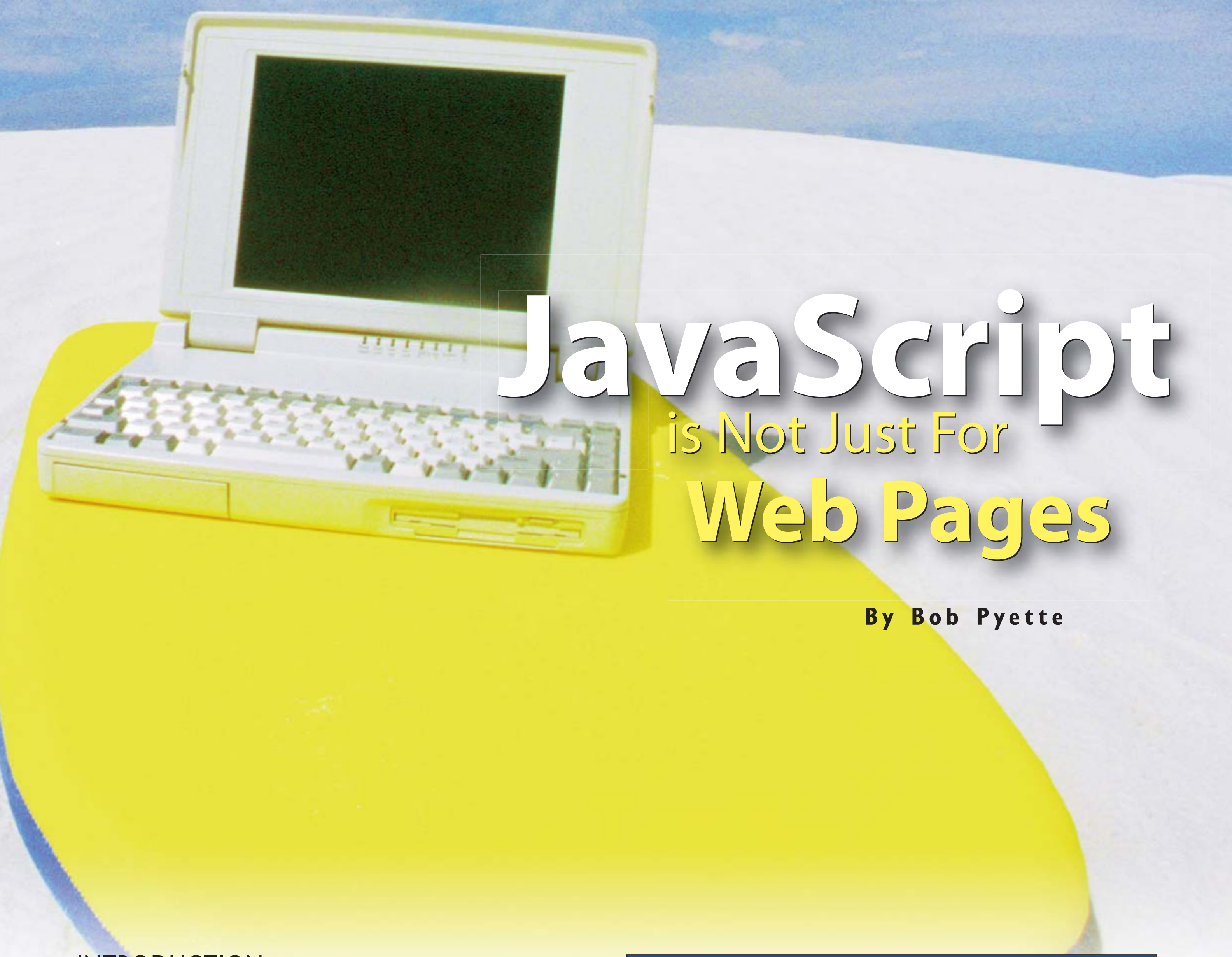

JavaScript is Not Just For Web Pages

By Bob Pyette

INTRODUCTION

There is nothing like a good scripting language that is when it comes to a clean, easy-to-use, high-level toolkit for both the programmer and the non-programmer alike. Generally, scripting languages are easier and faster to code in than the more structured and compiled languages such as C++ and Java, yet they still lend themselves to a wide range of applications.

JavaScript is Netscape's scripting language that is often used in web-based development. It was originally designed to add interactivity to HTML pages you see via your web browser. Similar to IBM's REXX and other scripting languages, it provides support for variables, language constructs, such as "if" statements, "for" and "while" loops, and many other scripting elements. In 1997, the ECMA international standards body standardized the core portion of the language. The result was a language, technically called ECMAScript, that looks and feels like JavaScript without the browser-specific parts.

Rhino is an open-source implementation of JavaScript that is typically embedded into Java applications. Rhino is one component of the Mozilla project. Developers of Java-based applications, such as job scheduling systems and identity management systems, are now embedding Rhino JavaScript to offer scripting capabilities to their users.

Adding scripting to an application brings a number of advantages. The functionality of an application is easily extended using a common and powerful tool. Scripting languages provide a natural and concise method of specifying behavior since they are programming languages, after all. Since scripts are text, they can be easily incorporated into XML documents.

So while you may not be a web developer, the chances of encountering JavaScript as part of your day-to-day job are increasing. JavaScript is not just for web pages.

This article will help you familiarize yourself with JavaScript. It introduces some basic concepts, describes how to create and run a simple JavaScript script, describes some common JavaScript elements, and provides some scripting examples on how to use these elements. Note that the term "JavaScript" is used throughout this article to refer to the JavaScript scripting language in general.

FIGURE 1: BASIC JAVASCRIPT IN AN HTML DOCUMENT

```
<html>
<script language="JavaScript">
// This is my first JavaScript script
MyName ="Bob";
document.write("Hello " + MyName);
</script>
</html>
```

JAVA VS. JAVASCRIPT

Many users fear the name “JavaScript” because they think it is the same as Java. When you say JavaScript can be used to handle users’ requirements, often the reaction is “But we’re not programmers.” Java and JavaScript are two completely different languages. It is like comparing REXX to C++. Java, developed by Sun Microsystems, is a powerful and very complex programming language. JavaScript, on the other hand, is a powerful scripting language and can be thought of as a lightweight programming language.

JavaScript, unlike Java, is an interpreted language which means it does not need to be compiled. A JavaScript program, normally referred to as a JavaScript script, is lines of executable computer code.

CREATING AND RUNNING A SIMPLE SCRIPT

Since you may be writing and running JavaScript scripts from a variety of your own applications, this first example makes use of a general-purpose technique.

The simplest way to create and run a simple JavaScript script is to use a text editor on your PC, such as Notepad, code the script, save it with a .htm extension (not with a .txt extension), and then double-click or open the file to run it. Although JavaScript statements are not HTML, they can be included within an HTML document.

FIGURE 1 shows a simple JavaScript script created using NotePad and stored as a .htm file.

The actual code for the script is stored between the <script> and </script> tags. The <script> tag identifies the scripting language as JavaScript. The <html> and </html> tags indicate this is an HTML document. The script starts with a comment line indicated by //. In the script, a variable called MyName is assigned the value "Bob". A method called document.write outputs the string "Hello” followed by the value of MyName. When this script opens in your web browser, the result is the string "Hello Bob".

If you are using JavaScript within your own application, the application should generate the tags it needs. Some applications may even include their own JavaScript editors, allowing you to easily imbed built-in functions and other language elements, and test and debug your scripts prior to implementation.

SOME GENERAL NOTES

JavaScript is case-sensitive. You will need to get used to the fact that *NAME, Name*, and *name* are all different. Watch your capitalization when referring to variables, objects, and functions.

Out of habit, some users will end each JavaScript statement with a semicolon. In general, semicolons are optional. However, they are required if you put more than one statement on a single line.

JavaScript ignores spaces. You can add white space (i.e. blank lines, indents, spaces between operators, etc.) to your script to make it more readable.

BASIC ELEMENTS OF JAVASCRIPT

This section introduces some basic elements of JavaScript, and includes some examples of JavaScript code.

FIGURE 2: AUTOMATIC CONVERSION OF DATA TYPES

```
Answer = 36;
Fact = “Number of years since man first landed on the moon is “ + Answer;
```

COMMENTS

You can add comments to your JavaScript script. A single-line comment starts with // and ends at the end of the line. A comment that spans multiple lines starts with /* and ends with */. You can include any number of comment lines between the comment delimeters.

Data Types

The basic JavaScript data types are:

- Numbers, such as 293 or 3.14. JavaScript supports both integers and floating-point numbers.
- Boolean (or logical) values, which can be either true or false.
- Strings, such as "Hello world".
- null (or empty) value, represented by the keyword null.

JavaScript is a loosely typed language. This means you do not have to specify the data type of a variable when you define it, and data types are converted automatically as needed during script execution. For example, in FIGURE 2, the variable Answer is first assigned the numeric value of 36. Then an expression concatenates a string with this number. JavaScript automatically converts the number to a string.

VARIABLES

You can use variables to store and manipulate values in a script. You give variables names by which you refer to them. Variable names must begin with a letter or the underscore character. Subsequent characters can also include the digits 0 – 9. Some examples of valid names are CaMeL, _total, jan31, and APPLstart.

You can define a variable by simply assigning a value to it or by using the keyword var. For example, age = 65 and var age = 65; each assign the number 65 to a variable called age.

There are many functions, or methods, you can use to work with variables. Some examples will follow later in this article.

LITERALS

You use literals to represent fixed values. JavaScript supports the following types of literals:

- Integers, such as 47 or -852.
- Floating-point literals, such as 1.2345, -5.2E3, or 2E-5.
- Boolean literals, either true or false.
- String literals, such as "coconut", 'One small step for man', or "123".

To use special characters in strings, such as a quotation mark ("), an ampersand (&), or a backslash (\), you must "escape" the special character by preceding the character with a backslash. For example, to assign the file path "c:\temp" to a variable called Mydir, use the following:

```
Mydir="C:\\temp"
```

EXPRESSIONS

An expression is a combination of literals, variables, and operators that evaluates to a single value. An arithmetic expression evaluates to a number; a string expression evaluates to a character string; a logical expression evaluates to either true or false.

OPERATORS

JavaScript supports many different types of operators that you can use in your expressions. Some of the more common operators are listed below.

The comparison operators are: == (is equal to), != (is not equal to), < (less than), <= (less than or equal to), > (greater than), >= (greater than or equal to).

Remember to use = when assigning a value to a variable (e.g. MyName = "Bob"), and = = when comparing values (e.g. if MyName == "Bob" …). Confusing these two is a common mistake in JavaScript programming.

You can use the following logical operators: || (or), && (and), ! (not)

In addition to the standard arithmetic operators (+, -, *, /), you can use % for modulus (division remainder), ++ (increment by 1), and -- (decrement by 1). You can also use + as a string operator to concatenate strings and variables.

CONDITIONAL STATEMENTS

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

- If statement – includes a condition and an action to take if the condition is true.
- If…else statement – identifies the actions to take based on a condition being true and the condition being false.

If you want to use more than one statement after an if or else statement, you must enclose the statements in curly braces { }.

FIGURE 3 shows an example of conditional statements. If the variable Latejob has a value of "payroll" and the variable Hour is greater than 10, the Notify variable is assigned the value 'fredbloggs@mycompany.com' and the Severity variable is set to 2. Otherwise, Severity is set to 4.

You can also use a conditional operator to assign a value to a variable based on some condition. Basically, it is a short alternative to coding a simple if…else statement where a variable is assigned different values. The syntax is:

```
variable = (condition) ? (true action) : (false action)
```

FIGURE 4 shows two approaches for assigning a variable different values based on a condition. If the remainder when MyNumber is divided by 2 is equal to 0, then assign the string "Even" to the variable named MyType. Otherwise, assign "Odd" to the variable named MyType.

JavaScript also supports the switch statement, which allows a script to evaluate an expression and attempt to match the expression's value to a case label. This allows you to combine several tests of the same variable or expression into a single block of statements.

FIGURE 3: CONDITIONAL EXPRESSION USING IF ...ELSE

```
if (LateJob == "payroll" && Hour > 10)
 {
   Notify = 'fredbloggs@mycompany.com';
   Severity = 2;
 }
 else
   Severity = 3;
```

FIGURE 4: TWO APPROACHES TO AN IF CONDITION

```
//One approach
if (MyNumber%2==0) Mytype = "Even";
 else Mytype = "Odd"

//Another approach
MyType = (MyNumber%2==0) ? "Even" : "Odd";
```

FIGURE 5: COMBINING SEVERAL TESTS WITH SWITCH

```
switch (DayOfWeekNumber)
{
case 0:
 Start_batch = "10AM";
 Late_end = "5AM tomorrow";
 break;
case 6:
 Start_batch="2PM";
 Late_end = "9AM tomorrow";
 break;
default:
 Start_batch="4PM";
 Late_end = "6AM tomorrow";
}
```

In FIGURE 5, the expression DayOfWeekNumber is evaluated once. The value of this variable is compared with the values for each "case" in the structure. If there is a match then the block of code associated with that case is executed. Use the "break" statement to prevent the code from running into the next case automatically. If there is no match to a case, the "default" code applies. For example, if DayOfWeekNumber = 6, control passes to "case 6" where the Start_batch variable is set to "2PM" and the Late_end variable is set to "9AM tomorrow".

LOOPING

To run the same block of code multiple times, you can use looping statements in your code. In JavaScript, there are the following looping statements:

- while—loop while a condition is true.
- do while—loop through a block of code once, and then repeat the loop while a condition is true. The condition is tested at the end of the loop rather than at the beginning.
- for—loop for a specified number of times. The "for" loop typically uses a variable (i.e. a counter) to keep track of how many times the loop has executed, and it stops when the counter reaches a certain number.

A simple example of a "for" loop is shown in FIGURE 6. The variable pyramid is initialized to 0. In the "for" statement, the initial expression is i=1, the condition is i<10, and the increment is i++ (add 1 to i). A left brace is used to signal the beginning of a block, a right brace is used at the end of the block. All the statements between the braces are executed with each iteration of the loop. Each time through the loop, the value of the counter i increases by 1 and the value of pyramid increases by the value of i. The loop stops when the condition i < 10 is false. The result is a display of a series of 9 numbers: 1 3 6 10 and so on, where the difference between two consecutive numbers increases by 1.

Note that a shortcut you could use for incrementing the value of pyramid in this example is pyramid+=i, which takes the value of pyramid, adds i to it, and assigns the result back to pyramid.

FIGURE 6: LOOPING WITH THE FOR STATEMENT

```
pyramid = 0;
for (i=1; i<10; i++)
{
pyramid = pyramid + i;
document.write(pyramid,” “);
}
```

FIGURE 7: USING STRING METHODS AND PROPERTIES

```
Animal = “Hippopotamus”;
HowLong = Animal.length;
AllCaps = Animal.toUpperCase();
Shortform = Animal.substring(0,5);
```

OBJECTS AND METHODS

Objects allow you to combine several kinds of data (properties) and functions to act on the data (methods) into a single, convenient package. Some examples of built-in JavaScript objects are:

- Date object—used to work with dates and times.
- Math object—includes math constants and functions.
- String object—used to work with text.
- Array object—stores a set of values in a single variable name, where each value is an element of the array with an associated index number, ranging from 0 to the number of elements - 1. For example: Month[0], Month[1], …, Month[11].

Each property is basically a variable in itself, and is contained within the object. Each property can be assigned a value. You can use properties to store any type of data a variable can store. The general syntax for using a property is object.property_name. Methods are functions that are stored as properties of an object. The general syntax for using a method is object.method_name().

In FIGURE 7, a variable called Animal is assigned the value "Hippopotamus". Subsequent statements use methods and properties to work with this variable, as follows:

- HowLong has the value 12 because Animal.length resolves to the length of the string "Hippopotamus".
- AllCaps has the value "HIPPOPOTAMUS". Animal.toUpperCase() converts the value of the variable Animal to all upper-case characters.
- Shortform has the value "Hippo". Animal.substring(0,5) resolves to a string containing the 1st – 5th characters of the variable Animal. Note that the first position in a string is position 0. substring(*start,end*) starts at position *start* and includes all characters up to but not including position *end*.

SUMMARY

This article introduced the JavaScript scripting language through descriptions and examples. Although I have only scratched the surface of this powerful language, you can use many of these basic scripting elements to create useful scripts.

There are many books and web sites available where you explore the JavaScript language in more detail. Detailed reference information on the ECMAScript language specification can be found at http://www.ecma-international.org/publications/standards/Ecma-262.htm.

JavaScript is a powerful scripting language that is finding its way into many applications outside of its traditional web-based applications. JavaScript is not just for web pages anymore.

NaSPA member Bob Pyette is a Product Planner for Toronto-based Cybermation Inc. and has over 20 years' experience in the Information Technology industry. You can reach him at bpyette@cybermation.com.