Introduction to JavaScript

In this lesson we will provide a brief introduction to JavaScript. We won't go into a ton of detail. There are a number of good JavaScript tutorials on the web.

What is JavaScript?

JavaScript is an *object-based interpreted* scripting language that is embedded in HTML pages. With it you can control and program elements of your own web pages on the client side without depending on CGI scripts on a server. You can go so far as to create forms, create dynamic HTML, connect to Java applets, and even perform complex mathematical operations. However, JavaScript is relatively simple to use compared to most CGI scripts and in some cases is easier to learn.

JavaScript was originally developed by Netscape and called LiveScript. In 1995, Netscape and Sun changed the name to JavaScript so that Sun could become part of the development. Just remember that JavaScript is not Java! Despite the similarity in names (and in some of the syntax) the two are entirely separate languages.

One problem with many programming languages is that you usually need to get the right compiler or interpreter to compile or run programs. The nice thing about JavaScript is that all modern browsers have support for JavaScript built in. So you probably already have everything you need on your system to start writing and running JavaScript programs.

JavaScript Basics – "Hello World" Program

JavaScript code is inserted between tags, just like normal HTML tags:

```
<SCRIPT LANGUAGE="JavaScript">
     Put your Javascript code here
</SCRIPT>
```

Some old browsers may not recognize the <SCRIPT> tag, and will attempt to interpret the JavaScript code as HTML. To prevent this, often the HTML comment tags are inserted so that the old browsers will ignore the JavaScript. However, the JavaScript interpreter is smart enough to ignore the comment tags!

```
 <SCRIPT LANGUAGE="JavaScript">
     <!—HTML Comment, JavaScript Begins Here
     Put your Javascript code here
     // JavaScript comment
     // JavaScript ends here -->
</SCRIPT>
```

The double slashes, "//" is for JavaScript comments. The JavaScript interpreter will ignore anything preceded by two slashes.

Let's jump right into a simple program that prints "Hello, World" to the screen:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
     document.write("Hello, World!");
</SCRIPT>
</BODY>
</HTML>
```

This simple program executes a single statement. Statements are ended with a semicolon. When the web page is loaded, it prints "Hello, World" to the *document object*. We'll describe the document object later. For now, all you need to know is that *document.write* is a **method**, or **function**, that prints output to the web browser.

Point to remember: JavaScript code that is entered directly into the head or body of an HTML document is executed when the web page is loaded.

Let's look at one additional method, or function, that can be used to generate output. We'll do the same thing, except display the message in an **Alert Box**:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
     alert("Hello, World!");
</SCRIPT>
</BODY>
</HTML>
```

This program calls the "alert" function, which displays a message in a message box. When this page is loaded, it will display "Hello World!" in a box and stop until the user presses "OK". The message box will have a large yellow "Alert!" icon so that you can tell the difference between a JavaScript alert box and a different one (for example, one prompting you for a password, so someone can't write a JavaScript program to spoof your password).

Variables and Basic Data Types

When you write scripts you will almost always want to work with some information, or data. You can use several different types of data:

- Numbers, such as 3.14159 or 213
- Strings, such as "Hello, World". Strings are enclosed in quotes.

- Boolean values, such as true or false.
- No value, using the special keyword *null*.

A *variable* is a fundamental construct in almost every programming language. A variable is just a storage location for data. Variables use the following syntax:

```
Identifier = Value;
```

e.g.

```
name="Kenrick";
pi=3.1415;
```

In this case, the "=" is an assignment operator. It assigns the value on the right hand side into the variable on the left hand side. This is different from a statement of mathematical equality.

You may pick almost any name you want for an identifier, but you should pick one that makes sense. Some people like to use a first letter that identifies the type; e.g. sName or sAddress for strings, nValue or nPi for numbers, etc. The restrictions on identifiers are:

- Identifiers are case-sensitive, but no spaces
- No JavaScript keywords (e.g., null, if, var, true, false)

To define a variable, use the format:

```
var identifier;                    or
var identifier=value;
```

For example what is the output of the following program?

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
     var nPi = 3.14;
     var nSomeNum;

     nSomeNum=2;
     document.write("Pi * SomeNum = " + nSomeNum*nPi);
</SCRIPT>
</BODY>
</HTML>
```

The output of this is "Pi * SomeNum = 6.28".

In this little program we added a few things aside from variables. We also used a mathematical operator and a concatenation operator.

The * symbol is multiplication, and it multiplies the value of nSomeNum by the value of nPi. Other mathematical operators we could use are:

+	addition
*	multiplication
-	subtraction
/	division
%	modulus (returns the remainder)

We also used the "+" symbol to concatenate, or join, values together. In this case we concatenated the string "Pi * SomeNum" with the value of the result. We could join together as many of these as we like.

To assign a variable a string value, we just put the string within either single or double quotation marks. If you do not use quotation marks, the JavaScript language thinks that you are assigning the value of a variable to another variable:

```
<HTML><HEAD></HEAD><BODY>
<SCRIPT LANGUAGE="JavaScript">
     var sString1;
     var sString2;
     sString1="DooFarb";
     sString2=sString1;
     document.write("String1 = " + sString1 + "<P>String2="
+ sString2);
</SCRIPT></BODY></HTML>
```

This program will output:

DooFarb
DooFarb

Because sString1 gets assigned DooFarb, and then sString2 gets the value of sString1. Both values are printed out. We used the HTML tag for a paragraph to separate the two strings.

Functions

At this point you should know enough to put together scripts that perform various math operations and can output various bits of text. If you have a commonly performed sequence of instructions, we can lump them together into a function. For example, consider the script below:

```
<HTML><HEAD></HEAD><BODY>
```

```
<SCRIPT LANGUAGE="JavaScript">
     var nResult;
     nResult = 3*3;
     document.write("3 squared is " + nResult +"<P>");
     nResult = 4*4;
     document.write("4 squared is " + nResult+"<P>");
     nResult = 5*5;
     document.write("5 squared is " + nResult+"<P>");
</SCRIPT></BODY></HTML>
```

This will output:

3 squared is 9
4 squared is 16
5 squared is 25

But this program is not very efficient; there is a lot of repeating code. We can move the repetitive code into a function. Functions are normally defined in the **<HEAD>** block of the HTML:

```
<HTML>
<HEAD>
<SCRIPT Language="JavaScript">
     function doCalculation(x) {
      var nResult;
      nResult=x*x;
      document.write(x + " squared is " + nResult + "<P>");
     }
</SCRIPT>
</HEAD>
<BODY><SCRIPT Language="JavaScript">
     doCalculation(3);
     doCalculation(4);
     doCalculation(5);
</SCRIPT></BODY></HTML>
```

This program results in the same output as the previous program. But it is organized better, in that we can make a single *function call* to invoke the calculation. The syntax for a function is:

```
 function functionName(argument1, argument2, argument3) {
     … block of JavaScript code
 }
```

Functions can also return a value. Another way we could have written this script is as follows:

```
<HTML>
```

```
<HEAD>
<SCRIPT Language="JavaScript">
      function doCalculation(x) {
            var nResult;
            nResult=x*x;
            return nResult;
      }
</SCRIPT>
</HEAD>
<BODY><SCRIPT Language="JavaScript">
      var nResult;
      nResult = doCalculation(3);
      document.write("3 squared is " + nResult + "<P>");
      nResult = doCalculation(4);
      document.write("4 squared is " + nResult + "<P>");
      document.write("5 squared is " + doCalculation(5) + "<P>");
</SCRIPT></BODY></HTML>
```

In this example, the doCalculation is returning the square as the result. This result can be assigned when the function is invoked, or used directly in another function (the write function).

Note that we have the variable "nResult" defined twice. Variables defined in functions only have local scope within their block of code (the function). So for example, if multiple people were writing different functions, they don't have to worry about if their variables have the same names and could be getting their values confused.

Control Structures

We'll probably want to make decisions and determine what instructions get executed at what time. To do that we need to have control structures. We'll introduce two of them here, the if-then statement, and loops.

If Statements

When you want to test for a single condition, use the if statement. The syntax is:

```
if (condition) {
      Statements
}
```

If the condition is true, then the statements are executed. We could also add an else statement that is executed if the condition is false:

```
if (condition) {
      Statements
```

```
} else {
     Other statements
}
```

For example:

```
nYear=1999;
if (nYear==2000) {
     document.write("The world is going to end");
} else {
     document.write("Still plenty of time left");
}
```

will print only "Still plenty of time left".

Things to note here: We needed to use "==" to compare nYear with 2000. The double equal sign is the comparison operator for equality. If we used a single equal sign, we would be assigning 2000 to nYear! *This is one of the most common bugs ever in writing computer programs*. The comparison operators we have are:

<	Less Than
>	Greater Than
>=	Greater Than or Equal to
<=	Less Than or Equal to
!=	Not equal to
==	Equal to

Finally, there are logical operators we could use:

&&	AND
||	Or
!	Not

For example:

```
nYear=2000;
bStoredWater=true;
if (nYear==2000 && bStoredWater==true) {
     document.write("You ready for Y2K!");
} else {
     document.write("Not ready yet.");
}
```

This says that you are ready for Y2K but only if the variable nYear is set to 2000 and if bStoredWater is set to true.

Loops

If you want to perform an operation many times, one way to do it is to duplicate the code repeatedly. Fortunately we have another construct to help us do this, the loop. We'll cover two types of loops, for loops and while loops:

For loops take the following syntax:

```
for (initial-condition; ending-condition; command) {
     … statements
}
```

This structure begins with the initial condition and executes it once. Then it executes the statements and the command each time, until the ending condition is met. It is normally used with numbers, as in the following example:

```
var  money=1;
var count;
for (count=1; count<30; count=count+1) {
     money=money*2;
     document.write("On day " + count + " you have "
      + money + "<BR>");
}
```

This ends with:

On day 29 you have 536870912

We can do the same thing using while loops. The format is:

```
while (condition) {
     execute JavaScript …
}
```

For example:

```
var  money=1;
var count;
count=1;
while (count<30) {
     money=money*2;
     document.write("On day " + count + " you have " +
          money + "<BR>");
     count=count+1;
}
```

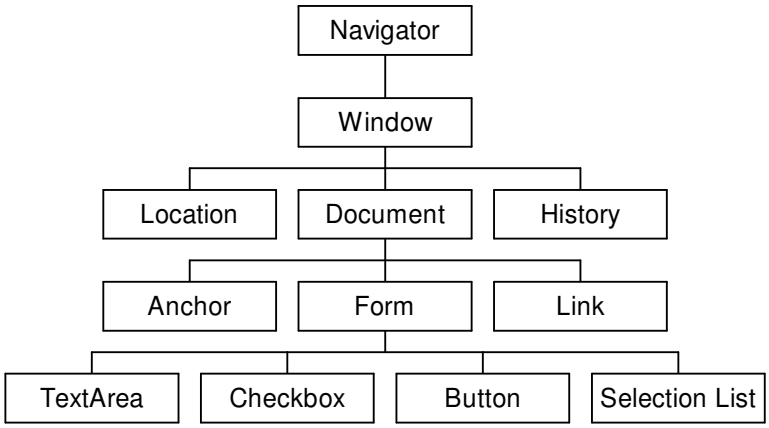
This results in the same output as the for-loop example. The two methods are equivalent and you can take your pick as to which you want to use.

Introducing Objects

Although we haven't used too many of them yet, the JavaScript language is based on objects. The "document.write" method we have been using is invoking the "document" object.

The basic object is a *window*. Everything that you do in JavaScript must take place inside the window object. Within the window you can define *documents*. You can divide the document space by using *frames*, and then inside the document you can display text and images and buttons and other form elements.

The following is a hierarchy of objects:



Objects may contain other objects, or have properties, or methods. Properties are values associated with the object. Properties are usually readable, and sometimes writable. Methods are the same as functions associated with the object, like the doCalculation function we wrote earlier.

For example, here are some properties and methods associated with the document object:

Document Properties
- forms – array of forms on the document
- location – string containing current URL
- title – string containing current Title
- bgColor – background color
- fgColor – foreground color

Document Methods
- write() – display information into the document

To reference an object or property, you need to follow the hierarchy of objects shown on the previous page. You reference a property or method using the dot syntax:

```
object.property
```

```
object.method()
```

As some examples, the following code would take us straight to the yahoo page:

```
document.location = "http://www.yahoo.com";
```

The following code would repeatedly change the background color back and forth 100 times (somewhat annoyingly!)

```
var counter;
for (counter=1; counter<100; counter=counter+1) {
     document.bgColor="red";
     document.bgColor="blue";
     document.bgColor="green";
}
```

The following close calls the close() method of the window object. This will close the browser when the page is loaded!

```
window.close();
```

As a more complex example, consider the following HTML file:

```
<HTML>
<HEAD></HEAD>
<BODY>
<FORM NAME="myForm">
     <INPUT NAME="myTextBox" value="Something">
</FORM>
<SCRIPT Language="JavaScript">
     document.write(document.myForm.myTextBox.value);
</SCRIPT></BODY></HTML>
```

In this script, we have a form that we have named "myForm" and a input box we have named "myTextBox" with the default value of "Something". We can access the form's input box by navigating down the hierarchy of objects. From the document is contained the form named myForm, and from myForm is named the textbox myTextBox. We put this together as "document.myForm.myTextBox.value" to access the value of the textbox. In this case, it will print out "Something" below the textbox.

We can also set the value of the textbox. What will the following program do?

```
<HTML>
<HEAD></HEAD>
<BODY>
<FORM NAME="myForm">
     <INPUT NAME="myTextBox">
```

```
</FORM>
<SCRIPT Language="JavaScript">
     document.myForm.myTextBox.value = "Doofarb";
</SCRIPT></BODY></HTML>
```

By combining forms with input or text boxes, you can use this as a method for providing user input.

One final example illustrates the use of the Timer function:

```
     setTimeout("code", milliseconds)
```

This method invokes the code, which must be in quotation marks, after the given number of milliseconds. For example, the following repeatedly changes the background color after 2 seconds:

```
<HTML>
<HEAD>
<SCRIPT Language="JavaScript">
     function changeColor() {
          if (document.bgColor=="#ff0000") {
               document.bgColor="#00ff00";
          } else {
               document.bgColor="#ff0000";
          }
          setTimeout("changeColor()",2000);
     }
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT Language="JavaScript">
     setTimeout("changeColor()",2000);
</SCRIPT></BODY></HTML>
```

Predefined Objects

There are a number of predefined objects in JavaScript that you can use. Here is a brief description of a few of them. To create an object we use the *new* operator:

```
     obj = new object();
```

Date object:

```
     var someDay;
     var today;
     today = new Date();       // Current date
     someDay = new Date(1999, 0, 9, 22, 30, 23)
```

```
        // Jan 9, 1999, 22:30:23

    document.write("Today="+today+"<P>");
    document.write("someDay="+someDay+"<P>");
    document.write("day:"+today.getDay() + " month:" +
        today.getMonth() + " yr: " + today.getYear());
```

Output:

> Today=Wed Sep 22 22:29:57 GMT-0700 (Pacific Daylight Time) 1999
> someDay=Sat Jan 09 22:30:23 GMT-0800 (Pacific Standard Time) 1999
> day:3 month:8 yr: 99

Finally, we can combine the previous exercise of the Timer with a form and a date. What is the output of the following script?

```
<HTML>
<HEAD>
<SCRIPT Language="JavaScript">
    function showTime() {
      var s;
      var today=new Date();
      s = today.getHours() + ":" + today.getMinutes() + ":" +
          today.getSeconds();
      document.timeForm.timeText.value = s;
      setTimeout("showTime()",1000);
    }
</SCRIPT>
</HEAD>
<BODY>
<FORM name="timeForm">
    <INPUT name=timeText value="">
</FORM>
<SCRIPT Language="JavaScript">
    showTime();
</SCRIPT></BODY></HTML>
```

We will be using the date later to compute expiry times for cookies.

Arrays

Arrays are very useful for storing repeat amounts of data. For example, if you wanted to store 1000 names, you could define 1000 different variables. But this would not be a very pleasant task. An easier way is to use an *array*, which allows you to define an arbitrary number of variables at once, and then access each variable by an index.

You can create an array with the Array() object:

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

After you define an array, you can access elements by number, enclosed in brackets []:

```
myArray[0]=”hello”;
myArray[1]=”world”;
myArray[2]=3.14159;
myArray[3]=otherVar;
```

For example:

```
var myArray=new Array();
myArray[0]=”hello”;
myArray[1]=3.14;
var count;
for (count=0; count<2; count=count+1) {
     document.write(myArray[count] + “<P>”);
}
```

This script will print:

Hello
3.14

Remember that arrays begin at index 0, not at index 1! A common operation is to store lists of data in arrays, and then perform other operations like sorting the array or adding or deleting elements from the arrays.

Events

Events and *event handlers* are very important for JavaScript programming. Events are mostly caused by user actions. When an event happens, an event handler is the code that reacts to the event. For example, if the user clicks on a button a Click-event occurs. If the mousepointer moves across a link a MouseOver-event occurs. There are several different events. We want our JavaScript program to react to certain events. This can be done with the help of event-handlers. A button might create a popup window when clicked.
This means the window should pop up as a reaction to a Click-event. The event-handler we need to use is called onClick. The following code shows an easy example of the event-handler onClick:

```
 <form>
<input type="button" value="Click me"
     onClick="alert(‘You just clicked me!’)">
</form>
```

When you click on this, you’ll get an alert box with the message “You just clicked me!”. Note that to use quotes within quotes, you can use single quotes inside the double quotes.

The format for events is:

```
onEventName = “javascript code”
```

JavaScript Code in the event is usually a function that you write that performs some task.

OnClick Event

Here is an example that uses a function to change colors of the background:

```
<HTML>
<HEAD>
<SCRIPT Language="JavaScript">
     function setBackground(theColor) {
          document.bgColor=theColor;
     }
</SCRIPT>
</HEAD>
<BODY>
<form>
<input type="button" value="Red"
     onClick="setBackground('#FF0000');"><br>
<input type="button" value="Green"
     onClick="setBackground('#00FF00');"><br>
<input type="button" value="Blue"
     onClick="setBackground('#0000FF');"><br>
</form>
</BODY></HTML>
```

Note that this script has no <SCRIPT> in the <BODY> of the HTML. This means that no code is excuted until the user presses a button.

Here are some more commonly used examples of the onClick event:

```
<input type=”button” value=”Yahoo”
onClick=”document.location=’http://www.yahoo.com’”>
```

This jumps to Yahoo when the link is clicked.

```
<HTML>
<HEAD>
<SCRIPT Language="JavaScript">
     function calcSalesTax() {
          document.SalesForm.SalesTax.value = 0.0725 *
               document.SalesForm.SalesCost.value;
     }
</SCRIPT>
</HEAD>
```

```
<BODY>
<form name=SalesForm>
Enter cost: <input name=SalesCost value=""><br>
The tax is: <input name=SalesTax value=""><br>
<input type="button" value="Calculate"
onClick="calcSalesTax();"><br>
</form>
</BODY></HTML>
```

This program allows the user to enter a value into the SalesCost box, and upon clicking "Calculate" will compute the amount of sales tax and display it in the SalesTax box.

Another common usage of the onClick event is to verify if all of the textboxes that may be required have text entered.

OnChange Event

The OnChange event tells us when a form element has changed. This is often used with text boxes, check boxes, or selection lists, to perform some action once a selection is made:

```
<HTML>
<HEAD>
<SCRIPT Language="JavaScript">
     function doSelect(index) {
          if (index==0) {
               document.location="http://www.yahoo.com";
          } else if (index==1) {
               document.location="http://www.google.com";
          } else {
               document.location="http://www.live.com";
          }
     }
</SCRIPT>
</HEAD>
<BODY>
<form name=myForm>
<select name=mySelection
onChange="doSelect(document.myForm.mySelection.selectedInde
x);">
   <option>Yahoo
   <option>Google
   <option>Live
</select>
```

```
</form>
</BODY></HTML>
```

The above script will change the browser to the selected search engine when the value is changes (i.e. if you select the current selection nothing will happen).

mouseOver, mouseOut events

The mouseOver and mouseOut events indicate when the mouse has moved over an anchored object or has moved off the object. The following code adds a message to the status bar of your browser (in the bottom corner) when you move on top of the text box:

```
<a href="#"
 onMouseOver="window.status='You moved the mouse here'; return
true;"
 onMouseOut="window.status='You moved the mouse off'; return
true;">
Move your mouse over here!</a>
```

When adding text to the status bar, we need to add "return true" to tell the browser to display the text.

You most commonly see mouseOver and mouseOut events with images, so that the image changes when the mouse is placed over the image.

By now, you should have a grasp of the kinds of things you can do with JavaScript. For more information, such as a more complete list of events, methods, and properties, see the web. In the next lesson we'll look at loading images, getting feedback from users, and using cookies.