STRATEGIC WEBSITE TECHNOLOGIES

LECTURE 3
(JAVA SCRIPT)

Introduction

JavaScript is a scripting language that was jointly created by Netscape and Sun.

Some of the properties of JavaScript are the following:

•A Simple, Easy to Learn Object-Based Scripting Language

•Designed for Programming User Events

•Derived from Java but not Java

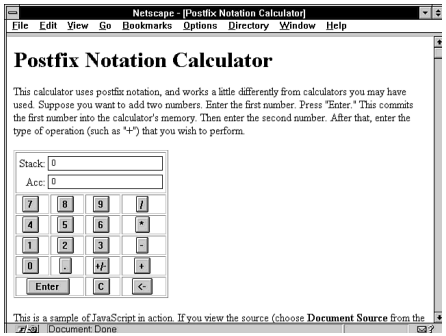
•An Interpreted Language

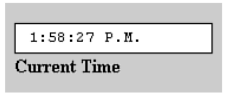
•Limited client-server interaction

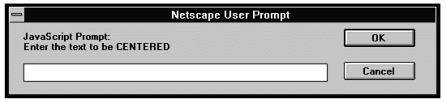
•Integrated into HTML

•Platform Independence

Examples on using JavaScript







Every script must be contained inside a SCRIPT container tag.

In order to hide the JavaScript program from those of browsers that do not scripting the program is put into a comment container:

```
<SCRIPT LANGUAGE="JavaScript">
<!-- HIDE THE SCRIPT FROM OTHER BROWSERS
JavaScript program
// STOP HIDING FROM OTHER BROWSERS -->
</SCRIPT>
```

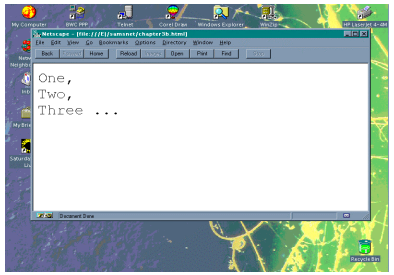
The Simplest JavaScript Program

```
<BODY>
   <H3>The following text is script generated:</H3>
   <SCRIPT LANGUAGE="JavaScript">
   <!--
   /* Our script only requires one quick statement! */
   document.write("Hello World!")
   // end hiding-->
   </SCRIPT>
</BODY>
```

The following text is script generated:

Hello World!

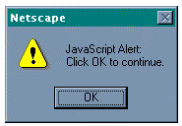
```
<PRE>
<SCRIPT LANGUAGE="JavaScript">
<!-- HIDE FROM OTHER BROWSERS
document.writeln("One,");
document.writeln("Two,");
document.write("Three ");
document.write("...");
// STOP HIDING FROM OTHER BROWSERS -->
</SCRIPT>
</PRE>
```



Working with dialog boxes

JavaScript provides the ability for programmers to generate output in small dialog boxes. The simplest way to direct output to a dialog box is to use the alert() method. E.g.

alert("Click OK to continue."); →



Interacting with the User

The simplest way to interact with the user is with the prompt() method. E.g.

prompt("Enter Your favourite colour:","Blue");

Data Types in JavaScript

JavaScript uses four data types—numbers, strings, boolean values, and a null value

Literals

The term literals refers to the way in which each of the four data types are represented. Literals are fixed values which literally provide a value in a program. For example, 11 is a literal number, "hello" is a string literal and **true** is a boolean literal.

Integers

Integers are numbers without any portion following the decimal point; (1,56, -45)

Floating Point Values

Floating point values can include a fractional component. (1.43)

Strings

A string literal contains zero or more characters enclosed in single or double quotes:

"Hello!", '245', ""

Boolean

A boolean literal can take two values: either true or false.

The null Value

The null value is a special value in JavaScript representing just that—nothing.

Declaring Variables

Although it is possible to declare variables by simply using them, declaring variables helps to ensure that programs are well organized and to keep track of the scope of variables.

You can declare a variable using the var command:

var example; (or)

var example = "An Example";

Rules for JavaScript Variable Names

There are specific rules you must follow when choosing a variable name:

•Variable names can include letters of the alphabet, both upper- and lowercase. They can also include the digits 0-9 and the underscore (_) character.

•Variable names cannot include spaces or any other punctuation character.

•The first character of the variable name must be either a letter or the underscore character.

•Variable names are case-sensitive; totalnum, Totalnum, and TotalNum are separate variable names.

•There is no official limit on the length of variable names, but they must fit within one line.

Incorporating Variables in a Script

```
<HEAD>
  <TITLE>Example 3.1</TITLE>
  <SCRIPT LANGUAGE="JavaScript">
    <!--HIDE FROM OTHER BROWSERS
    var name=prompt("Enter Your Name:","Name");
    // STOP HIDING FROM OTHER BROWSERS -->
  </SCRIPT>
</HEAD>
<BODY>
  <SCRIPT LANGUAGE="JavaScript">
    <!-- HIDE FROM OTHER BROWSERS
    document.write('<IMG SRC="logo.gif">');
    document.write("<H1>Greetings, " + name +
       ". Welcome to JavaScript World!</H1>");
    // STOP HIDING FROM OTHER BROWSERS -->
  </SCRIPT>
</BODY>
```

Functions

Functions offer the ability for programmers to group together program code that performs a specific task—or function— into a single unit that can be used repeatedly throughout a program.

They are generally defined in the <HEAD> part using this syntax:

```
function function_name(value_name) {
    ...function code...
    return (new_value)
}
```

E.g.

```
function printName(name) {
  document.write("<HR>Your Name is <B><I>");
  document.write(name);
  document.write("</B></I><HR>");
}
```

```
<HTML>
  <HEAD>
  <TITLE>JavaScript table test</TITLE>
  <SCRIPT language="JAVASCRIPT">
  function PrintRow(name, age, birthday) {
    document.write("<TR> <TD>", name, "</TD> <TD>", age, "</TD> <TD>", birthday,
   "</TD> </TR>\n");
  }
  </SCRIPT>
  </HEAD>
  <BODY>
  <H1>Table Test Using JavaScript</H1>
  <TABLE>
  <SCRIPT language="JAVASCRIPT">
  PrintRow("Fred", 27, "June 17");
  PrintRow("Tom", 24, "March 13");
  PrintRow("Rebecca", 25, "November 30");
  </SCRIPT>
  </TABLE>
  End of table.
  </HTML>
```

```
<HEAD>
 <TITLE>Simple Math</TITLE>
 <SCRIPT> <!--
 function simple_math(num) {
   document.write("The call passed ",num," to the function.<BR>");
   new_num = num * 2;        // multiple the value by 2
   document.write(num, " * 2 equals ",new_num,"<BR>");
   return new_num;           // return new_num to the function call
 }
 // end hiding -->
</SCRIPT>
</HEAD>
<BODY>
  <H3>Let's watch some simple math:</H3>
  <SCRIPT>  <!--
    x = 5;
    document.write("The starting number is ",x,"<BR>");
    new_x = simple_math(x);
    document.write("The function returned the number ",new_x,"<BR>");
  // end hiding -->
  </SCRIPT>
</BODY>
</HTML>
```

Let's watch some simple math:

The starting number is 5
The call passed 5 to the function.
5 * 2 equals 10
The function returned the number 10

Variable Conversions

JavaScript is a loosely typed language. You don't need to declare a variable's type when you define it, and you don't need to do anything special to store a different type of data in a variable.

There are, however, some cases when you want to convert data to a specific type.

The parseInt Function

The following statement assigns the variable a to the value 39:

```
a = parseInt("39 steps");
```

The parseFloat Function

The following statement assigns the value 2.7178 to the variable a:

```
a = parseFloat("2.7178 is the base of a natural logarithm.“);
```

The eval Function

```
a = eval("20 + 1 + 4");
var text = "Fred";
var statement = text + "= 31";
eval(statement);
```

Arithmetic Operators

The standard binary arithmetic operators : addition (+), subtraction (-), multiplication (*), and division (/).

In addition to these basic operators, there is the modulus (%) operator which calculates the remainder of dividing its operands.

8 + 5 → 13
25.5 - 17.3 → 8.2
12 % 5 → 2

if-else construct

Using the if-else construct, combined with expressions, it is possible to alter the flow of a program—to determine which sections of program code run based on a condition.

if condition {
 several lines of JavaScript code
}
else {
 several lines of JavaScript code
}

```
if (day == "Saturday") {
  document.writeln("It's Saturday!");
} else {
  document.writeln("It's not Saturday.");
}
```

Conditional Operator

Conditional expressions can evaluate to one of two different values based on a condition. The structure of a conditional expression is:

(condition) ? val1 : val2

The way a conditional expression works is that the condition, which is any expression that can be evaluated to a boolean value, is evaluated; based on the result, the whole expression either evaluates to val1 (true condition) or val2 (false condition).

The expression

(day == "Saturday") ? “It is Saturday!" : “It isn’t Saturday."

evaluates to “It is Saturday!" when day is "Saturday". Otherwise the expression evaluates to “It isn’t Saturday!".

```
<HTML> <HEAD> <TITLE>Example 3.4</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!-- HIDE FROM OTHER BROWSERS
var question="What is 10+10?";
var answer=20;
var correct='CONGRATS!';
var incorrect='YOU HAVE TO STUDY MATHS!';
var response = prompt(question,"0");
if (response != answer) {
  // THE ANSWER WAS WRONG: OFFER A SECOND CHANCE
  if (confirm("Wrong! Press OK for a second chance."))
    response = prompt(question,"0");
}
var output = (response == answer) ? correct : incorrect;
// STOP HIDING FROM OTHER BROWSERS -->
</SCRIPT> </HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
<!-- HIDE FROM OTHER BROWSERS
// OUTPUT RESULT
document.write(output);
// STOP HIDING FROM OTHER BROWSERS -->
</SCRIPT> </BODY> </HTML>
```

Comparison Operators

== Returns true if the operands are equal

!= Returns true if the operands are not equal

> Returns true if the left operand is greater than the right operand

< Returns true if the left operand is less than the right operand

>= Returns true if the left operand is greater than or equal to the right operand

<= Returns true if the left operand is less than or equal to the right operand

Logical Operators

&& Logical "and"—returns true when both operands are true; otherwise it returns false.

|| Logical "or"—returns true if either operand is true. It only returns false when both operands are false.

! Logical "not"—returns true if the operand is false and false if the operand is true. This is a unary operator and precedes the operand.

Operator Precedence

Operator precedence is the set of rules which determine the order in which these compound expressions are evaluated.

The operators are evaluated in the following order (from lowest precedence to highest):

1. Assignment operators (= += -= *= /= %=)
2. Conditional (?:)
3. Logical or (||)
4. Logical and (&&)
5. Equality (== !=)
6. Relational (< <= > >=)
7. Addition/subtraction (+ -)
8. Multiply/divide/modulus (* / %)
9. Parentheses (())

Objects and properties

JavaScript is an object-oriented language. An object is a custom data type that can combine data with functions to act upon it. The data items in an object are its *properties*, and the functions are its *methods*.

Properties can be numbers, strings, or even other objects. Each property has a name associated with it, which you must use to refer to that property.

For example, any variable that contains a string in JavaScript is actually a String object. The String object has a single property called length, which can be referred as

len = address.length;

or

len = address["length"];

Methods

Methods are simply functions that have been linked to an object and work on the properties of that object.

As an example, the built-in string object includes some methods to work with strings. One of them, toLowerCase(), converts the string to all lowercase.

To call a method, you use a period to divide the string name and the method name, as with properties. However, because a method is a function, you must include parentheses for the parameters. The following JavaScript code demonstrates the use of this method:

var text = "THIS IS UPPERCASE";

var ltext = text.toLowerCase();

The toLowerCase() method does not modify the string itself; instead, it returns a lowercase version.

Built in Objects

Built-in objects include string objects, the Date object, and the Math object. They are referred to as built-in because they really do not have anything to do with Web pages, HTML, URLs, the current browser environment, or anything visual.

String Objects

Any variable whose value is a string is actually a string object. Literal strings such as "HelloWorld" are also string objects.

The following JavaScript commands output the text "Hello!" in large, blinking, bold letters

```
var sample = "Hello!";
var sampleBig = sample.big();
var sampleBlink = sampleBig.blink();
var sampleBold = sampleBlink.bold();
document.write(sampleBold);
```

The following text displays the same word but as a hypertext link to the file http://some.domain/some/file.html.

```
var sample = "Hello!";
sample = sample.link("http://some.domain/file.html");
document.write(sample);
```

Because these methods return strings, you can also string together a series of methods and rewrite the first example as

```
var sample = "Hello!";
document.write(sample.big().blink().bold());
```

substring and Case methods

```
var sample = "tEsT";
var newSample = sample.substring(0,1).toUpperCase() +
       sample.substring(1,sample.length).toLowerCase();
```

The Math Object

Where the string object enables you to work with text literals, the Math object provides methods and properties to move beyond the simple arithmetic manipulations offered by the arithmetic operators.

```
document.write(Math.E,"<BR>");
document.write(Math.PI,"<BR>");
document.write(Math.abs(-5.23),"<BR>");
document.write(Math.asin(0.3),"<BR>");
document.write(Math.sin(Math.PI/6),"<BR>");
document.write(Math.floor(2.6),"<BR>");
document.write(Math.round(2.6),"<BR>");
document.write(Math.exp(1),"<BR>");
document.write(Math.log(10),"<BR>");
document.write(Math.max(2,-4),"<BR>");
document.write(Math.pow(2,8),"<BR>");
document.write(Math.sqrt(4),"<BR>");
document.write(Math.random(),"<BR>");
```

The Date Object

You can create a Date object any time you need to store a date, and use the Date object's methods to work with the date.

Strangely, the Date object has no properties. To set or obtain values from a Date object, you must use the methods described below.

Creating a Date Object

You can create a Date object using the new keyword.

```
birthday = new Date();
birthday = new Date("June 20, 1996 08:00:00");
birthday = new Date(6, 20, 96);
birthday = new Date(6, 20, 96, 8, 0, 0);
```

Setting Date Values

Methods to set components of a Date object to values:

•setDate() sets the day of the month.

•setMonth() JavaScript numbers the months from 0 to 11.

•setYear() sets the year.

•setTime() the number of milliseconds since January 1st, 1970.

•setHours(), setMinutes(), and setSeconds() set the time.

Getting Date Values

You can use the get methods to get values from a Date object.

•getDate() gets the day of the month.

•getMonth() gets the month.

•getYear() gets the year.

•getTime() the number of milliseconds since January 1st, 1970.

•getHours(), getMinutes(), and getSeconds() get the time.

The window Object

The window object is at the top of the object hierarchy. A window object exists for each open browser window. The properties of this object describe the document in the window and provide information about the window. Three of the window object's properties are child objects:

•The location object stores the location (URL) that is displayed in the window.

•The document object holds the Web page itself.

•The history object contains a list of sites visited before and after the current site.

You can, for example, write text to status line using the status property of the window object.

```
window.status="This is status line text!";
```

The window.open() method enables you to open a new browser window. A typical statement to open a new window looks like this:

```
WindowName=window.open("URL", "WindowName", "Feature List");
```

SmallWin = window.open("","small","width=100,height=120,toolbar=0,status=0");

window.close() method closes a window. E.g. SmallWin.close();

Using Timeouts

These are handy for periodically updating a Web page or for delaying a message or function. You begin a timeout with the setTimeout() method.

```
ident=window.setTimeout("alert('Time is up!');",10000);
```

Before a timeout has elapsed, you can stop it with the clearTimeout() method, specifying the identifier of the timeout to stop:

```
window.clearTimeout(ident);
```

The location Object

The location object is a property of the window object. It contains information about the current URL being displayed by the window.

You can read the location of the current window by using the properties listed previously. In addition, you can send the user to a new URL by changing the location object's properties. For example,

window.location.href="http://www.domain.com/path"

location.reload() reloads the current document; this is the same as the reload button on Netscape's toolbar.

The document Object

This object represents the contents of the current HTML Web page. This object includes a wide variety of attributes.

bgColor is the background color.

fgColor is the foreground (text) color.

linkColor is the color used for nonvisited links.

vlinkColor is the color for visited links.

document.write() method prints text as part of the HTML page in a document window.

The history Object

This object holds information about the URLs that have been visited before and after the current one, and it includes methods to go to previous or next locations.

*** The saying "*You can't change history*" also applies to JavaScript.

The history object has one property: **length.**

history.go() goes to a specified location in the history list. You can specify a positive number to go forward, a negative number to go back, or a string to be searched for in the history list. There is also **history.back()** and **history.forward().**

**You can't see the URLs in the history list!!!*

The form Object

Each form in your HTML page is represented in JavaScript by a form object. The form object has the same name as the NAME attribute in the <FORM> tag you used to define it.

The most important property of the form object is the *elements* array, which contains an object for each of the form elements.

For example, these expressions both refer to the first element in the order form, the name1 text field:

document.order.elements[0]

document.order.name1

length is the number of elements in the form.

Properties and methods of fields in a form

name is the name given to the field.

defaultValue is the default value; (This is a read-only property.)

value is the current value.

checked is checked status for a checkbox or a radio button.

length is the number of radio buttons in the group.

selectedIndex returns the index value of the currently selected item.

Methods:

focus() sets the focus to the field. This positions the cursor in the field and makes it the "current" field.

blur() is the opposite; it removes the focus from the field.

select() selects the text in the field, just as a user can do with the mouse.

click() acts as if the user clicked on the field.

Events in JavaScript

Events provide the basis of interacting with the browser window and the currently loaded document.
Events are triggered in the browser primarily by user actions, including finishing loading a page, entering data in a form, and clicking on form buttons.

You can use JavaScript to respond to these events. For example, you can have custom messages displayed in the status line (or somewhere else on the page) as the user moves the mouse over links. You can also update fields in a form whenever another field changes.

Event	***Description***
blur	Occurs when input focus is removed from a form element (when the user clicks outside a field)
click	Occurs when the user clicks on a link or form element
change	Occurs when the value of a form field is changed by the user
focus	Occurs when input focus is given to a form element
load	Occurs when a page is loaded into Navigator
mouseover	Occurs when the user moves the pointer over a hypertext link
select	Occurs when the user selects a form element's field
submit	Occurs when a form is submitted (i.e. when the user clicks on a submit button)
unload	Occurs when the user leaves a page

Event Handlers

In order to take advantage of events in JavaScript, it is necessary to use event handlers.

Event handlers are scripts, in the form of attributes of specific HTML tags.

The event handlers you write are executed when the specified events occur. The basic format of an event handler is:
<HTML_TAG OTHER_ATTRIBUTES eventHandler="JavaScript Program">

E.g.
<INPUT TYPE="text" onChange="checkField(this)">

<BODY onLoad="alert('Welcome to my page!');"
onUnload="alert('Goodbye! Sorry to see you go!');">

Object	*Event Handlers Available*
Selection List	onBlur, onChange, onFocus
Text Element	onBlur, onChange, onFocus, onSelect
Textarea Element	onBlur, onChange, onFocus, onSelect
Button Element	onClick
Checkbox	onClick
Radio Button	onClick
Hypertext Link	onClick, onMouseOver
Reset Button	onClick
Submit Button	onClick
Document	onLoad, onUnload
Window	onLoad, onUnload
Form	onSubmit

Emulating Events

In addition to event handlers, it is possible to emulate events. This can prove particularly useful to submit a form without requiring the user to click on a submit button or to force the input focus into a particular form field based on user actions.

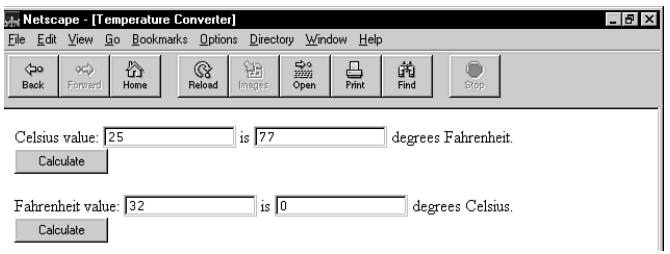
The following list outlines the event methods available in JavaScript.

- blur()
- click()
- focus()
- select()
- submit()

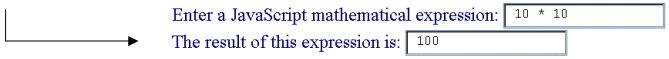
INFINITE RECURSION WARNING!!!

```
<HEAD>
     <TITLE>Temperature Converter</TITLE>
     <SCRIPT LANGUAGE="JavaScript">
     function c2f(form) {
          celsius = form.celsius.value;
          form.fahrenheit.value = (celsius*1.8)+32;
     }
     function f2c(form) {
          fahrenheit = form.fahrenheit.value;
          form.celsius.value = (fahrenheit-32)/1.8;
     }
     </SCRIPT>
 </HEAD>
```

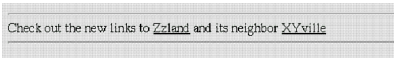
```
<BODY>
 <FORM>
   Celsius value
  <INPUT TYPE="text" NAME="celsius" SIZE=15>  is
  <INPUT TYPE="text" NAME="fahrenheit" SIZE=15> degrees Fahrenheit.<br>
  <INPUT TYPE="button" VALUE="Calculate" ONCLICK="c2f(this.form)"><br>
 </FORM>
 <FORM>
   Fahrenheit value
   <INPUT TYPE="text" NAME="fahrenheit" SIZE=15>  is
   <INPUT TYPE="text" NAME="celsius" SIZE=15> degrees Celsius.   <br>
   <INPUT TYPE="button" VALUE="Calculate" ONCLICK="f2c(this.form)"> <br>
 </FORM>
</BODY>
```



```
<HTML> <HEAD>
 <SCRIPT LANGUAGE="JavaScript">
<!-- HIDE FROM OTHER BROWSERS
function calculate(form) {
  form.results.value = eval(form.entry.value);
}
function getExp(form) {
  form.entry.blur();
  form.entry.value = prompt("Please enter a JavaScript mathematical expression","");
   calculate(form);
}
// STOP HIDING FROM OTHER BROWSERS --> </SCRIPT>
</HEAD>
<BODY>
<FORM METHOD=POST> Enter a JavaScript mathematical expression:
<INPUT TYPE=text NAME="entry" VALUE="" onFocus="getExp(this.form);">
 <BR>
The result of this expression is:
<INPUT TYPE=text NAME="results" VALUE="" onFocus="this.blur();">
</FORM> </BODY> </HTML>
```



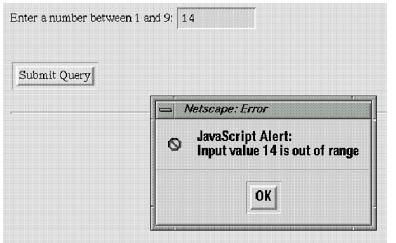

```
<HTML> <HEAD> <TITLE>A Nationalist JavaScript Page</TITLE>
  <SCRIPT LANGUAGE="JavaScript"> <!--
    var UserWarned=false;
    function warnthem(lnk) {
      var theirhost = lnk.hostname;     //get hostname of link
      var domain = "", lastdot = 0, len = 0;
      len = theirhost.length;      // string length of hostname
      lastdot = theirhost.lastIndexOf(".");    //find last dot
      domain = theirhost.substring(lastdot+1, len);
      if ( domain == "zz" && !UserWarned) {
         alert("Country zz only has 1200 baud modems");
         UserWarned=true;
      }
    } -->
    </SCRIPT> </HEAD>
    <BODY>
    <HR> Check out the new links to
 <A HREF="http://home.std.zz" onMouseOver="warnthem(this)">Zzland</A>
      and its neighbor
 <A HREF="http://home.xyz.xy" onMouseOver="warnthem(this)">XYville</A>
    <HR>
    </BODY>   </HTML>
```



Validating Forms

- One of the most important uses of JavaScript is in validating forms.
- JavaScript can be used to validate forms on the client site.
- This reduces the workload of the server and also speeds up the process of submitting forms.
- Form validation is generally implemented with the onSubmit event handler.

```
<HTML> <HEAD> <TITLE>A Simple Form Validation Example</TITLE>
 <SCRIPT LANGUAGE="JavaScript"> <!--
  function checkit() {    // submit validation function
    var strval = document.myform.mytext.value;
    var intval = parseInt(strval);     //convert to integer
    if ( 0 < intval && intval < 10 ) {   // input ok?
      return( true );                    // allow submit
    } else {                             // input bad - tell user
       alert("Input value " + strval + " is out of range");
       return( false );                  // forbid submit
    }
  }
  --> </SCRIPT> </HEAD>
  <BODY>
  <HR>
  <FORM NAME="myform" METHOD="post"
      ACTION="mailto:me@myhost.com"
         onSubmit="checkit()">
    <P>Enter a number between 1 and 9:
    <INPUT TYPE="text" NAME="mytext" VALUE="1" SIZE="10"></P>
    <BR><INPUT TYPE="submit">
  </FORM>
  <HR>
  </BODY> </HTML>
```



Loops

Most programming relies on the capability to repeat a number of lines of program code based on a condition or a counter. This is achieved by using loops.

The **for** keyword is the first tool to look at for creating loops.

```
for (var = 1; var < 10; var++) {
   JavaScript Statements to be repeated
}
```

There are three parameters to the for loop, separated by semicolons:

•The first parameter (var = 1 in the example) specifies a variable and assigns an initial value to it. This is called the *initial expression*, because it sets up the initial state for the loop.

•The second parameter (var < 10 in the example) is a condition that must remain true to keep the loop running. This is called the *condition* of the loop.

•The third parameter (var++ in the example) is a statement that executes with each iteration of the loop. This is called the *increment expression*, because it is usually used to increment the counter.

```
for (i=1; i<10; i++) {
     document.write("This is line ",i,"\n"); }
```

The **while loop**

The other keyword for loops in JavaScript is while. Unlike for loops, while loops don't necessarily use a variable to count. Instead, they execute as long as (while) a condition is true.

For example,

```
while (total < 10) { n++; total += values[n]; }
```

is equivalent to

```
for (n=0;total < 10; n++) { total += values[n]; }
```

break and **continue** statements:

The **break** command does what the name implies—it breaks out of the loop completely, even if the loop isn't complete. For instance, if you want to give students three chances to get a test question correct, you could use the break statement:

```
var answer = "";
var correct = "100";
var question = "What is 10 * 10?";
for (k = 1; k <= 3; k++) {
  answer = prompt(question,"0");
  if (answer == correct) { alert ("Correct!");
                           break; }
}
```

The **continue** statement skips the rest of the loop, but unlike **break**, it continues with the next iteration of the loop:

Arrays

Many languages support *arrays*-numbered sets of variables. For example, scores for 20 different students might be stored in a scores array. You could then refer to scores[1] for the first student's score, scores[5] for the fifth student, and so on.

You can create an array by using the Array object. For example,

```
scores = new Array(20);
```

you can then assign values for the first and tenth scores with these statements:

```
scores[0] = 50;
scores[9] = 85;
```

Note that unlike many programming languages the elements of an array does not have to be of the same type in JavaScript.

```
<html> <head> <script>
function showhouse( somehouse ) { // display properties
  for( var iter = 0; iter < 4; iter++) { // four properties
     document.write("<BR>Property "+ iter +
                    " is "+ somehouse[iter]);
  }
  document.write("<BR>");
}
</script> </head>
<body> <script>
var myhouse= new Array(4);
myhouse[0] = 5;               // rooms
myhouse[1] = "Modern";        // style
myhouse[2] = 1989;            // year_built
myhouse[3] = true;            // has_garage
showhouse(myhouse);
</script> </body> </html>
```

Defining Objects

Before creating a new object, it is necessary to define that object by outlining its properties. This is done by using a function that defines the name and properties of the function. This type of function is known as a *constructor function*. For example,

```
function house ( rms, stl, yr, garp ) {
   this.rooms = rms;    // rooms
   this.style = stl;    // architecture style
   this.yearbuilt = yr; // year constructed
   this.hasgarge = garp;// does it have a garage?
}

myhouse = new house( 5, “Modern", 1989, true);

document.write(myhouse.style);
```

for … in loop

There's a third type of loop available in JavaScript. The for...in loop is not as flexible as ordinary for or while loops; instead, it is specifically designed to perform an operation on each property of an object.

Like an ordinary for loop, this type of loop uses an index variable. (i in the example). For each iteration of the loop, the variable is set to the next property of the object. This makes it easy when you need to check or modify each of an object's properties.

```
function house ( rms, stl, yr, garp ) {
   this.rooms = rms;    // rooms
   this.style = stl;    // architecture style
   this.yearbuilt = yr; // year constructed
   this.hasgarge = garp;// does it have a garage?
}
function showany(anyobj) {  // display properties
  for( var iter in anyobj ){ // iterate over all properties
      document.write("<BR>Property " + iter +
                     " is " +  anyobj[iter]);
  }
  document.write("<BR>");
}
-----------------------------------------------------
myhouse = new house( 5, “Modern", 1989, true)
yourhouse = new house( 8, “Classic", 1922, false );
showany(myhouse)
showany(yourhouse)
```

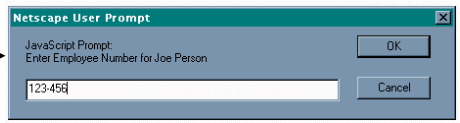
Adding Methods to Objects

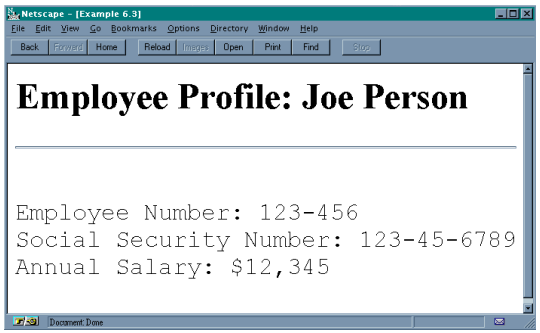
In addition to adding properties to object definitions, you can also add a method to an object definition. Because methods are essentially functions associated with an object, first you need to create a function that defines the method you want to add to your object definition.

Then you can add the method to the object as if you are adding a property to the object.

```
<HTML> <HEAD> <SCRIPT>
//DEFINE METHOD
function displayInfo() {
  document.write("<H1>Employee Profile: " + this.name +
                 "</H1><HR><PRE>");
  document.writeln("Employee Number: " + this.number);
  document.writeln("Social Security Number: " + this.socsec);
  document.writeln("Annual Salary: " + this.salary);
  document.write("</PRE>");
}
//DEFINE OBJECT
function employee() {
  this.name=prompt("Enter Employee's Name","Name");
  this.number=prompt("Enter Employee Number for " +
                        this.name,"000-000");
  this.socsec=prompt("Enter Social Security Number for " +
                         this.name,"000-00-0000");
  this.salary=prompt("Enter Annual Salary for " +
                         this.name,"$00,000");
  this.displayInfo=displayInfo;
}
</SCRIPT>  </HEAD>
```

```
<BODY>
<SCRIPT LANGUAGE="JavaScript">
  newEmployee=new employee();
  newEmployee.displayInfo();
</SCRIPT>
</BODY>
</HTML>
```





Cookies

You can use cookies to store a preference for the user, or to remember the user when they come back to your page.

Cookies are stored for the current document, and they are accessed with the *document.cookie* property. This property is a text value which can contain the following components:

name=value: A name and value, separated by the equal sign. This is the actual data stored in the cookie.

expires=date: An expiration date. If this date is not included, the cookie is erased when the user exits the browser. (For the format of the date, see the example later.)

domain=machine: The domain name for which the cookie is valid. By default, this is the domain of the current page.

path=path: The URL path for which the cookie is valid. By default, this is the current URL.

```
<HTML> <HEAD>
 <TITLE>The page that remembers your name</TITLE>
 <SCRIPT>
  if (document.cookie.substring(0,2) != "n=") {
    nam = window.prompt("Enter your name");
    document.cookie = "n=" + nam + ";";
    document.cookie +=
           "expires=Tuesday, 31-Dec-2010 23:59:00 GMT"
  }
 </SCRIPT> </HEAD>
 <BODY>
 <H1>Here is Your name</H1> <HR>
 <SCRIPT>
   indx = document.cookie.indexOf(";");
   nam = document.cookie.substring(2,indx+1);
   document.write("Hello there, ", nam);
 </SCRIPT>
 </BODY>
</HTML>
```