Contents

Foreword

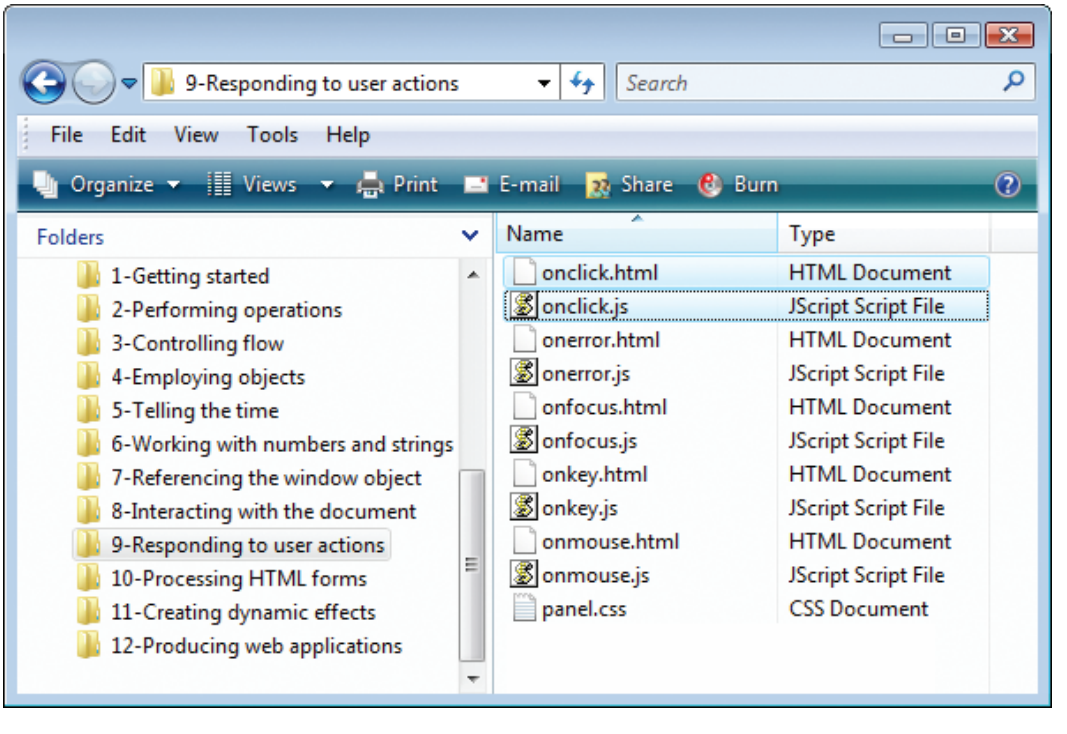
The examples in this book have been carefully prepared to demonstrate JavaScript features. You are encouraged to try out the examples on your own computer to discover the exciting possibilities offered by JavaScript. The straightforward descriptions should allow you to easily recreate the examples manually or, if you prefer, you can download an archive containing all the source code by following these simple steps:

1. Open your browser and visit our website at **http://www.ineasysteps.com**

2. Navigate to the "resource center" and choose the "Downloads" section

3. Find the "From JavaScript in easy steps, 4th edition" item in the "Source code" list, then click on the hyperlink entitled "All code examples" to download the ZIP archive

4. Extract the contents of the ZIP archive to any convenient location on your computer – for easy reference these are arranged in sub-folders whose names match each chapter title of this book. The documents are named as described in the book and are located in the appropriate chapter folder of the archive. For example, the **onclick.js** script, listed in the ninth chapter, is located in the folder named **9-Responding to user actions**



1 Getting started

Welcome to the exciting world of JavaScript. This chapter demonstrates how to incorporate script within a HTML document and introduces JavaScript functions and variables.

Introduction

JavaScript is an object-based scripting language whose interpreter is embedded inside web browser software, such as Microsoft Internet Explorer, Mozilla Firefox, Opera and Safari. This allows scripts contained in a web page to be interpreted when the page is loaded in the browser to provide functionality and dynamic effects. For security reasons JavaScript cannot read or write files, with the exception of "cookie" files that store minimal data.

Brendan Eich, creator of the JavaScript language.

Created by Brendan Eich at Netscape, JavaScript was first introduced in December 1995, and was initially named "LiveScript". It was soon renamed, however, to perhaps capitalize on the popularity of Sun Microsystem's Java programming language – although it bears little resemblance.

Before the introduction of JavaScript, web page functionality required the browser to call upon "server-side" scripts, resident on the web server, where slow response could impede performance. Calling upon "client-side" scripts, resident on the user's system, overcame the latency problem and provided a superior experience.

JavaScript quickly became very popular but a disagreement arose between Netscape and Microsoft over its licensing – so Microsoft introduced their own version named "JScript". Although similar to JavaScript, the new JScript version had extended features and some differences that remain today. Recognizing the danger of fragmentation the JavaScript language was standardized by the European Computer Manufacturer's Association (ECMA) in June 1997 as "ECMAScript". This helped to stabilize core features but the name, sounding like some kind of skin disease, is not widely used and most people will always call the language "JavaScript".

Hot tip

The Document Object Model (DOM) is a hierarchical arrangement of objects representing the currently loaded HTML document.

The JavaScript examples in this book describe three key ingredients:

- **Language basics** – illustrating the mechanics of the language syntax, keywords, operators, structure, and built-in objects
- **Web page functionality** – illustrating how to use the browser's Document Object Model (DOM) to provide user interaction and to create Dynamic HTML (DHTML) effects
- **Rich internet applications** – illustrating the latest AJAX techniques to create responsive web-based applications

JavaScript keywords

Keywords					
break	case	catch	continue	default	delete
do	else	false	finally	for	function
if	in	instanceof	new	null	return
switch	this	throw	true	try	typeof
var	void	while	with		

The words listed in the table above are all "keywords" that have special meaning in JavaScript and may not be used when choosing names in scripts. You should also avoid using any of the reserved words that are listed in the table below as they may be introduced in future versions of JavaScript.

Reserved words				
abstract	boolean	byte	char	class
const	debugger	double	enum	export
extends	final	float	goto	implements
import	int	interface	long	native
package	private	protected	public	short
static	super	synchronized	throws	transient
volatile				

Other words to avoid when choosing names in scripts are the names of JavaScript's built-in objects and browser DOM objects:

Objects (Built-in)				
Array	Date	Math	Object	String

Objects (DOM)				
window	location	history	navigator	document
images	links	forms	elements	XMLHttpRequest

Don't forget

JavaScript is a case-sensitive language where, for example, **VAR**, **Var**, and **var** are regarded as different words – of these three only **var** is a keyword.

Beware

Notice that all built-in object names begin with a capital letter and must be correctly capitalized, along with the DOM's **XMLHttpRequest** object.

Including inline script

JavaScript code can be included in a web page by adding HTML **<script> </script>** tags, to enclose the script, and the opening tag must have a **type** attribute specifying the unique MIME type of "text/javascript" – to identify the element's contents as JavaScript.

A HTML **<script>** element may also include helpful code comments. The JavaScript engine ("parser") ignores everything between /* and */ characters, allowing multi-line comments, and ignores everything between // characters and the end of a line, allowing single-line comments – like this:

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

/* This is a multi-line comment that might describe the script's
purpose and provide information about the author and date. */

// This is a single line comment that might describe a line of code.

</script>
```

Alternative text can be provided, for occasions when JavaScript support is absent or disabled, by adding **<noscript> </noscript>** HTML tags to enclose an explanatory message.

The **<script>** element can appear anywhere within the HTML document's body section to include "inline" JavaScript code, which will be executed as the browser reads down the document. Additionally inline JavaScript code can be assigned to any of the HTML event attributes, such as **onload**, **onmouseover**, etc., which will be executed each time that event gets fired by a user action.

Hot tip

MIME (Multipart Internet Mail Extension) types describe content types – **text/html** for HTML, **text/css** for style sheets, and **text/javascript** for JavaScript code.

inline.html

1. Create a HTML document and add a ** ** element to its body section, in which to write from JavaScript, and assign its **id** attribute a value of "panel"

```
<body>
        <div id="panel">        </div>
</body>
```

2. In the ** ** element, insert a **<script>** element containing inline code to write a greeting in the panel

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

// Dynamically write a text string as this page loads.
document.write( "Hello World!" ) ;

</script>
```

Don't forget

Notice how each JavaScript statement must be terminated by a semi-colon character.

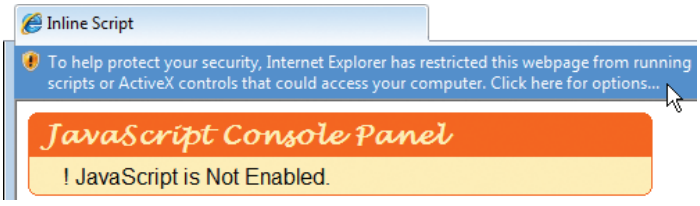
3 After the **<script>** element, insert a **<noscript>** element for alternative text when JavaScript support is absent

```
<noscript>
        <div>! JavaScript is Not Enabled.</div>
</noscript>
```

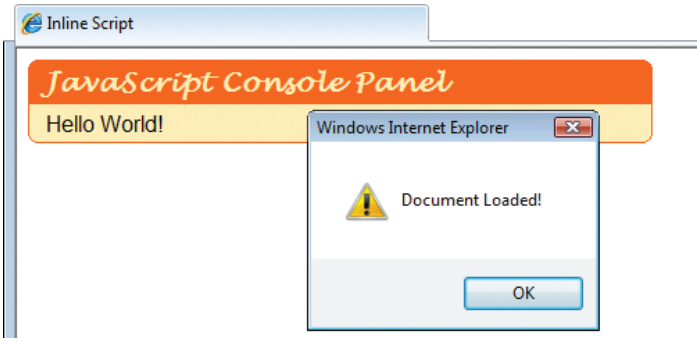
4 Now add an attribute to call a JavaScript method whenever the document gets loaded into the browser

```
// Display a message dialog after the page has loaded.
<body onload="window.alert( 'Document Loaded!' );">
```

5 Save the HTML document and disable JavaScript support in your browser, then open the web page to see the alternative text get written in the panel



6 Enable JavaScript support to see the inline script write the greeting in the panel and open a dialog box



In this example the JavaScript code calls upon the **write()** method of the **document** DOM object, to write the text string within its parentheses into the HTML document, then calls upon the **alert()** method of the **window** DOM object to display the text string specified within its parentheses on the face of a dialog box.

Hot tip

Extra HTML elements have been added around the panel and styled to give it the Web 2.0 look – the actual panel ** ** contains black text.

Beware

Text strings must be enclosed within quote characters. Nested inner strings should be surrounded by single quote characters to avoid conflict with the double quote characters that surround outer strings.

Calling head section script

Adding JavaScript functionality with numerous inline **<script>** elements throughout the body section of a HTML document is perfectly legitimate but it intrudes on the structural nature of the HTML elements and does not make for easy code maintenance. It is better to avoid inline script and, instead, place the JavaScript statements inside a "function" block within a single **<script>** element in the head section of the HTML document – between the **<head> </head>** tags.

Hot tip

A JavaScript function simply contains a set of statements to be executed whenever that function gets called.

A function block begins with the JavaScript **function** keyword, followed by a function name and trailing parentheses. These are followed by a pair of **{ }** curly brackets (braces) to enclose the statements. So its syntax looks like this:

```
function function-name( )
{
        // Statements to be executed go here.
}
```

Notice that spaces, carriage returns, and tabs are collectively known as "whitespace" and are completely ignored in JavaScript code so the function can be formatted for easy readability. Many script authors prefer to place the opening brace on the same line as the function name, but it is better to vertically align brace pairs – some statements also use braces so keeping all pairs aligned makes the code easier to read and helps prevent missing braces.

Typically a function to execute statements immediately after the HTML document has loaded in the browser is named "init" – as it performs initial tasks. This function can be called upon to execute its statements by stating its name (including the trailing parentheses) to the **onload** attribute of the **<body>** tag.

head.html

1. Create a HTML document and add a ** ** element to its body section, in which to write from JavaScript, and assign its **id** attribute a value of "panel"

```
<body>
        <div id="panel">        </div>
</body>
```

2. In the ** ** element, insert a **<noscript>** element for alternative text when JavaScript support is absent

```
<noscript>
        <div>! JavaScript is Not Enabled.</div>
</noscript>
```

3 In the head section of the document, insert a **<script>** element containing an "init" function block

```
<script type="text/javascript">
function init()
{

}
</script>
```

4 In the function block, insert a statement to write text content in the panel after the page has loaded

```
document.getElementById( "panel" ).innerHTML=
        "Hello... from a Head Section JavaScript Block!" ;
```

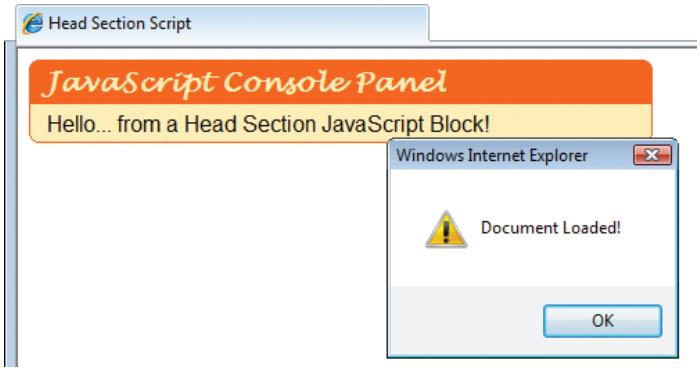
5 Next within the function block, insert a statement to display a message dialog box after the page has loaded

```
window.alert( "Document Loaded!" ) ;
```

6 Now add an attribute to the <body> tag – to call the function when the document gets loaded into the browser

```
<body onload="init()">
```

7 Save the HTML document then open it in a JavaScript-enabled browser to see the function write text content in the panel and open a dialog box



In this example the function calls upon the **getElementById()** method of the **document** DOM object, to reference the panel element then writes content by assigning a text string to its **innerHTML** property.

Don't forget

Finding a missing brace in a lengthy function block can be very difficult if brace pairs are not vertically aligned.

Beware

As JavaScript is a case-sensitive language you must be sure to correctly capitalize the **getElementById()** method and the **innerHTML** property.

Embedding external script

Where it is desirable to create a single portable HTML document its functionality can be provided by a **<script>** element within the document's head section, as in the previous example, and styling can be provided by a **<style>** element within the head section.

Where portability is of no importance greater efficiency can be achieved by creating external script and style files. For instance, all the examples in this chapter employ the same single style file to create the Web 2.0 look around the panel element. Similarly all HTML files throughout a website could therefore employ a single script file to embed JavaScript functionality in each web page. Often the JavaScript file may be referred to as a "library" because it contains a series on behavioral functions which can be called from any page on that website.

Embedding an external JavaScript file in the head section of a HTML document requires a **src** attribute be added to the usual **<script>** tag to specify the path to the script file. Where the script file is located in the same directory as the HTML document this merely needs to specify its file name and file extension – typically JavaScript files are given a ".js" file extension. For example, you can embed a local JavaScript file named "local.js" like this:

```
<script type="text/javascript" src="local.js"> </script>
```

The separation of structure (HTML), presentation (Cascading Style Sheets), and behavior (JavaScript), is recommended by the WorldWideWeb Consortium (W3C) as it makes site maintenance much simpler and each HTML document much cleaner – and so easier to validate.

Using HTML event attributes, such as **onload**, **onmouseover**, etc., to specify behavior continues to intrude on the structural nature of the HTML elements and is not in the spirit of the W3C recommendation. It is better to specify the behaviors in JavaScript code contained in an external file so the HTML document contains only structural elements, embedding behaviors and styles from elements in the head section specifying their file locations. The technique of completely separating structure and behavior in this way creates unobtrusive JavaScript, which is considered to be "best practise" and is employed throughout the rest of this book.

The W3C is the recognized body that oversees standards on the web. See the latest developments on their informative website at **www.w3.org**.

Remember to add the closing **</script>** tag. It is required even though the element is empty.

1. Create a HTML document then add a ** ** element to its body section, with an **id** attribute value of "panel", and containing alternative text for when JavaScript is absent

```
<div id="panel"><noscript>
  <div>! JavaScript is Not Enabled.</div></noscript>
</div>
```

external.html

2. In the head section of the HTML document, insert an element to embed an external JavaScript file

```
<script type="text/javascript" src="external.js"></script>
```

3. Open a plain text editor, like Windows Notepad, and add an "init" function to write content in the panel and to display a message dialog box

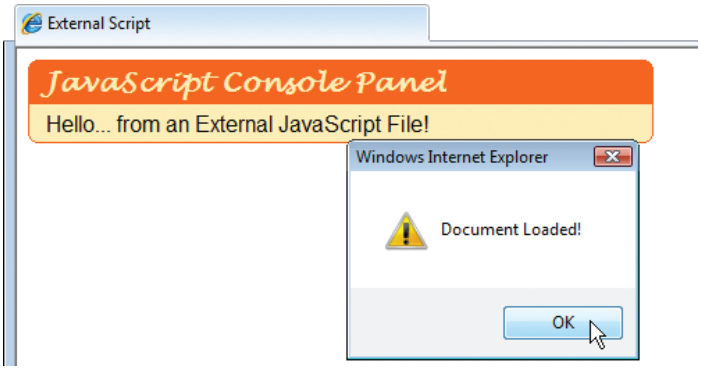
```
function init()
{
  document.getElementById( "panel" ).innerHTML=
          "Hello... from an External JavaScript File!" ;
  window.alert( "Document Loaded!" ) ;
}
```

external.js

4. After the function block, add a statement to call the function whenever the HTML document gets loaded

```
window.onload=init ;
```

5. Save the script alongside the HTML document then open the page in your browser to see the text and dialog



In this example the function name, without parentheses, is assigned to the **onload** property of the **window** DOM object.

Beware

An error will occur if you include parentheses when assigning a function to the **window.onload** property – just assign its name.

Storing data in variables

A "variable" is a container, common to every scripting and programming language, in which data can be stored and retrieved later. Unlike the "strongly typed" variables in most other languages, which must declare a particular data type they may contain, JavaScript variables are much easier to use because they are "loosely typed" – so they may contain any type of data:

Don't forget

A variable name is an alias for the value it contains – using the name in script references its stored value.

Data Type	Example	Description
boolean	**true**	A true (1) or false (0) value
number	**100** **3.25**	An integer or A floating-point number
string	**"M"** **"Hello World!"**	A single character or A string of characters, with spaces
function	**init** **fido.bark**	A user-defined function or A user-defined object method
object	**fido** **document**	A user-defined object or A built-in object

A JavaScript variable is declared using the **var** keyword followed by a space and a name of your choosing, within certain naming conventions. The variable name may comprise letters, numbers, and underscore characters, but may not contain spaces or begin with a number. Additionally you must avoid the JavaScript keywords, reserved words, and object names listed in the tables on page 9. The declaration of a variable in a script may simply create a variable to which a value can be assigned later, or may include an assignation to instantly "initialize" the variable with a value:

Hot tip

Choose meaningful names for your variables to make the script easier to understand later.

```
var myNumber;                          // Declare a variable.
myNumber = 10;                         // Initialize a variable.
var myString = "Hello World!";         // Declare and initialize a variable.
```

Multiple variables may be be declared on a single line too:

```
var i, j, k;                           // Declare 3 variables.
var num=10, char="C";                  // Declare and initialize 2 variables.
```

Upon initialization JavaScript automatically sets the variable type for the value assigned. Subsequent assignation of a different data type later in the script can be made to change the variable type. The current variable type can be revealed by the **typeof** keyword.

1 Create a HTML document that embeds an external JavaScript file and has a "panel" element

```
<script type="text/javascript" src="variable.js"></script>
<div id="panel"><noscript>
  <div>! JavaScript is Not Enabled.</div></noscript>
</div>
```

variable.html

2 Open a plain text editor, like Windows Notepad, and add a function to execute after the document has loaded

```
function init()
{

}
window.onload=init;
```

variable.js

3 In the function block, declare and intialize variables of different data types

```
var str="Text Content in JavaScript";
var num=100;
var bln=true;
var fcn=init;
var obj=document.getElementById( "panel");
```

Hot tip

The **typeof** returns a value of "undefined" for uninitialized variables.

4 Now insert statements to write the variable values and data types into the panel

```
obj.innerHTML=str + " : "+typeof str;
obj.innerHTML+="<br>"+num+" : "+typeof num;
obj.innerHTML+="<br>"+bln+" : "+typeof bln;
obj.innerHTML+="<br>init() : "+typeof fcn;
obj.innerHTML+="<br>"+obj+" : "+typeof obj;
```

Hot tip

Notice how the + operator is used here to join (concatenate) parts of a string and with += to append strings onto existing strings.

5 Save the script alongside the HTML document then open the page in your browser to see the variable data

Passing function arguments

Functions and variables are the key components of JavaScript.

A function may be called once or numerous times to execute the statements it contains. Those functions that belong to an object, such as **document.write()**, are known as "methods" – just to differentiate them from user-defined functions. Both have trailing parentheses that may accept "argument" values to be passed to the function for manipulation. For example, the text string value passed in the parentheses of the **document.write()** method that gets written into the HTML document.

Just as object functions are known as "methods" their variables are known as "properties".

The number of arguments passed to a function must match those specified within the parentheses of the function block declaration. For example, a user-defined function requiring exactly one argument looks like this:

```
function function-name ( arg )
{
  // Statements to be executed go here.
}
```

Multiple arguments can be specified as a comma-separated list:

```
function function-name ( argA, argB, argC )
{
  // Statements to be executed go here.
}
```

Multiple arguments must also be passed as a comma-separated list.

Like variable names, function names and argument names may comprise letters, numbers, and underscore characters, but may not contain spaces or begin with a number. Additionally you must avoid the JavaScript keywords, reserved words, and object names listed in the tables on page 9.

Optionally a function can return a value to the caller using the **return** keyword at the end of the function block. After a return statement has been made the script flow continues at the caller – so no further statements in the called function get executed. It is typical to return the result of manipulating passed argument values back to the caller:

```
function function-name ( argA, argB, argC )
{
  // Statements to be executed go here.

  return result ;
}
```

1 Create a HTML document that embeds an external JavaScript file and has a "panel" element

```
<script type="text/javascript" src="argument.js"></script>
<div id="panel"><noscript>
  <div>! JavaScript is Not Enabled.</div></noscript>
</div>
```

argument.html

2 Open a plain text editor and add a function to execute after the document has loaded

```
function init()
{

}
window.onload=init;
```

argument.js

3 In the function block, insert a statement that calls another user-defined function and passes it four argument values

```
document.getElementById("panel").innerHTML=
        stringify( "JavaScript", "In", "Easy", "Steps" );
```

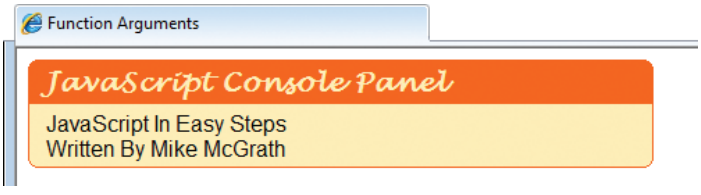
4 Next insert a second statement that also calls the user-defined function, passing it four different argument values

```
document.getElementById("panel").innerHTML+=
        stringify( "Written", "By", "Mike", "McGrath" );
```

5 Now, before the init function block, declare the function being called from the statements within the init function

```
function stringify( argA, argB, argC, argD )
{
  var str=argA+" "+argB+" "+argC+" "+argD+"<br>";
  return str;
}
```

6 Save the script alongside the HTML document then open the page in your browser to see the returned values



A function must have been declared before it can be called so function declarations should appear first in the script.

Recognizing variable scope

The extent to which a variable is accessible is called its "scope" and is determined by where the variable is declared:

- A variable declared inside a function block is only accessible to code within that same function block. This variable has "local" scope – it is only accessible locally within that function, so is known as a "local variable"
- A variable declared outside all function blocks is accessible to code within any function block. This variable has "global" scope – it is accessible globally within any function in that script so is known as a "global variable"

Local variables are generally preferable to global variables as their limited scope prevents possible accidental conflict with other variables. Global variable names must be unique throughout the entire script but local variable names only need be unique throughout their own function block – so the same variable name can be used in different functions without conflict.

scope.html

1. Create a HTML document that embeds an external JavaScript file and has a "panel" element

```
<script type="text/javascript" src="scope.js"></script>
<div id="panel"><noscript>
  <div>! JavaScript is Not Enabled.</div></noscript>
</div>
```

scope.js

2. Open a plain text editor then declare and initialize a global variable

```
var global="This is Worldwide Global news<hr>";
```

3. Add a function to execute after the document has loaded

```
function init()
{

}
window.onload=init;
```

4. In the function block, declare and initialize a local variable

```
var obj=document.getElementById( "panel" );
```

5 Next in the function block, write the value of the global variable into the panel

```
obj.innerHTML=global;
```

6 Now in the function block call two other functions, passing the value of the local variable to each one

```
us(obj);
eu(obj);
```

7 Before the "init" function block, insert a function with one argument that initializes a local variable, then appends its value and that of the global variable into the panel

```
function us(obj)
{
  var local="***This is United States Local news***<br>";
  obj.innerHTML+=local;
  obj.innerHTML+=global;
}
```

8 Before the init function block, insert another function with one argument that initializes a local variable, then appends its value and that of the global variable into the panel

```
function eu(obj)
{
  var local="---This is European Local news---<br>";
  obj.innerHTML+=local;
  obj.innerHTML+=global;
}
```

9 Save the script alongside the HTML document then open the page in your browser to see the values of the global and local variables written by the functions

Hot tip

Notice that the "local" variable names do not conflict because they are only visible within their respective function block.

Don't forget

A variable can be declared without initialization, then assigned a value later in the script to initialize it.

Summary

- JavaScript is a client-side, object-based, case-sensitive language whose interpreter is embedded in web browser software
- Variable names and function names must avoid the JavaScript keywords, reserved words, and object names
- For JavaScript code each opening HTML **<script>** tag must specify the MIME type of "text/javascript" to its **type** attribute
- Script blocks may include single-line and multi-line comments
- Each JavaScript statement must be terminated by a semi-colon
- Inline JavaScript code can be assigned to any HTML event attribute, such as **onload**, or enclosed within a **<script>** element in the document body section
- All JavaScript code is best located in an external file whose path is specified to a **src** attribute of the **<script>** tag
- Unobtrusive JavaScript places all script code in an external file and can specify a function to the **window.onload** DOM property to set behaviors when the HTML document loads
- JavaScript variables are declared using the **var** keyword and can store any data type – boolean, number, string, function, or object
- JavaScript functions are declared using the **function** keyword and the function name must have trailing parentheses, followed by a pair of { } braces enclosing statements to execute
- A function declaration may specify arguments within its trailing parentheses that must be passed from its caller
- A value can be returned to the caller using the **return** keyword
- Local variables declared inside a function are only accessible from within that function
- Global variables declared outside functions are accessible from within any function within that script