not executed before the user hits the button. We have added an onClick event to the button that will execute the function displaymessage() when the button is clicked.

You will learn more about JavaScript events in the JS Events chapter.

How to Define a Function

The syntax for creating a function is:

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

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

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

```
function functionname()
{
some code
}
```

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

The return Statement

The return statement is used to specify the value that is returned from the function.

So, functions that are going to return a value must use the return statement.

Example

The function below should return the product of two numbers (a and b):

```
function prod(a,b)
{
x=a*b;
return x;
}
```

When you call the function above, you must pass along two parameters:

```
product=prod(2,3);
```

The returned value from the prod() function is 6, and it will be stored in the variable called product.

The Lifetime of JavaScript Variables

When you declare a variable within a function, the variable can only be accessed within that function. When you exit the function, the variable is destroyed. These variables are called local variables. You can have local variables with the same name in different functions, because each is recognized only by the function in which it is declared.

If you declare a variable outside a function, all the functions on your page can access it. The lifetime of these variables starts when they are declared, and ends when the page is closed.

What is an Event?

Event Handlers

Event Handlers are JavaScript methods, i.e. functions of objects, that allow us as JavaScript programmers to control what happens when events occur.

Directly or indirectly, an **Event** is always the result of something a user does. For example, we've already seen Event Handlers like **onClick** and **onMouseOver** that respond to mouse actions. Another type of Event, an internal change-of-state to the page (**completion** of loading or **leaving** the page). An **onLoad** Event can be considered an indirect result of a user action.

Although we often refer to Events and Event Handlers interchangeably, it's important to keep in mind the distinction between them. An **Event** is merely something that happens - something that it is initiated by an **Event Handler** (**onClick**, **onMouseOver**, etc...).

The elements on a page which can trigger events are known as "targets" or "target elements," and we can easily understand how a button which triggers a *Click* event is a target element for this event. Typically, events are defined through the use of Event Handlers, which are bits of script that tell the browser what to do when a particular event occurs at a particular target. These Event Handlers are commonly written as attributes of the target element's HTML tag.

The Event Handler for a *Click* event at a form field button element is quite simple to understand:

```
<INPUT TYPE="button" NAME="click1" VALUE="Click me for fun!"
 onClick="event_handler_code">
```

The event_handler_code portion of this example is any valid JavaScript and it will be executed when the specified event is triggered at this target element. This particular topic will be continued in Incorporating JavaScripts into your HTML pages.

There are "three different ways" that Event Handlers can be used to trigger Events or Functions.

Method 1 (Link Events):

Places an Event Handler as an attribute within an <A HREF= > tag, like this:

```
<A HREF="foo.html" onMouseOver="doSomething()">   ...   </A>
```

You can use an Event Handler located within an <A HREF= > tag to make either an image or a text link respond to a mouseover Event. Just enclose the image or text string between the <A HREF= > and the </A> tags.

Whenever a user clicks on a link, or moves her cursor over one, JavaScript is sent a **Link Event**. One Link Event is called **onClick**, and it gets sent whenever someone clicks on a link. Another link event is called **onMouseOver**. This one gets sent when someone moves the cursor over the link.

You can use these events to affect what the user sees on a page. Here's an example of how to use link events. Try it out, View Source, and we'll go over it.

```
<A HREF="javascript:void(')"
  onClick="open('index.htm', 'links', 'height=200,width=200');">How to Use Link Events
</A>
```

The first interesting thing is that there are no <SCRIPT> tags. That's because anything that appears in the quotes of an **onClick** or an **onMouseOver** is automatically interpreted as JavaScript. In fact, because semicolons mark the end of statements allowing you to write entire JavaScripts in one line, you can fit an entire JavaScript program between the quotes of an **onClick**. It'd be ugly, but you could do it.

Here are the three lines of interest:

1. <A HREF="#" **onClick**="**alert**('Ooo, do it again!');">Click on me!</A>
2. <A HREF="javascript:**void**(')" **onClick**="**alert**('Ooo, do it again!');">
   Click on me!
   </A>
3. <A HREF="javascript:**alert**('Ooo, do it again!')" >Click on me!</A>

In the first example we have a normal <A> tag, but it has the magic **onClick**="" element, which says, "When someone clicks on this link, run the little bit of JavaScript between my quotes." Notice, there's even a terminating semicolon at the end of the alert. **Question**: is this required? NO.

Let's go over each line:

1. HREF="#" tells the browser to look for the anchor #, but there is no anchor "#", so the browser reloads the page and goes to top of the page since it couldn't find the anchor.
2. <A HREF="javascript:**void**(')" tells the browser not to go anywhere - it "deadens" the link when you click on it. HREF="javascript: is the way to call a function when a link (hyperlink or an HREFed image) is clicked.
3. HREF="javascript:**alert**('Ooo, do it again!')" here we kill two birds with one stone. The default behavior of a hyperlink is to click on it. By clicking on the link we call the window Method **alert**() and also at the same time "deaden" the link.

The next line is

```
<A HREF="javascript:void(")" onMouseOver="alert('Hee hee!');">
Mouse over me!
</A>
```

This is just like the first line, but it uses an **onMouseOver** instead of an **onClick**.

Method 2 (Actions within FORMs):

The second technique we've seen for triggering a Function in response to a mouse action is to place an **onClick** Event Handler inside a button type form element, like this:

```
<FORM>
    <INPUT TYPE="button" onClick="doSomething()">
</FORM>
```

While any JavaScript statement, methods, or functions can appear inside the quotation marks of an Event Handler, typically, the JavaScript script that makes up the Event Handler is actually a call to a function defined in the header of the document or a single JavaScript command. Essentially, though, anything that appears inside a command block (inside curly braces {}) can appear between the quotation marks.

For instance, if you have a form with a text field and want to call the function **checkField**() whenever the value of the text field changes, you can define your text field as follows:

```
<INPUT TYPE="text" onChange="checkField(this)">
```

Nonetheless, the entire code for the function could appear in quotation marks rather than a function call:

```
<INPUT TYPE="text" onChange="if (this.value <= 5) {
    alert("Please enter a number greater than 5");
}">
```

To separate multiple commands in an Event Handler, use semicolons

```
<INPUT TYPE="text" onChange="alert('Thanks for the entry.');
  confirm('Do you want to continue?');">
```

The advantage of using functions as Event Handlers, however, is that you can use the same Event Handler code for multiple items in your document and, functions make your code easier to read and understand.

Method 3 (BODY onLoad & onUnLoad):

The third technique is to us an Event Handler to ensure that all required objects are defined involve the **onLoad** and **onUnLoad**. These Event Handlers are defined in the <BODY> or <FRAMESET> tag of an HTML file and are invoked when the document or frameset are fully loaded or unloaded. If you set a flag

within the **onLoad** Event Handler, other Event Handlers can test this flags to see if they can safely run, with the knowledge that the document is fully loaded and all objects are defined. For example:

```
<SCRIPT>

var loaded = false;

function doit() {
    // alert("Everything is \"loaded\" and loaded = " + loaded);
    alert('Everything is "loaded" and loaded = ' + loaded);
}

</SCRIPT>

<BODY onLoad="loaded = true;">
-- OR --
<BODY onLoad="window.loaded = true;">

<FORM>
    <INPUT TYPE="button" VALUE="Press Me"
      onClick="if (loaded == true) doit();">
-- OR --
    <INPUT TYPE="button" VALUE="Press Me"
      onClick="if (window.loaded == true) doit();">
-- OR --
    <INPUT TYPE="button" VALUE="Press Me"
      onClick="if (loaded) doit();">
</FORM>

</BODY>
```

The **onLoad** Event Handler is executed when the document or frameset is fully loaded, which means that all images have been downloaded and displayed, all subframes have loaded, any Java Applets and Plugins (Navigator) have started running, and so on. The **onUnLoad** Event Handler is executed just before the page is unloaded, which occurs when the browser is about to move on to a new page. Be aware that when you are working with multiple frames, there is no guarantee of the order in which the **onLoad** Event Handler is invoked for the various frames, except that the Event Handlers for the parent frame is invoked after the Event Handlers of all its children frames -- This will be discussed in detail in **Week 8**.

Setting the bgColor Property

The first example allows the user to change the color by clicking buttons, while the second example allows you to change colors by using drop down boxes.

Event Handlers

EVENT	DESCRIPTION
onAbort	the user cancels loading of an image
onBlur	input focus is removed from a form element (when the user clicks outside the field) or focus is removed from a window
onClick	the user clicks on a link or form element
onChange	the value of a form field is changed by the user
onError	an error happens during loading of a document or image
onFocus	input focus is given to a form element or a window
onLoad	once a page is loaded, NOT while loading
onMouseOut	the user moves the pointer off of a link or clickable area of an image map
onMouseOver	the user moves the pointer over a hypertext link
onReset	the user clears a form using the Reset button
onSelect	the user selects a form element's field
onSubmit	a form is submitted (ie, when the users clicks on a submit button)
onUnload	the user leaves a page

Note: *Input focus* refers to the act of clicking on or in a form element or field. This can be done by clicking in a text field or by tabbing between text fields.

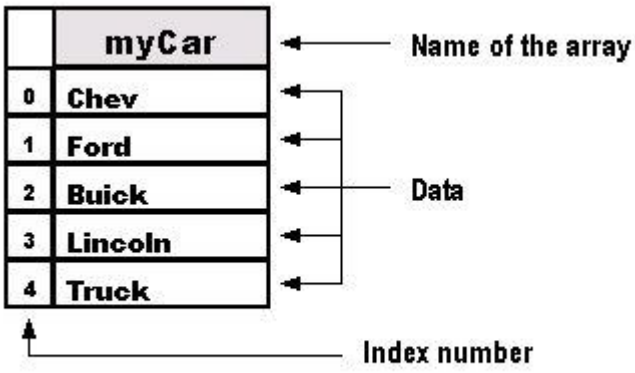
Which Event Handlers Can Be Used

OBJECT	EVENT HANDLERS AVAILABLE
Button element	**onClick, onMouseOver**
Checkbox	**onClick**
Clickable ImageMap area	**onClick, onMouseOver, onMouseOut**
Document	**onLoad, onUnload, onError**
Form	**onSubmit, onReset**
Framesets	**onBlur, onFocus**
Hypertext link	**onClick, onMouseOver, onMouseOut**
Image	**onLoad, onError, onAbort**

Radio button	**onClick**
Reset button	**onClick**
Selection list	**onBlur, onChange, onFocus**
Submit button	**onClick**
TextArea element	**onBlur, onChange, onFocus, onSelect**
Text element	**onBlur, onChange, onFocus, onSelect**
Window	**onLoad, onUnload, onBlur, onFocus**

JavaScript Arrays

An array object is used to create a database-like structure within a script. Grouping data points (*array elements*) together makes it easier to access and use the data in a script. There are methods of accessing actual databases (which are beyond the scope of this series) but here we're talking about small amounts of data.



Comparison of an array to a column of data

An array can be viewed like a column of data in a spreadsheet. The name of the array would be the same as the name of the column. Each piece of data (*element*) in the array is referred to by a number (*index*), just like a row number in a column.

An array is an *object*. Earlier, I said that an object is a thing, a collection of properties (*array elements*, in this case) grouped together.

You can name an array using the same format as a variable, a function or an object. Remember our basic rules: The first character cannot be a number, you cannot use a reserved word, and you cannot use spaces. Also, be sure to remember that the name of the array object is capitalized, e.g. **A**rray.

The JavaScript interpreter uses numbers to access the collection of elements (i.e. the data) in an array. Each index number (as it is the number of the data in the array's index) refers to a specific piece of data in the array, similar to an ID number. It's important to remember that the index numbering of the data starts at "0." So, if you have 8 elements, the first element will be numbered "0" and the last one will be "7."

Elements can be of any type: character string, integer, Boolean, or even another array. An array can even have different types of elements within the same array. Each element in the

array is accessed by placing its index number in brackets, i.e. myCar[4]. This would mean that we are looking for data located in the array myCar which has an index of "4." Since the numbering of an index starts at "0," this would actually be the fifth index. For instance, in the following array,

```
var myCar = new Array("Chev","Ford","Buick","Lincoln","Truck");
alert(myCar[4])
```

the data point with an index of "4" would be Truck. In this example, the indexes are numbered as follows: 0=Chev, 1=Ford, 2=Buick, 3=Lincoln, and 4=Truck. When creating loops, it's much easier to refer to a number than to the actual data itself.

The Size of the Array

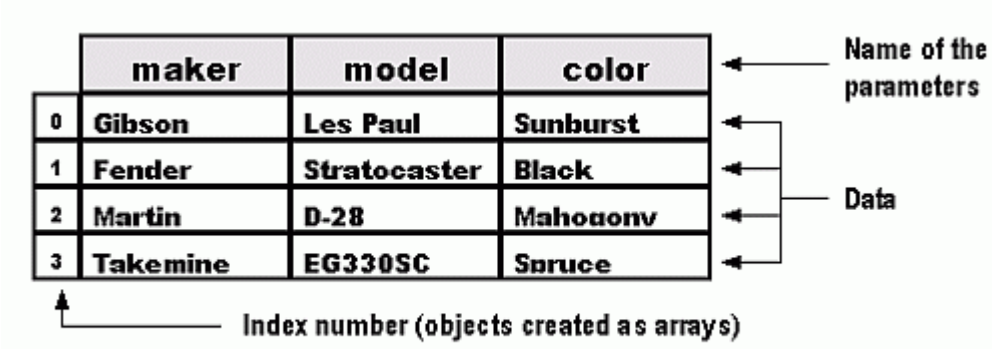
The size of an array is determined by either the actual number of elements it contains or by actually specifying a given size. You don't need to specify the size of the array. Sometimes, though, you may want to pre-set the size, e.g.:

```
var myCar = new Array(20);
```

That would pre-size the array with 20 elements. You might pre-size the array in order to set aside the space in memory.

Multidimensional Arrays

This type of an array is similar to parallel arrays. In a multidimensional array, instead of creating two or more arrays in tandem as we did with the parallel array, we create an array with several levels or "dimensions." Remember our example of a spreadsheet with rows and columns? This time, however, we have a couple more columns.



Comparison of a multidimensional array to a column of data

Multidimensional arrays can be created in different ways. Let's look at one of these method. First, we create the main array, which is similar to what we did with previous arrays.

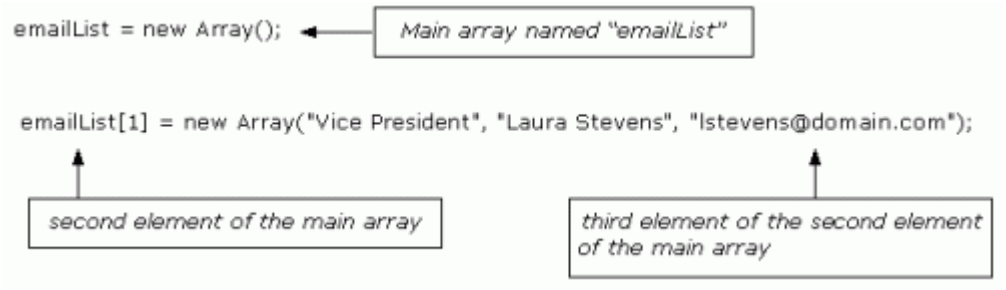
```
var emailList = new Array();
```

Next, we create arrays for elements of the main array:

```
emailList[0] = new Array("President", "Paul Smith", "psmith@domain.com");
emailList[1] = new Array("Vice President", "Laura Stevens", "lstevens@domain.com");
emailList[2] = new Array("General Manager", "Mary Larsen", "mlarsen@domain.com");
emailList[3] = new Array("Sales Manager", "Bob Lark", "blark@domain.com");
```

In this script we created "sub arrays" or arrays from another level or "dimension." We used the name of the main array and gave it an index number (e.g., emailList[0]). Then we created a new instance of an array and gave it a value with three elements.

In order to access a single element, we need to use a double reference. For example, to get the e-mail address for the Vice President in our example above, access the third element "[2]" of the second element "[1]" of the array named emailList.



It would be written like this:

```
var vpEmail = emailList[1][2]
alert("The address is: "+ vpEmail)
```

1. We declared a variable, named it emailList, and initialized it with a value of a new instance of an array.
2. Next, we created an array for each of the elements within the original array. Each of the new arrays contained three elements.
3. Then we declared a variable named vpEmail and initialized it with the value of the third element (lstevens@domain.com) of the second element "[1]" of the array named emailList.

You could also retrieve the information using something like:

```
var title = emailList[1][0]
var email = emailList[1][2]
alert("The e-mail address for the " + title +" is: " + email)
```

Array Properties

length

The length property returns the number of elements in an array. The format is arrayName.length. The length property is particularly useful when using a loop to cycle through an array. One example would be an array used to cycle banners:

```
var bannerImg = new Array();
  bannerImg[0]="image-1.gif";
  bannerImg[1]="image-2.gif";
  bannerImg[2]="image-3.gif";

var newBanner = 0
var totalBan = bannerImg.length

function cycleBan() {
  newBanner++
  if (newBanner == totalBan) {
    newBanner = 0
  }
  document.banner.src=bannerImg[newBanner]
  setTimeout("cycleBan()", 3*1000)
}
window.onload=cycleBan;
```

This portion is then placed in the body where the banner is to be displayed:

```
&lt;img src="image-1.gif" name="banner">
```

Let's take a look and see what happened here:

1. On the first line, we created a new instance of the array bannerImg, and gave it three data elements. (Remember, we are only making a copy of the Array object here.)
2. Next, we created two variables: newBanner, which has a beginning value of zero; and totalBan, which returns the length of the array (the total number of elements contained in the array).
3. Then we created a function named cycleBan. This function will be used to create a loop to cycle the images.
    a. We set the newBanner variable to be increased each time the function cycles. (*Review: By placing the increment operator [" ++ "] after the variable [the "operand"], the variable is incremented only after it returns its current value to the script. For example, its beginning value is "0", so in the first cycle it will return a value of "0" to the script and then its value will be increased by "1".*)
    b. When the value of the newBanner variable is equal to the variable totalBan (which is the length of the array), it is then reset to "0". This allows the images to start the cycle again, from the beginning.
    c. The next statement uses the Document Object Method (DOM - we'll be taking a look at that soon) to display the images on the Web page. Remember, we use the *dot operator* to access the properties of an object. We also read the statement backwards, i.e., "take the element from the array bannerImg, that is specified by the current value of the variable newBanner, and place it in the src attribute located in the element with the name attribute of banner, which is located in the document object."
    d. We then used the setTimeout function to tell the script how long to display each image. This is always measured in milliseconds so, in this case, the function cycleBan is called every 3,000 milliseconds (i.e., every 3 seconds).
4. Finally, we used the window.onload statement to execute the function cycleBan as soon as the document is loaded.

There are a total of five properties for the Array object. In addition to the length property listed above, the others are:

1. *constructor*: Specifies the function that creates an object's prototype.
2. *index*: Only applies to JavaScript arrays created by a regular expression match.
3. *input*: Only applies to JavaScript arrays created by a regular expression match.
4. *prototype*: Used to add properties or methods.

The other properties listed here are either more advanced or seldom used. For now, we'll stick to the basics.

Javascript Object Hierarchy

Hierarchy Objects			
Object	**Properties**	**Methods**	**Event Handlers**
Window	defaultStatus frames opener parent scroll self status top window	alert blur close confirm focus open prompt clearTimeout setTimeout	onLoad onUnload onBlur onFocus
History	length forward go	back	none
Navigator	appCodeName appName appVersion mimeTypes plugins userAgent	javaEnabled	none
document	alinkColor anchors applets area bgColor cookie fgColor forms images lastModified linkColor links location referrer title	clear close open write writeln	none (the onLoad and onUnload event handlers belong to the Window object.

	vlinkColor		
image	border complete height hspace lowsrc name src vspace width	none	none
form	action elements encoding FileUpload method name target	submit reset	onSubmit onReset
text	defaultValue name type value	focus blur select	onBlur onCharge onFocus onSelect

Built-in Objects			
Array	length	join reverse sort xx	none
Date	none	getDate getDay getHours getMinutes getMonth getSeconds getTime getTimeZoneoffset getYear parse prototype setDate setHours setMinutes setMonth setSeconds setTime	none

		setYear toGMTString toLocaleString UTC	
String	length prototype	anchor big blink bold charAt fixed fontColor fontSize indexOf italics lastIndexOf link small split strike sub substring sup toLowerCase toUpperCase	Window

JavaScript Array Object

The Array object is used to store multiple values in a single variable.

Create an Array

The following code creates an Array object called myCars:

```
var myCars=new Array();
```

There are two ways of adding values to an array (you can add as many values as you need to define as many variables you require).

1:

```
var myCars=new Array();
myCars[0]="Saab";
myCars[1]="Volvo";
myCars[2]="BMW";
```

You could also pass an integer argument to control the array's size:

```
var myCars=new Array(3);
myCars[0]="Saab";
myCars[1]="Volvo";
myCars[2]="BMW";
```

2:

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

Note: If you specify numbers or true/false values inside the array then the type of variables will be numeric or Boolean instead of string.

Access an Array

You can refer to a particular element in an array by referring to the name of the array and the index number. The index number starts at 0.

The following code line:

```
document.write(myCars[0]);
```

will result in the following output:

```
Saab
```

Modify Values in an Array

To modify a value in an existing array, just add a new value to the array with a specified index number:

```
myCars[0]="Opel";
```

Now, the following code line:

```
document.write(myCars[0]);
```

will result in the following output:

```
Opel
```

JavaScript Date Object

Create a Date Object

The Date object is used to work with dates and times.

The following code create a Date object called myDate:

```
var myDate=new Date()
```

Note: The Date object will automatically hold the current date and time as its initial value!

Set Dates

We can easily manipulate the date by using the methods available for the Date object.

In the example below we set a Date object to a specific date (14th January 2010):

```
var myDate=new Date();
myDate.setFullYear(2010,0,14);
```

And in the following example we set a Date object to be 5 days into the future:

```
var myDate=new Date();
myDate.setDate(myDate.getDate()+5);
```

Note: If adding five days to a date shifts the month or year, the changes are handled automatically by the Date object itself!

Compare Two Dates

The Date object is also used to compare two dates.

The following example compares today's date with the 14th January 2010:

```
var myDate=new Date();
myDate.setFullYear(2010,0,14);
var today = new Date();
if (myDate>today)
{
alert("Today is before 14th January 2010");
}
else
{
alert("Today is after 14th January 2010");
}
```

JavaScript Math Object

Math Object

The Math object allows you to perform mathematical tasks.

The Math object includes several mathematical constants and methods.

Syntax for using properties/methods of Math:

```
var pi_value=Math.PI;
var sqrt_value=Math.sqrt(16);
```

Note: Math is not a constructor. All properties and methods of Math can be called by using Math as an object without creating it.

Mathematical Constants

JavaScript provides eight mathematical constants that can be accessed from the Math object. These are: E, PI, square root of 2, square root of 1/2, natural log of 2, natural log of 10, base-2 log of E, and base-10 log of E.

You may reference these constants from your JavaScript like this:

```
Math.E
Math.PI
Math.SQRT2
Math.SQRT1_2
Math.LN2
Math.LN10
Math.LOG2E
Math.LOG10E
```

Mathematical Methods

In addition to the mathematical constants that can be accessed from the Math object there are also several methods available.

The following example uses the round() method of the Math object to round a number to the nearest integer:

```
document.write(Math.round(4.7));
```

The code above will result in the following output:

```
5
```

The following example uses the random() method of the Math object to return a random number between 0 and 1:

```
document.write(Math.random());
```

The code above can result in the following output:

```
0.4218824567728053
```

The following example uses the floor() and random() methods of the Math object to return a random number between 0 and 10:

```
document.write(Math.floor(Math.random()*11));
```

The code above can result in the following output:

```
4
```

JavaScript String Object

String object

The String object is used to manipulate a stored piece of text.

Examples of use:

The following example uses the length property of the String object to find the length of a string:

```
var txt="Hello world!";
document.write(txt.length);
```

The code above will result in the following output:

```
12
```

The following example uses the toUpperCase() method of the String object to convert a string to uppercase letters:

```
var txt="Hello world!";
document.write(txt.toUpperCase());
```

The code above will result in the following output:

```
HELLO WORLD!
```

Window Object

The Window object is the top level object in the JavaScript hierarchy.

The Window object represents a browser window.

A Window object is created automatically with every instance of a <body> or <frameset> tag.

IE: Internet Explorer, **F:** Firefox, **O:** Opera.

Window Object Collections

Collection	Description	IE	F	O
frames[]	Returns all named frames in the window	4	1	9

Window Object Properties

Property	Description	IE	F	O
closed	Returns whether or not a window has been closed	4	1	9
defaultStatus	Sets or returns the default text in the statusbar of the window	4	No	9
document	See Document object	4	1	9
history	See History object	4	1	9
length	Sets or returns the number of frames in the window	4	1	9
location	See Location object	4	1	9
name	Sets or returns the name of the window	4	1	9
opener	Returns a reference to the window that created the window	4	1	9
outerHeight	Sets or returns the outer height of a window	No	1	No
outerWidth	Sets or returns the outer width of a window	No	1	No
pageXOffset	Sets or returns the X position of the current page in relation to the upper left corner of a window's display area	No	No	No
pageYOffset	Sets or returns the Y position of the current page in relation to the upper left corner of a window's display area	No	No	No
parent	Returns the parent window	4	1	9
personalbar	Sets whether or not the browser's personal bar (or directories bar) should be visible			
scrollbars	Sets whether or not the scrollbars should be visible			
self	Returns a reference to the current window	4	1	9
status	Sets the text in the statusbar of a window	4	No	9
statusbar	Sets whether or not the browser's statusbar should be visible			
toolbar	Sets whether or not the browser's tool bar is visible or not (can only be set before the window is opened and you must have UniversalBrowserWrite privilege)			
top	Returns the topmost ancestor window	4	1	9

Window Object Methods

Method	Description	IE	F	O
alert()	Displays an alert box with a message and an OK button	4	1	9
blur()	Removes focus from the current window	4	1	9
clearInterval()	Cancels a timeout set with setInterval()	4	1	9
clearTimeout()	Cancels a timeout set with setTimeout()	4	1	9
close()	Closes the current window	4	1	9
confirm()	Displays a dialog box with a message and an OK and a Cancel button	4	1	9
createPopup()	Creates a pop-up window	4	No	No
focus()	Sets focus to the current window	4	1	9
moveBy()	Moves a window relative to its current position	4	1	9
moveTo()	Moves a window to the specified position	4	1	9
open()	Opens a new browser window	4	1	9
print()	Prints the contents of the current window	5	1	9
prompt()	Displays a dialog box that prompts the user for input	4	1	9
resizeBy()	Resizes a window by the specified pixels	4	1	9
resizeTo()	Resizes a window to the specified width and height	4	1.5	9
scrollBy()	Scrolls the content by the specified number of pixels	4	1	9
scrollTo()	Scrolls the content to the specified coordinates	4	1	9
setInterval()	Evaluates an expression at specified intervals	4	1	9
setTimeout()	Evaluates an expression after a specified number of milliseconds	4	1	9

Document Object

The Document object represents the entire HTML document and can be used to access all elements in a page.

The Document object is part of the Window object and is accessed through the window.document property.

IE: Internet Explorer, **F:** Firefox, **O:** Opera, **W3C:** World Wide Web Consortium (Internet Standard).

Document Object Collections

Collection	Description	IE	F	O	W3C
anchors[]	Returns a reference to all Anchor objects in the document	4	1	9	Yes
forms[]	Returns a reference to all Form objects in the document	4	1	9	Yes
images[]	Returns a reference to all Image objects in the document	4	1	9	Yes
links[]	Returns a reference to all Area and Link objects in	4	1	9	Yes

	the document				

Document Object Properties

Property	Description	IE	F	O	W3C
body	Gives direct access to the <body> element				
cookie	Sets or returns all cookies associated with the current document	4	1	9	Yes
domain	Returns the domain name for the current document	4	1	9	Yes
lastModified	Returns the date and time a document was last modified	4	1	No	No
referrer	Returns the URL of the document that loaded the current document	4	1	9	Yes
title	Returns the title of the current document	4	1	9	Yes
URL	Returns the URL of the current document	4	1	9	Yes

Document Object Methods

Method	Description	IE	F	O	W3C
close()	Closes an output stream opened with the document.open() method, and displays the collected data	4	1	9	Yes
getElementById()	Returns a reference to the first object with the specified id	5	1	9	Yes
getElementsByName()	Returns a collection of objects with the specified name	5	1	9	Yes
getElementsByTagName()	Returns a collection of objects with the specified tagname	5	1	9	Yes
open()	Opens a stream to collect the output from any document.write() or document.writeln() methods	4	1	9	Yes
write()	Writes HTML expressions or JavaScript code to a document	4	1	9	Yes
writeln()	Identical to the write() method, with the addition of writing a new line character after each expression	4	1	9	Yes

History Object

The History object is actually a JavaScript object, not an HTML DOM object.

The History object is automatically created by the JavaScript runtime engine and consists of an array of URLs. These URLs are the URLs the user has visited within a browser window.

The History object is part of the Window object and is accessed through the window.history property.

IE: Internet Explorer, **F:** Firefox, **O:** Opera.

History Object Properties

Property	Description	IE	F	O
length	Returns the number of elements in the history list	4	1	9

History Object Methods

Method	Description	IE	F	O
back()	Loads the previous URL in the history list	4	1	9
forward()	Loads the next URL in the history list	4	1	9
go()	Loads a specific page in the history list	4	1	9

Form Object

The Form object represents an HTML form.

For each instance of a <form> tag in an HTML document, a Form object is created.

IE: Internet Explorer, **F:** Firefox, **O:** Opera, **W3C:** World Wide Web Consortium (Internet Standard).

Form Object Collections

Collection	Description	IE	F	O	W3C
elements[]	Returns an array containing each element in the form	5	1	9	Yes

Form Object Properties

Property	Description	IE	F	O	W3C
acceptCharset	Sets or returns a list of possible character-sets for the form data	No	No	No	Yes
action	Sets or returns the action attribute of a form	5	1	9	Yes
enctype	Sets or returns the MIME type used to encode the content of a form	6	1	9	Yes
id	Sets or returns the id of a form	5	1	9	Yes
length	Returns the number of elements in a form	5	1	9	Yes
method	Sets or returns the HTTP method for sending data to the server	5	1	9	Yes
name	Sets or returns the name of a form	5	1	9	Yes
target	Sets or returns where to open the action-URL in a form	5	1	9	Yes

Standard Properties

Property	Description	IE	F	O	W3C
className	Sets or returns the class attribute of an element	5	1	9	Yes
dir	Sets or returns the direction of text	5	1	9	Yes
lang	Sets or returns the language code for an element	5	1	9	Yes
title	Sets or returns an element's advisory title	5	1	9	Yes

Form Object Methods

Method	Description	IE	F	O	W3C
reset()	Resets the values of all elements in a form	5	1	9	Yes
submit()	Submits a form	5	1	9	Yes

Image Object

The Image object represents an embedded image.

For each instance of an <img> tag in an HTML document, an Image object is created.

IE: Internet Explorer, **F:** Firefox, **O:** Opera, **W3C:** World Wide Web Consortium (Internet Standard).

Image Object Properties

Property	Description	IE	F	O	W3C
align	Sets or returns how to align an image according to the surrounding text	5	1	9	Yes
alt	Sets or returns an alternate text to be displayed, if a browser cannot show an image	5	1	9	Yes
border	Sets or returns the border around an image	4	1	9	Yes
complete	Returns whether or not the browser has finished loading the image	4	1	9	No
height	Sets or returns the height of an image	4	1	9	Yes
hspace	Sets or returns the white space on the left and right side of the image	4	1	9	Yes
id	Sets or returns the id of the image	4	1	9	Yes
isMap	Returns whether or not an image is a server-side image map	5	1	9	Yes
longDesc	Sets or returns a URL to a document containing a description of the image	6	1	9	Yes
lowsrc	Sets or returns a URL to a low-resolution version of an image	4	1	9	No
name	Sets or returns the name of an image	4	1	9	Yes
src	Sets or returns the URL of an image	4	1	9	Yes
useMap	Sets or returns the value of the usemap attribute of an client-side image map	5	1	9	Yes
vspace	Sets or returns the white space on the top and bottom of the image	4	1	9	Yes
width	Sets or returns the width of an image	4	1	9	Yes

Standard Properties

Property	Description	IE	F	O	W3C
className	Sets or returns the class attribute of an element	5	1	9	Yes
title	Sets or returns an element's advisory title	5	1	9	Yes

Area Object

The Area object represents an area of an image-map (An image-map is an image with clickable regions).

For each instance of an <area> tag in an HTML document, an Area object is created.

IE: Internet Explorer, **F:** Firefox, **O:** Opera, **W3C:** World Wide Web Consortium (Internet Standard).

Area Object Properties

Property	Description	IE	F	O	W3C
accessKey	Sets or returns the keyboard key to access an area	5	1	No	Yes
alt	Sets or returns an alternate text to be displayed, if a browser cannot show an area	5	1	9	Yes
coords	Sets or returns the coordinates of a clickable area in an image-map	5	1	9	Yes
hash	Sets or returns the anchor part of the URL in an area	4	1	No	No
host	Sets or returns the hostname and port of the URL in an area	4	1	No	No
href	Sets or returns the URL of a link in an image-map	4	1	9	Yes
id	Sets or returns the id of an area	4	1	9	Yes
noHref	Sets or returns whether an area should be active or inactive	5	1	9	Yes
pathname	Sets or returns the pathname of the URL in an area	4	1	9	No
protocol	Sets or returns the protocol of the URL in an area	4	1	9	No
search	Sets or returns the query string part of the URL in an area	4	1	9	No
shape	Sets or returns the shape of an area in an image-map	5	1	9	Yes
tabIndex	Sets or returns the tab order for an area	5	1	9	Yes
target	Sets or returns where to open the link-URL in an area	4	1	9	Yes

Standard Properties

Property	Description	IE	F	O	W3C
className	Sets or returns the class attribute of an element	5	1	9	Yes
dir	Sets or returns the direction of text	5	1	9	Yes
lang	Sets or returns the language code for an element	5	1	9	Yes
title	Sets or returns an element's advisory title	5	1	9	Yes

Navigator Object

The Navigator object is actually a JavaScript object, not an HTML DOM object.

The Navigator object is automatically created by the JavaScript runtime engine and contains information about the client browser.

IE: Internet Explorer, **F:** Firefox, **O:** Opera.

Navigator Object Collections

Collection	Description	IE	F	O
plugins[]	Returns a reference to all embedded objects in the document	4	1	9

Navigator Object Properties

Property	Description	IE	F	O
appCodeName	Returns the code name of the browser	4	1	9
appMinorVersion	Returns the minor version of the browser	4	No	No
appName	Returns the name of the browser	4	1	9
appVersion	Returns the platform and version of the browser	4	1	9
browserLanguage	Returns the current browser language	4	No	9
cookieEnabled	Returns a Boolean value that specifies whether cookies are enabled in the browser	4	1	9
cpuClass	Returns the CPU class of the browser's system	4	No	No
onLine	Returns a Boolean value that specifies whether the system is in offline mode	4	No	No
platform	Returns the operating system platform	4	1	9
systemLanguage	Returns the default language used by the OS	4	No	No
userAgent	Returns the value of the user-agent header sent by the client to the server	4	1	9
userLanguage	Returns the OS' natural language setting	4	No	9

Navigator Object Methods

Method	Description	IE	F	O
javaEnabled()	Specifies whether or not the browser has Java enabled	4	1	9
taintEnabled()	Specifies whether or not the browser has data tainting enabled	4	1	9

ZIP CODE VALIDATION

<!-- TWO STEPS TO INSTALL ZIP CODE VALIDATION:

1. Copy the coding into the HEAD of your HTML document

2. Add the last code into the BODY of your HTML document -->

<!-- STEP ONE: Paste this code into the HEAD of your HTML document -->

<HEAD>

<SCRIPT LANGUAGE="JavaScript">

<!-- Original: Brian Swalwell -->

```
<!-- This script and many more are available free online at -->

<!-- The JavaScript Source!! http://javascript.internet.com -->

<!-- Begin

function validateZIP(field) {

var valid = "0123456789-";

var hyphencount = 0;

if (field.length!=5 && field.length!=10) {

alert("Please enter your 5 digit or 5 digit+4 zip code.");

return false;

}

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

temp = "" + field.substring(i, i+1);

if (temp == "-") hyphencount++;

if (valid.indexOf(temp) == "-1") {

alert("Invalid characters in your zip code.  Please try again.");

return false;

}

if ((hyphencount > 1) || ((field.length==10) && ""+field.charAt(5)!="-")) {

alert("The hyphen character should be used with a properly formatted 5 digit+four zip code, like '12345-
6789'.   Please try again.");

return false;

   }

}

return true;
```

```
}

//  End -->

</script>

</HEAD>


<!-- STEP TWO: Copy this code into the BODY of your HTML document  -->


<BODY>

<center>

<form name=zip onSubmit="return validateZIP(this.zip.value)">

Zip: <input type=text size=30 name=zip>

<input type=submit value="Submit">

</form>

</center>

<p><center>

<font face="arial, helvetica" size="-2">Free JavaScripts provided<br>

by <a href="http://javascriptsource.com">The JavaScript Source</a></font>

</center><p>
```