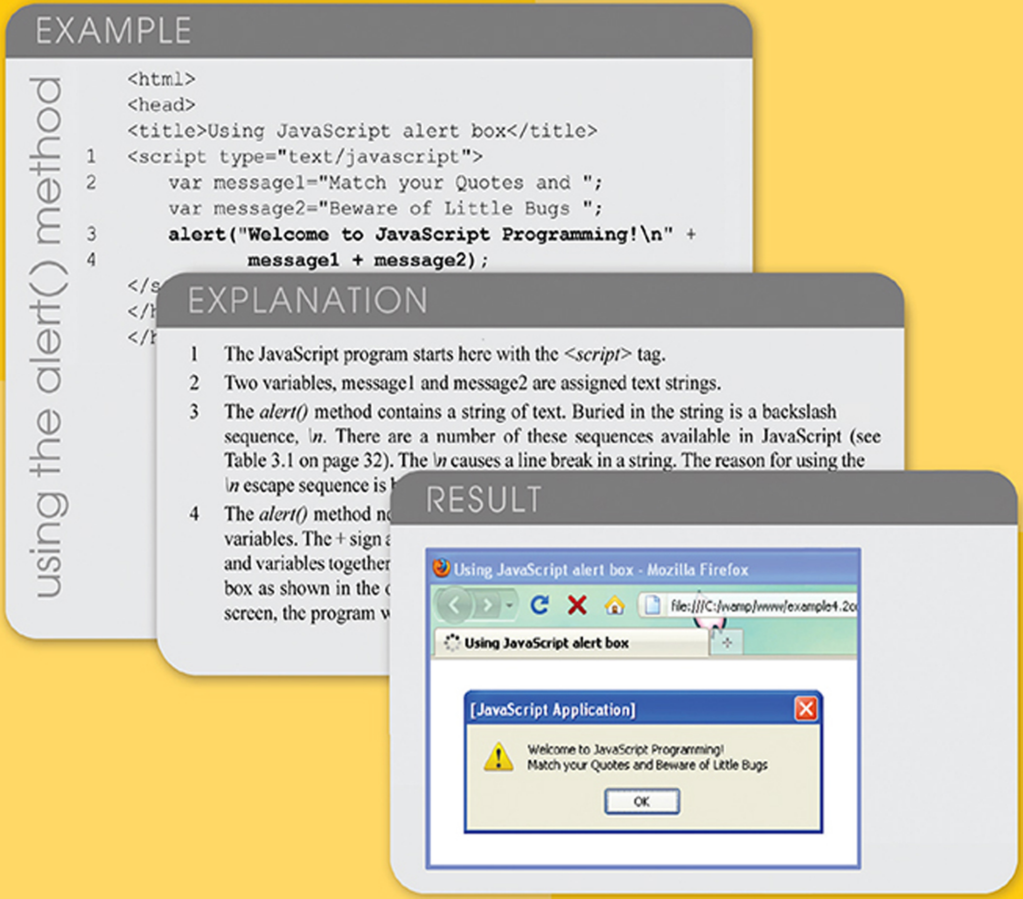
JavaScript *by* Example

Second Edition



```
    <html>
    <head>
    <title>Using JavaScript alert box</title>
1   <script type="text/javascript">
2       var message1="Match your Quotes and ";
        var message2="Beware of Little Bugs ";
3       alert("Welcome to JavaScript Programming!\n" +
4               message1 + message2);
    </s
    </h
    </h
```

1 The JavaScript program starts here with the <*script*> tag.

2 Two variables, message1 and message2 are assigned text strings.

3 The *alert()* method contains a string of text. Buried in the string is a backslash sequence, \n. There are a number of these sequences available in JavaScript (see Table 3.1 on page 32). The \n causes a line break in a string. The reason for using the \n escape sequence is

4 The *alert()* method no
variables. The + sign a
and variables together
box as shown in the
screen, the program w

Using JavaScript alert box - Mozilla Firefox

file:///C:/wamp/www/example4.2c

Using JavaScript alert box

[JavaScript Application]

Welcome to JavaScript Programming!
Match your Quotes and Beware of Little Bugs

OK

Ellie Quigley

JavaScript by Example

Second Edition

This page intentionally left blank

JavaScript by Example

Second Edition

Ellie Quigley

Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Library of Congress Cataloging-in-Publication Data

Quigley, Ellie.

JavaScript by example / Ellie Quigley.—2nd ed.
p. cm.
Includes index.
ISBN 978-0-13-705489-3 (pbk. : alk. paper)
1. JavaScript (Computer program language) I. Title.
QA76.73.J39Q54 2010
005.13'3—dc22

2010020402

ISBN-13: 978-0-13-705489-3
ISBN-10: 0-13-705489-0
4 20

Editor-in-Chief
Mark L. Taub

Managing Editor
John Fuller

Full-Service Production Manager
Julie B. Nahil

Production Editor
Dmitri Korzh
Techne Group

Copy Editor
Teresa Horton

Indexer
Potomac Indexing, LLC

Proofreader
Beth Roberts

Editorial Assistant
Kim Boedigheimer

Cover Designer
Anne Jones

Composition
Techne Group

Contents

Preface

This second edition of *JavaScript by Example* is really more than a new edition; it is a new book! So much has changed since the first edition in 2002, and now with the newfound popularity of Ajax, JavaScript is on a roll! Almost every personal computer has JavaScript installed and running and it is the most popular Web scripting language around, although it comes under different aliases, including Mocha, LiveScript, JScript, and ECMAScript. There are a lot of books out there dedicated to some aspect of the JavaScript language and if you are new to JavaScript, it would be difficult to know where to start. This book is a "one size fits all" edition, dedicated to those of you who need a balance between the technical side of the language and the fun elements, a book that addresses cross-platform issues, and a book that doesn't expect that you are already a guru before you start. This edition explains how the language works from the most basic examples to the more complex, in a progression that seemlessly leads you from example to example until you have mastered the basics all the way to the more advanced topics such as CSS, the DOM, and Ajax.

Because I am a teacher first, I found that using my first edition worked well in the classroom, but I needed more and better examples to get the results I was looking for. Many of my students have been designers but not programmers, or programmers who don't understand design. I needed a text that would accommodate both without leaving either group bored or overwhelmed. This huge effort to modernize the first edition went way beyond where I had expected or imagined. I have learned much and hope that you will enjoy sharing my efforts to make this a fun and thorough coverage of a universally popular and important Web programming language.

Acknowledgments

Many thanks go to the folks at Prentice Hall: Mark L. Taub, editor-in-chief, and the most supportive person I know; Julie Nahil, Full-Service Production Manager; John Fuller, Managing Editor; and Ann Jones, Cover Designer. Thanks also to Dmitri Korzh, Production Editor at Techne Group. Finally, a special thank you to Thomas Bishop who spent hours reviewing and sending constructive criticism that greatly improved the quality of the book; to Brendon Crawford for reviewing the manuscript; and to Elizabeth Triplett for her artwork to give the chapters a cheerful beginning.

Ellie Quigley
September, 2010

chapter

1

Introduction to JavaScript

1.1 What JavaScript Is

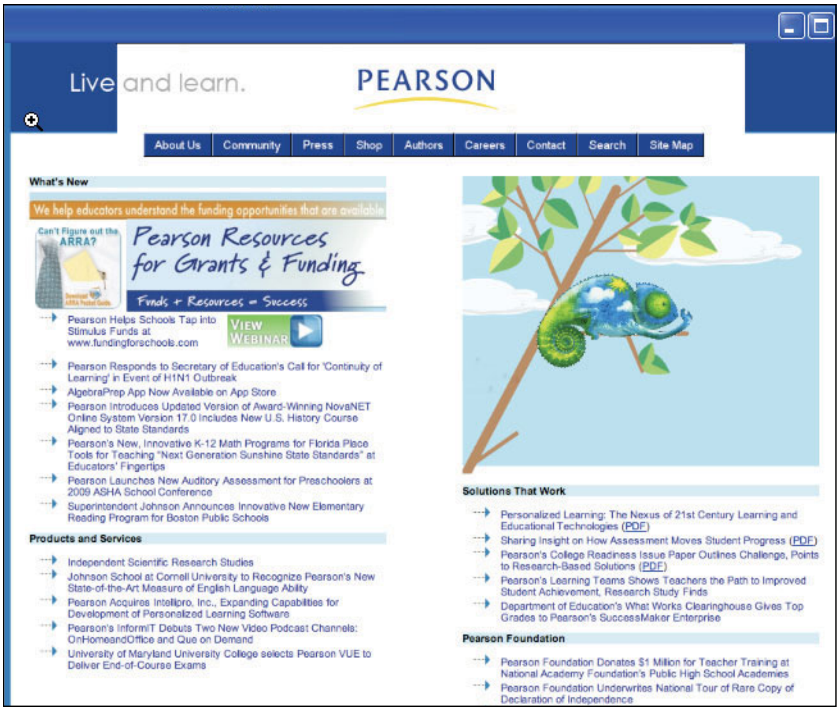
JavaScript is a popular general-purpose scripting language used to put energy and pizzaz into otherwise dead Web pages by allowing a page to interact with users and respond to events that occur on the page. JavaScript has been described as the glue that holds Web pages together.[1] It would be a hard task to find a commercial Web page, or almost any Web page, that does not contain some JavaScript code (see Figure 1.1).

JavaScript, originally called LiveScript, was developed by Brendan Eich at Netscape in 1995 and was shipped with Netscape Navigator 2.0 beta releases. JavaScript is a scripting language that gives life, hence LiveScript, to otherwise static HTML pages. It runs on most platforms and is hardware independent. JavaScript is a client-side language designed to work in the browser on your computer, not the server. It is built directly into the browser (although not restricted to browsers), Microsoft Internet Explorer and Mozilla Firefox being the most common browsers. In syntax, JavaScript is similar to C, Perl, and Java; for example, *if* statements and *while* and *for* loops are almost identical. Like Perl, it is an interpreted language, not a compiled language.

Because JavaScript is associated with a browser, it is tightly integrated with HTML. Whereas HTML is handled in the browser by its own networking library and graphics renderer, JavaScript programs are executed by a JavaScript interpreter built into the browser. When the browser requests such a page, the server sends the full content of the document, including HTML and JavaScript statements, over the network to the client. When the page loads, HTML content is read and rendered line by line until a JavaScript opening tag is read, at which time the JavaScript interpreter takes over. When the closing JavaScript tag is reached, the HTML processing continues.

1. But the creator of JavaScript, Brendan Eich, says it's even more! In his article, "Innovators of the Net: Brendan Eich and JavaScript," he says, "Calling JavaScript 'the glue that holds web pages together' is short and easy to use, but doesn't do justice to what's going on. Glue sets and hardens, but JavaScript is more dynamic than glue. It can create a reaction and make things keep going, like a catalyst."

JavaScript handled by a browser is called client-side JavaScript. Although JavaScript is used mainly as a client-side scripting language, it can also be used in contexts other than a Web browser. Netscape created server-side JavaScript to be programmed as a CGI language, such as Python or Perl, but this book will address JavaScript as it is most commonly used—running on the client side, your browser.


Live and learn.
PEARSON
About Us
Community
Press
Shop
Authors
Careers
Contact
Search
Site Map
We help educators understand the funding opportunities that are available
Can't Figure out the ARRA?
Pearson Resources for Grants & Funding
Funds + Resources = Success
VIEW WEBINAR

What's New

- Pearson Helps Schools Tap into Stimulus Funds at www.fundingforschools.com
- Pearson Responds to Secretary of Education's Call for 'Continuity of Learning' in Event of H1N1 Outbreak
- AlgebraPrep App Now Available on App Store
- Pearson Introduces Updated Version of Award-Winning NovaNET Online System Version 17.0 Includes New U.S. History Course Aligned to State Standards
- Pearson's New, Innovative K-12 Math Programs for Florida Place Tools for Teaching "Next Generation Sunshine State Standards" at Educators' Fingertips
- Pearson Launches New Auditory Assessment for Preschoolers at 2009 ASHA School Conference
- Superintendent Johnson Announces Innovative New Elementary Reading Program for Boston Public Schools

Products and Services

- Independent Scientific Research Studies
- Johnson School at Cornell University to Recognize Pearson's New State-of-the-Art Measure of English Language Ability
- Pearson Acquires Intellipro, Inc., Expanding Capabilities for Development of Personalized Learning Software
- Pearson's InformIT Debuts Two New Video Podcast Channels: OnHomeandOffice and Que on Demand
- University of Maryland University College selects Pearson VUE to Deliver End-of-Course Exams

Solutions That Work

- Personalized Learning: The Nexus of 21st Century Learning and Educational Technologies (PDF)
- Sharing Insight on How Assessment Moves Student Progress (PDF)
- Pearson's College Readiness Issue Paper Outlines Challenge, Points to Research-Based Solutions (PDF)
- Pearson's Learning Teams Shows Teachers the Path to Improved Student Achievement, Research Study Finds
- Department of Education's What Works Clearinghouse Gives Top Grades to Pearson's SuccessMaker Enterprise

Pearson Foundation

- Pearson Foundation Donates $1 Million for Teacher Training at National Academy Foundation's Public High School Academies
- Pearson Foundation Underwrites National Tour of Rare Copy of Declaration of Independence

Figure 1.1 A dynamic Web page using JavaScript to give it life. For example, if the user rolls the mouse over any of the text after the arrows, the text will become underscored links for navigation.

1.2 What JavaScript Is Not

JavaScript is not Java. "Java is to JavaScript what Car is to Carpet"[2] Well, that quote might be a little extreme, but suggests that these are two very different languages. Java was developed at Sun Microsystems. JavaScript was developed at Netscape. Java applications can be independent of a Web page, whereas JavaScript programs are embedded in a Web page and must be run in a browser window.[3] Java is a strongly typed language with strict guidelines, whereas JavaScript is loosely typed and flexible. Java data

2. From a discussion group on Usenet, also p. 4 *Beginning JavaScript with DOM Scripting and Ajax* by Christian Heilmann, APRESS, 2006.

types must be declared. JavaScript types such as variables, parameters, and function return types do not have to be declared. Java programs are compiled. JavaScript programs are interpreted by a JavaScript engine that lives in the browser.

JavaScript is not HTML, but JavaScript code can be embedded in an HTML document and is contained within HTML tags. JavaScript has its own syntax rules and expects statements to be written in a certain way. JavaScript doesn't understand HTML, but it can contain HTML content within its statements. All of this will become clear as we proceed.

JavaScript is not used to read or write the files on client machines with the exception of writing to cookies (see Chapter 16, "Cookies"). It does not let you write to or store files on the server. It does not open or close windows already opened by other applications and it cannot read from an opened Web page that came from another server.

JavaScript is object based but not strictly object oriented because it does not support the traditional mechanism for inheritance and classes found in object-oriented programming languages, such as Java and C++. The terms private, protected, and public do not apply to JavaScript methods as with Java and C++.

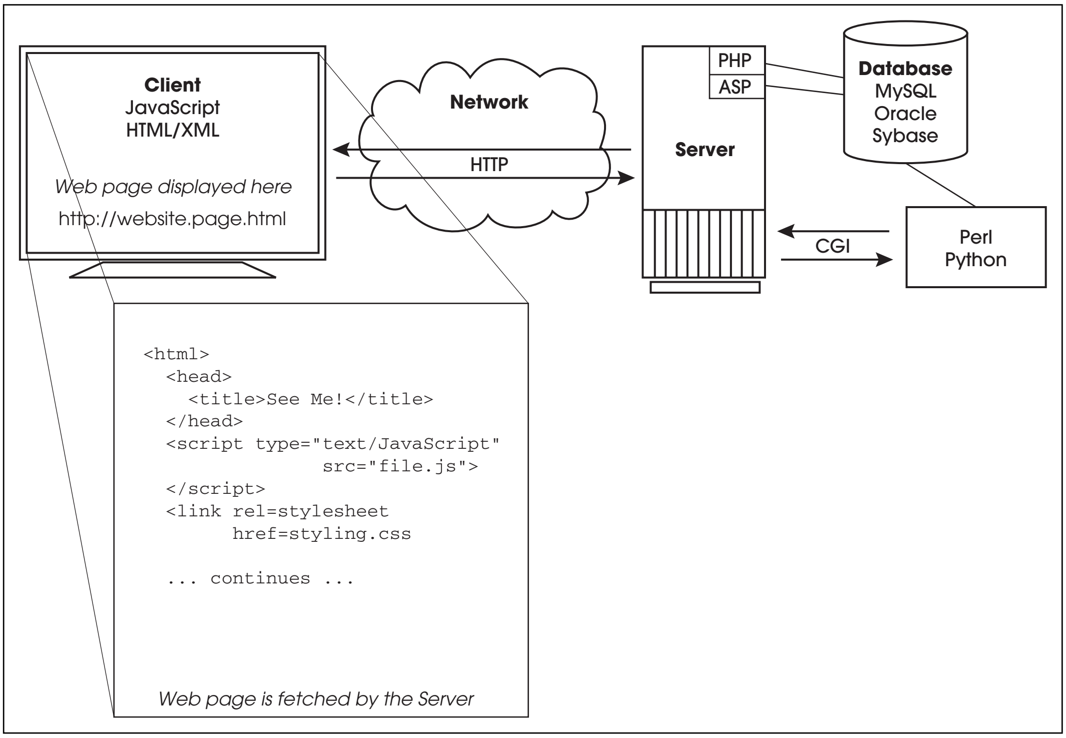
JavaScript is not the only language that can be embedded in an application. VBScript, for example, developed by Microsoft, is similar to JavaScript, but is embedded in Microsoft's Internet Explorer.

1.3 What JavaScript Is Used For

JavaScript programs are used to detect and react to user-initiated events, such as a mouse going over a link or graphic. They can improve a Web site with navigational aids, scrolling messages and rollovers, dialog boxes, dynamic images, and so forth. JavaScript lets you control the appearance of the page as the document is being parsed. Without any network transmission, it lets you validate what the user has entered into a form before submitting the form to the server. It can test to see if the user has plug-ins and send the user to another site to get the plug-ins if needed. It has string functions and supports regular expressions to check for valid e-mail addresses, Social Security numbers, credit card data, and the like. JavaScript serves as a programming language. Its core language describes such basic constructs as variables and data types, control loops, *if/else* statements, *switch* statements, functions, and objects.[4] It is used for arithmetic calculations, manipulates the date and time, and works with arrays, strings, and objects. It handles user-initiated events, sets timers, and changes content and style on the fly. JavaScript also reads and writes cookie values, and dynamically creates HTML based on the cookie value.

3. The JavaScript interpreter is normally embedded in a Web browser, but is not restricted to the browser. Servers and other applications can also use the JavaScript interpreter.
4. The latest version of the core JavaScript language is JavaScript 1.8.1, supported by Mozilla and Microsoft Internet Explorer.

1.4 JavaScript and Its Place in a Web Page



```
<html>
  <head>
    <title>See Me!</title>
  </head>
  <script type="text/JavaScript"
                src="file.js">
  </script>
  <link rel=stylesheet
        href=styling.css

  ... continues ...
```

Figure 1.2 The life cycle of a typical Web page.

1.4.1 Analysis of the Diagram

The Players. The players in Figure 1.2 are the applications involved in the life cycle of a Web page:

1. A browser (Firefox, Internet Explorer, Safari, Opera). This is where JavaScript lives!
2. A network (HTTP).
3. A server (Apache, Windows IIS, Zeus).
4. A server module (PHP, ASP.NET, ColdFusion, Java servlet).
5. External files and/or a database (MySQL, Oracle, Sybase).

The Steps. Figure 1.2 illustrates the life cycle of a Web page from when the client makes a request until it gets a response.

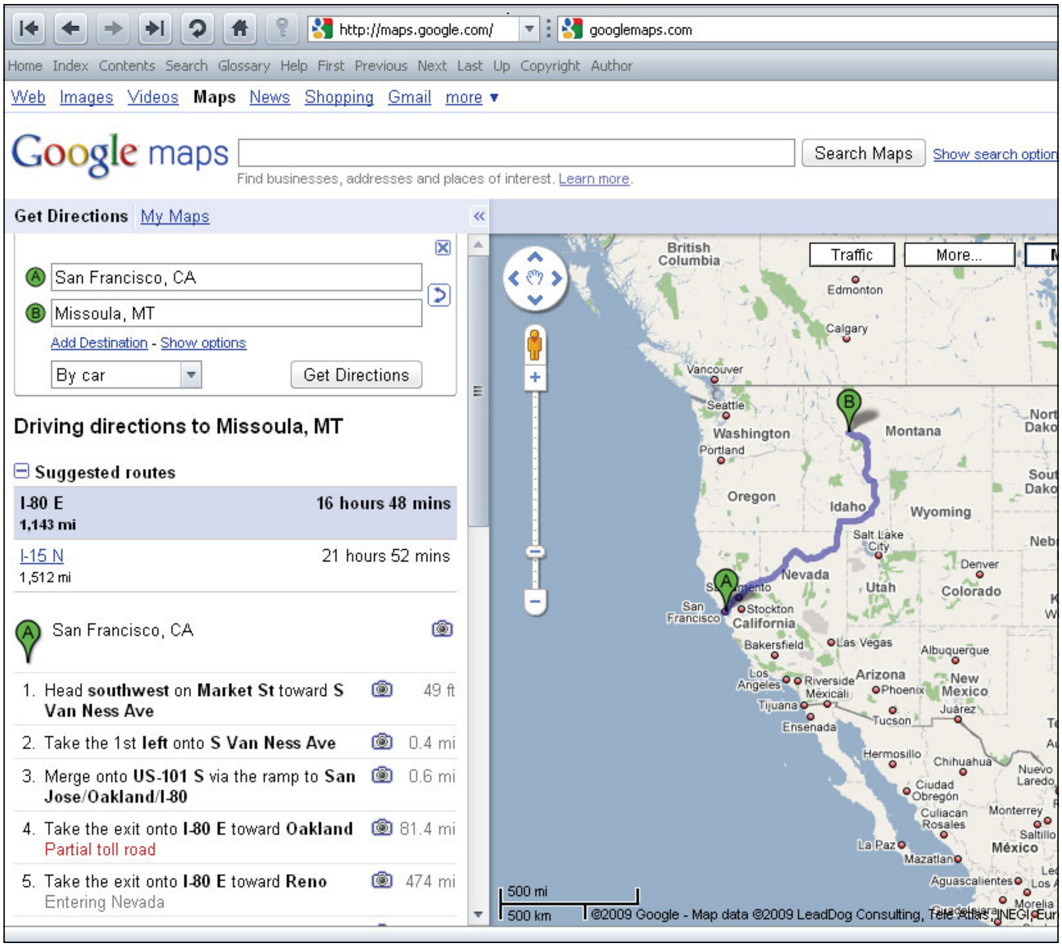
1. On the left hand side of the diagram, we see the client, or browser where the request is made. The user makes a request for a Web site by typing the address

of the Web site in the browser's URL location box. The "request" is transmitted to the server via Hypertext Transfer Protocol (HTTP). The Web server on the other side accepts that request. If the request is for an HTML file, the Web server responds by simply returning the file to the client's browser. The browser will then render the HTML tags, format the page for display, and wait for another request. If the page contains JavaScript tags, the JavaScript interpreter will handle that code based on a user-initiated event such as clicking a button, rolling a mouse over a link or image, or submitting a form. It is with JavaScript that the page becomes interactive. JavaScript detects whatever is happening on the page and responds. It handles fillout forms, feedback, animation, slide-shows, and multimedia. It responds to a key press, a mouse moving over an image, or a user submitting a form. It can read cookies and validate data. It can dynamically change a cell in an HTML table, change the text in a paragraph, or add a new bullet item to a list. But it doesn't do everything. It cannot close a window it didn't open, query a database, update the value in a file upload field, or write to files on a server. After the JavaScript interpreter has completed its tasks, and the page has been fully rendered, another request can be made to the server. Going back and forth between the browser and the server is known as the Request/Response loop, the basis of how the Web works.

2. The cloud between the client side and the server side represents the network. This can be a very large network such as the Internet consisting of millions upon millions of computers, an intranet within an organization, or a wireless network on a personal desktop computer or handheld device. The user doesn't care how big or small the network is—it is totally transparent. The protocol used to transfer documents to and from the server is called HTTP.
3. The server side includes an HTTP Web server such as Apache, Microsoft's IIS, or lighttpd. Web servers are generic programs capable of accepting Web-based requests and providing the response to them. In most cases, this response is simply retrieving the file from server's local file system. With dynamic Web sites, which require processing beyond the capabilities of JavaScript, such as processing form information, sending e-mail, starting a session, or connecting to a database, Web servers turn over the request for a specific file to an appropriate helper application. Web servers, such as Apache and Internet Information Service (IIS) have a list of helper applications that process any specific language. The helper application could be an external program, such as a CGI/Perl script, or one built right into the server, such as ColdFusion, ASP.NET, or a PHP script. For example, if the Web server sees a request for a PHP file, it looks up what helper application is assigned to process PHP requests, turns over the request to the PHP module, and waits until it gets the result back.

1.5 What Is Ajax?

Ajax stands for Asnychronous JavasScript and XML, a term that was coined by Jesse James Garrett in 2005. Ajax is not new. It's been around since 1996, and is a technique used to create fast interactivity without having to wait for a response from the server. As shown in our Web cycle example in Figure 1.2, the browser sends a request to the server and waits for a response, often with a little wheel-shaped icon circling around in the location bar reminding you that the page is loading. As you wait, the browser sits with you and waits, and after each subsequent request, you must wait for the entire page to reload to get the contents of the new page. Ajax lets you send data back and forth between the browser and server without waiting for the whole page to reload. Only parts of the page that change are replaced. Several requests can go out while you are scrolling, zooming in and out, filling out a form, and so on, as those other parts are loaded in the background. Because this interactivity is asnychronous, feedback is immediate with no long waiting times between requests. Some examples of Ajax applications are Ajax Stock Qutos Ticker (SentoSoft LTD), Flickr for photo storage and display, Gmail, Google Suggest, and perhaps the best example, Google Maps at *maps.google.com* (see Figure 1.3).



A San Francisco, CA

B Missoula, MT

By car

Driving directions to Missoula, MT

Suggested routes

I-80 E 1,143 mi	16 hours 48 mins
I-15 N 1,512 mi	21 hours 52 mins

A San Francisco, CA

1. Head **southwest** on **Market St** toward **S Van Ness Ave**	49 ft
2. Take the 1st **left** onto **S Van Ness Ave**	0.4 mi
3. Merge onto **US-101 S** via the ramp to **San Jose/Oakland/I-80**	0.6 mi
4. Take the exit onto **I-80 E** toward **Oakland** Partial toll road	81.4 mi
5. Take the exit onto **I-80 E** toward **Reno** Entering Nevada	474 mi

Figure 1.3 Google uses Ajax for interactivity. © 2010 Google.

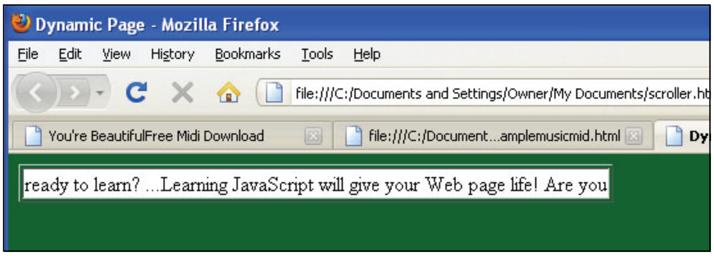
When you use this Web page, you have complete and fast interactivity. You can zoom in, zoom out, move around the map, get directions from one point to another, view the location's terrain, see traffic, view a satellite picture, and so on. In Chapter 18 we discuss how this technique works, but for now think of it as JavaScript on steroids.

1.6 What JavaScript Looks Like

Example 1.1 demonstrates a small JavaScript program. The Web page contains a simple HTML table cell with a scrolling message (see Figure 1.4). Without JavaScript the message would be static, but with JavaScript, the message will continue to scroll across the screen, giving life to an otherwise dead page. This example will be explained in detail later, but for now it is here to show you what a JavaScript program looks like. Notice that the *<script></script>* tags have been highlighted. Between those tags you will see JavaScript code that produces the scrolling effect in the table cell. Within a short time, you will be able to read and write this type of script.

EXAMPLE 1.1

```
<html>
   <head><title>Dynamic Page</title>
      <script type="text/javascript">
         // This is JavaScript. Be patient. You will be writing
         // better programs than this in no time.
         var message="Learning JavaScript will give your Web
                      page life!";
         message += " Are you ready to learn? ";
         var space="...";
         var position=0;
         function scroller(){
            var newtext = message.substring(position,message.length)+
            space + message.substring(0,position);
            var td = document.getElementById("tabledata");
            td.firstChild.nodeValue = newtext;
            position++;
            if (position > message.length){position=0;}
            window.setTimeout(scroller,200);
         }
      </script>
   </head>
   <body bgColor="darkgreen" onload="scroller();">
      <table border="1">
         <tr>
            <td id="tabledata" bgcolor="white">message goes here</td>
         </tr>
      </table>
   </body>
</html>
```


Dynamic Page - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/Documents and Settings/Owner/My Documents/scroller.ht
You're BeautifulFree Midi Download
file:///C:/Document...amplemusicmid.html
Dy

ready to learn? ...Learning JavaScript will give your Web page life! Are you

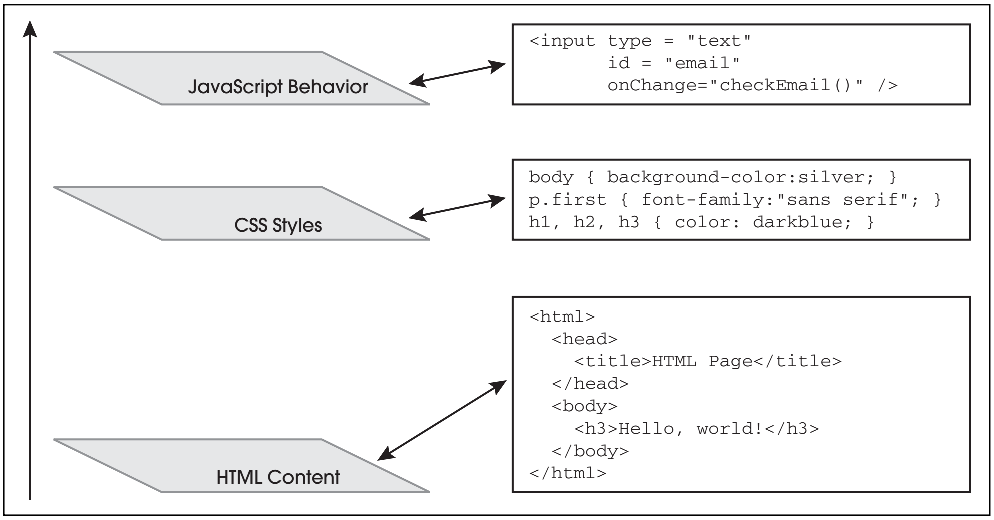
Figure 1.4 Scrolling text with JavaScript (output of Example 1.1).

1.7 JavaScript and Its Role in Web Development

When you start learning JavaScript, JavaScript code will be embedded directly in the content of an HTML page. Once we have covered the core programming constructs, you will see how a document is structured by using the document object model (DOM), and how JavaScript can get access to every element of your page. Finally you will be introduced to cascading style sheets (CSS), a technology that allows you to design your page with a stylized presentation. The combination of HTML, CSS, and JavaScript will allow you to produce a structured, stylized, interactive Web page. As your knowledge grows, so will your Web page, until it becomes necessary to create more pages and link them together. And then you still have to be sure your visitors are having a pleasant experience, no matter what browser they are using, at the same time trying to manage the site behind the scenes. To keep all of this in perspective, Web designers have determined that there are really three fundamental parts to a Web page: the content, the way the content is presented, and the behavior of that content.

1.7.1 The Three Layers

When a Web page is designed on the client (browser) side, it might start out as a simple HTML static page. Later the designer might want to add style to the content to give the viewer a more visually attractive layout. Last, to liven things up, JavaScript code is added to give the viewer the ability to interact with the page, make the page do something. A complete Web page, then, can be visualized as three separate layers: the content or structural layer, the style or presentation layer, and the behavior layer (see Figure 1.5). Each of these layers requires careful planning and skill. Designers are not necessarily programmers and vice versa. Separating the layers allows the designer to concentrate on the part he or she is good at, while the programmer can tweak the code in the JavaScript application without messing up the design. Of course, there is often a blurred line between these layers but the idea of separating content structure and style from behavior lends to easier maintenance, less repetition, and hopefully less debugging.



```
<input type = "text"
       id = "email"
       onChange="checkEmail()" />
```

```
body { background-color:silver; }
p.first { font-family:"sans serif"; }
h1, h2, h3 { color: darkblue; }
```

```
<html>
  <head>
    <title>HTML Page</title>
  </head>
  <body>
    <h3>Hello, world!</h3>
  </body>
</html>
```

Figure 1.5 Three layers that make up a Web page.

Content or Structure. In Web development, HTML/XML markup makes up the content layer, and it also structures the Web document. The content layer is what a viewer sees when he or she comes to your Web page. Content can consist of text or images and include the links and anchors a viewer uses to navigate around your Web site. Because HTML/XML elements are used to create the structural content of your page, misusing those elements might not seem relevant for a quick visual fix, but might be very relevant when applying CSS and JavaScript. For example, using headings out of order to force a change in font size, such H1, H3, and then H2 tags, in that order is invalid HTML. These tags are intended to define the structure of the document on the display. The browser views the Web page as a tree-like structure, a model consisting of objects, where each HTML element (e.g., HEAD, BODY, H1) is an object in the model. This document tree, the DOM, defines the hierarchical logic of your document, which becomes an important tool for creating dynamic content. Because the structure is so important, valid markup should be a priority before going to the next layer: the CSS presentation layer. See Section 1.12 for markup validation tools.

Style or Presentation. The style or presentation layer is how the document will appear and on what media types. This layer is defined by CSS. Prior to CSS, nearly all of the presentation was contained within the HTML markup; all font colors, background styles, element positions and alignments, borders, and so on, had to be explicitly, often repeatedly, included in the HTML markup for the page. If, for example, you decided you wanted your page to have a blue font for all headings, then you would have to change each heading in the document. CSS changed all that. It gave designers the ability to move the presentational content into separate style sheets, resulting in much simpler HTML markup. Now you could change the font color in one place to affect all of the pages in your site. Although styles can be embedded within a document and give you control over selected elements, it is more likely they will be found in separate *.css* files to let you produce sweeping changes over an entire document. With one CSS file you can control the style of one or thousands of documents. External style sheets are cached, reduce the amount of code, and let you modify an entire site without mangling the HTML content pages. And CSS works with JavaScript and the DOM to create a dynamic presentation, often known as DHTML.

Behavior. The behavior layer is the layer of a Web page that makes the page perform some action. For most Web pages, the first level of behavior is JavaScript. JavaScript allows you to dynamically control the elements of the Web page based on user interaction such as an individual keystroke, moving a mouse, submitting form input, and so on. JavaScript also makes it easy to perform style changes on the fly. Although traditionally CSS and JavaScript are separate layers, now with the DOM, they work so closely together that the lines are somewhat blurred. JavaScript programs are often stored in external files, which are then put in libraries where other programmers can share them. See *http://JavaScriptlibraries.com/.*

Unobtrusive JavaScript. When you hear this phrase, "Make sure you use unobtrusive JavaScript," and you will hear or read about it once you have started really using JavaScript, it refers to the three layers we just discussed. It is a technique to completely separate JavaScript from the other two layers of Web development by putting JavaScript code in its own file and leaving the HTML/XHTML/XML and CSS in their own respective files. In the following chapters we have included most of the JavaScript examples in the same the HTML document because the files are small and serve to teach a particular aspect of the language. So for the time being, we will be obtrusive.

Once you have learned the JavaScript basics and start working on larger applications, you might want to understand this more fully. For the seven rules of unobtrusive JavaScript, go to *http://icant.co.uk/articles/seven-rules-of-unobtrusive-JavaScript/.*

1.8 JavaScript and Events

HTML is static. It structures and defines how the elements of a Web page will appear in the browser; for example, it is used to create buttons, tables, text boxes, and fillout forms, but it cannot by itself react to user input. JavaScript is not static; it is dynamic. It reacts asynchronously to events triggered by a user. For example, when a user fills out a form; presses a button, link, or image; or moves his or her mouse over a link, JavaScript can respond to the event and interact dynamically with the user. JavaScript can examine user input and validate it before sending it off to a server, or cause a new image to appear if a mouse moves over a link or the user presses a button, reposition objects on the page, even add, delete, or modify the HTML elements on the fly. Events are discussed in detail in Chapter 13, "Handling Events," but you should be made aware of them right at the beginning because they are inherently part of what JavaScript does, and there will be many examples throughout this text that make use of them.

The events, in their simplest form, are tied to HTML. In the following example, an HTML form is created with the *<form>* tag and its attributes. Along with the *type* and *value* attributes, the JavaScript *onClick* event handler is just another attribute of the HTML *<form>* tag. The type of input device is called a *button* and the value assigned to the button is "*Pinch me*". When the user clicks the button in the browser window, a JavaScript event, called *click*, will be triggered. The *onClick* event handler is assigned a value that is the command that will be executed after the button has been clicked. In our example, it will result in an alert box popping up in its own little window, displaying "*OUCH!!*". See the output of Example 1.2 in Figures 1.6 and 1.7.

EXAMPLE 1.2

```
<html>
  <head><title>Event</title></head>
  <body>
    <form>
      <input type ="button"
             value = "Pinch me"
             onClick="alert('OUCH!!')" />
    </form>
  </body>
</html>
```

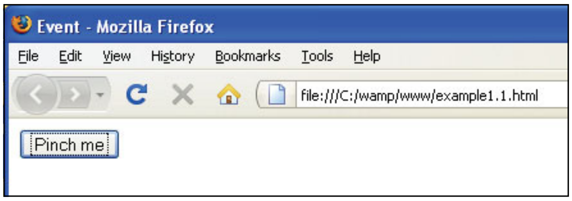
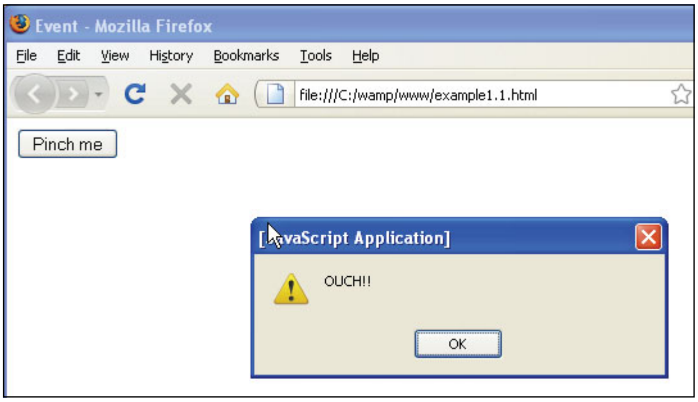

Event - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/wamp/www/example1.1.html
Pinch me

Figure 1.6 User initiates a click event when he or she clicks the mouse on the button.


Event - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/wamp/www/example1.1.html
Pinch me
[JavaScript Application]
OK

OUCH!!

Figure 1.7 The *onClick* event handler is triggered when the button labeled "Pinch me" is pressed.

Some of the events that JavaScript can handle are listed in Table 1.1.

Table 1.1 JavaScript Event Handlers

Event Handler	***What Caused It***
onAbort	Image loading was interrupted.
onBlur	The user moved away from a form element.
onChange	The user changed a value in a form element.
onClick	The user clicked a button-like form element.
onError	The program had an error when loading an image.
onFocus	The user activated a form element.
onLoad	The document finished loading.
onMouseOut	The mouse moved away from an object.
onMouseOver	The mouse moved over an object.
onSubmit	The user submitted a form.
onUnLoad	The user left the window or frame.

1.9 Standardizing JavaScript and the W3C

ECMAScript, which is more commonly known by the name JavaScript™, is an essential component of every Web browser and the ECMAScript standard is one of the core standards that enable the existence of interoperable Web applications on the World Wide Web.

—Ema International

During the 1990s Microsoft Internet Explorer and Netscape were competing for industry dominance in the browser market. They rapidly added new enhancements and proprietary features to their browsers, creating incompatibilities that made it difficult to view a Web site the same way in the two browsers. These times were popularly called the Browser Wars, ending with Microsoft's Internet Explorer browser winning. For now there seems to be peace among modern browsers, due to the fact that the World Wide Web Consortium (W3C) set some standards. To be a respectable browser, compliance with the standards is expected.

To guarantee that there is one standard version of JavaScript available to companies producing Web pages, European Computer Manufacturers Association (ECMA) worked with Netscape to provide an international standardization of JavaScript called ECMAScript. ECMAScript is based on core JavaScript and behaves the same way in all applications that support the standard. The first version of the ECMA standard is documented in the ECMA-262 specification. Both JavaScript (Mozilla) and JScript (Microsoft IE) are really just a superset of ECMAScript and strive to be compatible with ECMAScript even though they have some of their own additions.[5] After ECMAScript was released, W3C began work on a standardized DOM, known as DOM Level 1, and recommended in late 1998. DOM Level 2 was published in late 2000. The current release of the DOM specification was published in April 2004. By 2005, large parts of W3C DOM were well supported by common ECMAScript-enabled browsers, including Microsoft Internet Explorer version 6 (2001), Gecko-based browsers (like Mozilla Firefox, and Camino), Konqueror, Opera, and Safari. In fact 95% of all modern browsers support the DOM specifications.

For the latest information on the latest ECMA-252 edition 5, see *http://www.ecmascript.org/*.

1.9.1 JavaScript Objects

Everything you do in JavaScript involves objects, just as everything you do in real life involves objects. JavaScript sees a Web page as many different objects, such as the browser object, the document object, and each element of the document as an object; for example, forms, images, and links are also objects. In fact every HTML element in the page can be viewed as an object. HTML H1, P, TD, FORM, and HREF elements are all examples of objects. JavaScript has a set of its own core objects that allow you to manipulate strings, numbers, functions, dates, and so on, and JavaScript allows you to create your own objects. When you see a line such as:

```
document.write("Hello, world");
```

the current page is the document object. After the object, there is a dot that separates the object from the *write* method. A method is a function that lets the object do something. The method is always followed by a set of parentheses that might or might not contain data. In this example the parentheses contain the string “Hello, world” telling JavaScript to write this string in the document window, your browser. In Chapter 8, “Objects,” we discuss objects in detail. Because everything in JavaScript is viewed as an object, it is important to understand the concept from the start.

1.9.2 The Document Object Model

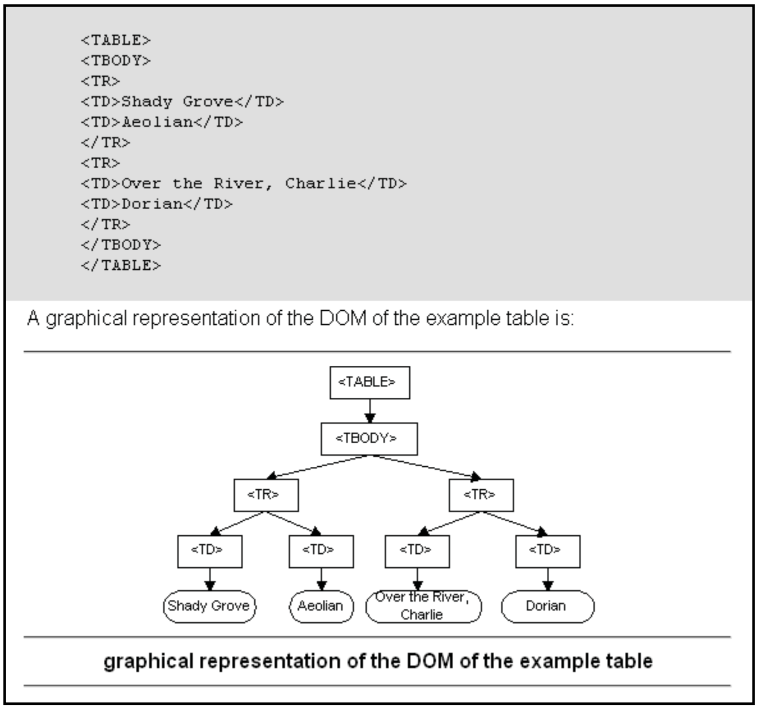
What is the DOM? A basic Web document consists of HTML/XML markup. The browser’s job is to turn that markup into a Web page so that you can see text, input devices, pictures, tables, and so on in your browser window. It is also the browser’s job to store its interpretation of the HTML page as a model, called the Document Object Model. The model is similar to the structure of a family tree, consisting of parents, children, siblings, and so on. Each element of the tree is related to another element in the

5. ECMAScript 5th edition adds some new features and is now available for review and testing (2009).

tree. These elements are referred to as nodes, with the root parent node of the tree at the top. With this upside down tree model every element of the document becomes an object accessible by JavaScript (and other applications), thus giving the JavaScript programmer control over an entire Web page; that is, the ability to navigate, create, add, modify, or delete the elements and their content dynamically.

As mentioned earlier, the DOM, Level 1[6] (see *http://www.w3.org/DOM*), a standard application programming interface (API) developed by the W3C is implemented by all modern browsers, including Microsoft Internet Explorer version 6 (2001), Gecko-based browsers (like Mozilla Firefox and Camino), Konqueror, Opera, and Safari.

After you learn the fundamentals of JavaScript, you will see how to create and manipulate objects, how to use the core objects, and then how to use JavaScript to control every part of your Web page with the DOM. With CSS, the DOM, and JavaScript you can reposition elements on a page dynamically, create animation, create scrolling marquees, and change the style of the page with fancy fonts and colors based on user input or user-initiated events, such as rolling the mouse over an image or link, clicking an icon, submitting a fillout form, or just opening up or closing a new window. Figure 1.8 demonstrates



```
<TABLE>
<TBODY>
<TR>
<TD>Shady Grove</TD>
<TD>Aeolian</TD>
</TR>
<TR>
<TD>Over the River, Charlie</TD>
<TD>Dorian</TD>
</TR>
</TBODY>
</TABLE>
```

A graphical representation of the DOM of the example table is:

graphical representation of the DOM of the example table

Figure 1.8 http://www.w3.org/TR/DOM-Level-2-Core/introduction.html.

6. DOM Levels 2 and 3 have also been developed by W3C, but DOM Level 1 is supported by most browsers.

an HTML table and how it is represented as a tree where each element is related to its parent and siblings as described by the W3C shown at *http://www.w3.org/DOM*.

1.10 What Browser?

When a user receives a page that includes JavaScript, the script is sent to the JavaScript interpreter, which executes the script. Because each browser has its own interpreter, there are often differences in how the code will be executed. And as the competing companies improve and modify their browsers, new inconsistencies may occur. There are not only different types of browsers to cause the incompatibilities but also different versions of the same browser. Because modern browsers conform to the W3C standards, these inconsistencies tend to be less of a distraction than they were in the past. Popular browsers today are shown in Table 1.2.

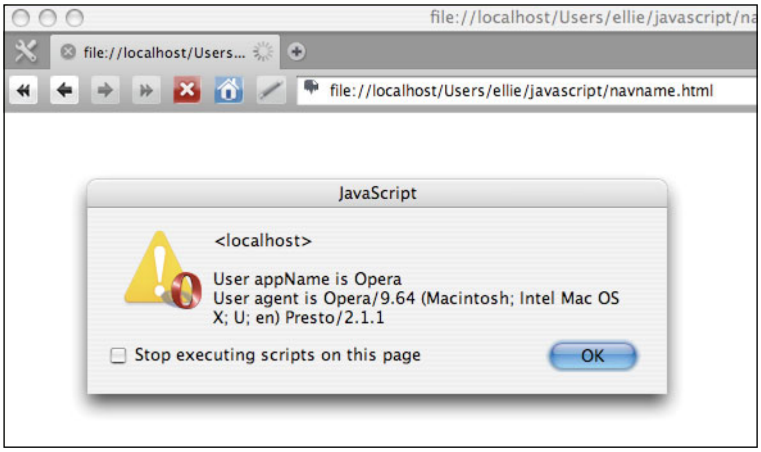
Table 1.2 Modern Browsers

Browser	***Web Site***
Internet Explorer	*microsoft.com/windows/ie*
Firefox	*mozilla.org/products/firefox*
Safari	*apple.com/safari*
Opera	*opera.com*
Google Chrome	*google.com/chrome*
Konqueror	*konqueror.org/*

The little script in Example 1.3 should tell you what browser you are using. Even though the application name might display Netscape for Firefox and Microsoft Internet Explorer for Opera, if you examine the user agent, you will be able find Firefox or Opera as part of the output string (see Figure 1.9). Programs that determine the browser type are called browser sniffers. We have a complete example in Chapter 10, “It’s the BOM! Browser Objects.”

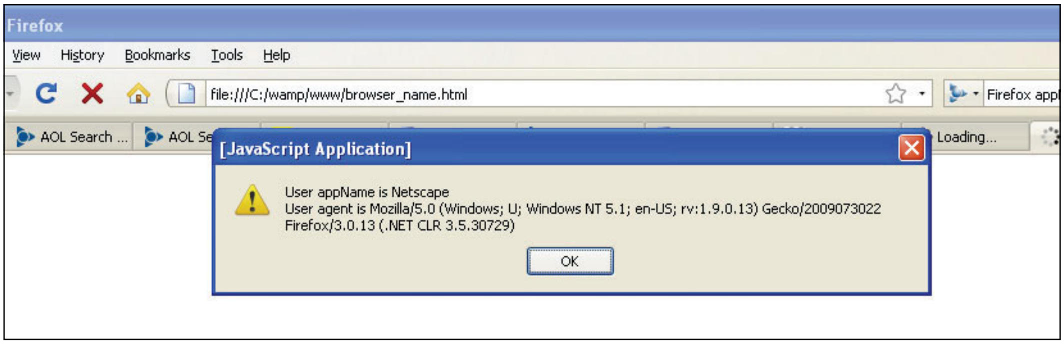
EXAMPLE 1.3

```
<script type="text/javascript">
  alert("User appName is "+ navigator.appName +
        "\nUser agent is "+ navigator.userAgent);
</script>
```



```
<localhost>

User appName is Opera
User agent is Opera/9.64 (Macintosh; Intel Mac OS
X; U; en) Presto/2.1.1
```

file://localhost/Users/ellie/javascript/navname.html
JavaScript
Stop executing scripts on this page
OK



```
User appName is Netscape
User agent is Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.0.13) Gecko/2009073022
Firefox/3.0.13 (.NET CLR 3.5.30729)
```

Firefox
View History Bookmarks Tools Help
file:///C:/wamp/www/browser_name.html
AOL Search ...
Loading...
[JavaScript Application]
OK

Figure 1.9 Output from Example 1.3.

1.10.1 Versions of JavaScript

JavaScript has a history. Invented by Netscape, the first version was JavaScript 1.0. It was new and buggy and has long since been replaced by much cleaner versions. Microsoft has a scripting language comparable to JavaScript called JScript. Table 1.3 lists versions of both JavaScript and JScript. For a discussion of JavaScript versions and development see *http://ejohn.org/blog/versions-of-JavaScript/*.

Table 1.3 JavaScript Versions

JavaScript or JScript Version	***Browsers Supported***
JavaScript 1.0 1996	Netscape Navigator 2.0, Internet Explorer 3.0
JavaScript 1.1 1996	Netscape Navigator 3.0, Internet Explorer 4.0
JavaScript 1.2 1997	Netscape Navigator 4.0–4.05, Internet Explorer 4.0

Table 1.3 JavaScript Versions (continued)

JavaScript or JScript Version	***Browsers Supported***
JavaScript 1.3 1998	ECMA-232, Netscape Navigator 4.06–4.7x, Internet Explorer 5.0
JavaScript 1.5 2000	ECMA-232, Netscape Navigator 6.0+, Mozilla Firefox, Internet Explorer 5.5+, JScript 5.5, 5.6, 5.7, 6
JavaScript 1.6 2006	Mozilla Firefox, Safari
JavaScript 1.7 2006	Mozilla Firefox, Safari, Google Chrome
JavaScript 1.8 2008	Mozilla Firefox

JavaScript is supported by Firefox, Explorer, Opera, and all newer versions of these browsers. In addition, HotJava 3 supports JavaScript, as do iCab for the Mac, WebTV, OmniWeb for OS X, QNX Voyager and Konqueror for the Linux KDE environment. NetBox for TV, AWeb and Voyager 3 for Amiga, and SEGA Dreamcast and ANT Fresco on RISC OS also support JavaScript.

JavaScript 2 and the Future of the Web - Windows Internet Explorer
https://developer.mozilla.org/presentations/xtech20
AOL Search
File Edit View Favorites Tools Help
Favorites
JavaScript 2 and the Future of the Web
Page Safety Tools
Angst Over JavaScript 2

Figure 1.10 JavaScript2 and the Web, an informative paper by Brendan Eich.

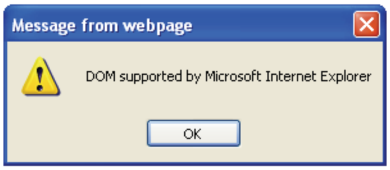
So where is JavaScript now? As of December 2009, the ECMA-262 Standard is in its 5th edition. JavaScript is a dialect of ECMAScript, but JavaScript 1.8 is comparable to ECMAScript, edition 3 and is currently the most widely used version (JavaScript 1.9 is available for download). To understand some of the proposals for a JavaScript2 version (ECMAScript Edition 4), Brian Eich, the creator of JavaScript, wrote an interesting article a few years ago that he published on the Web. If nothing else, it tells you some of the pros and cons of the current state of the JavaScript language and the obstacles faced in trying to change it. See Figure 1.10.

1.10.2 Does Your Browser Follow the Standard?

Modern browsers are using versions of JavaScript 1.5 or above, which generally follow the standards set by the W3C. The snippet of code in Example 1.4 tests to see if you are using a modern version of JavaScript that follows the standard DOM (see Figure 1.11). Both the *getElementById* and *createTextNode* are part of the W3C standard, which supports the DOM.

EXAMPLE 1.4

```
<script type="text/javascript">
  if (document.getElementById && document.createTextNode){
    alert("DOM supported by " + navigator.appName);
  }
</script>
```

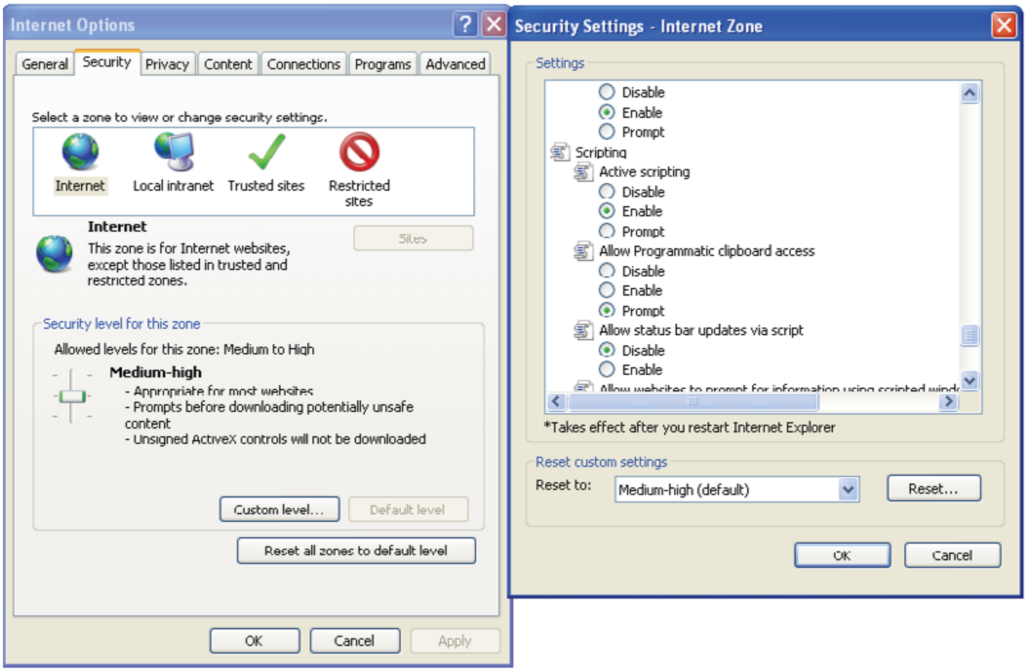


DOM supported by Microsoft Internet Explorer

Figure 1.11 Internet Explorer supports the standard.

1.10.3 Is JavaScript Enabled on Your Browser?

To see if JavaScript is enabled on your browser, you can check the options menu of Firefox by going to the Tools menu/Options/Content. If using Apple's Safari browser, go to Safari menu/Preferences/Security and with Internet Explorer, go to the Tools menu/Internet Options/Security/Custom Level and enable Active scripting (see Figure 1.12). If using Opera go to the Opera menu/Preferences/Advanced/Content and click Enable JavaScript. An easy way to test if your browser has JavaScript enabled is to go to the Web site http://www.mistered.us/test/alert.shtml and follow directions (see Figure 1.13).



Select a zone to view or change security settings.

Internet
This zone is for Internet websites, except those listed in trusted and restricted zones.

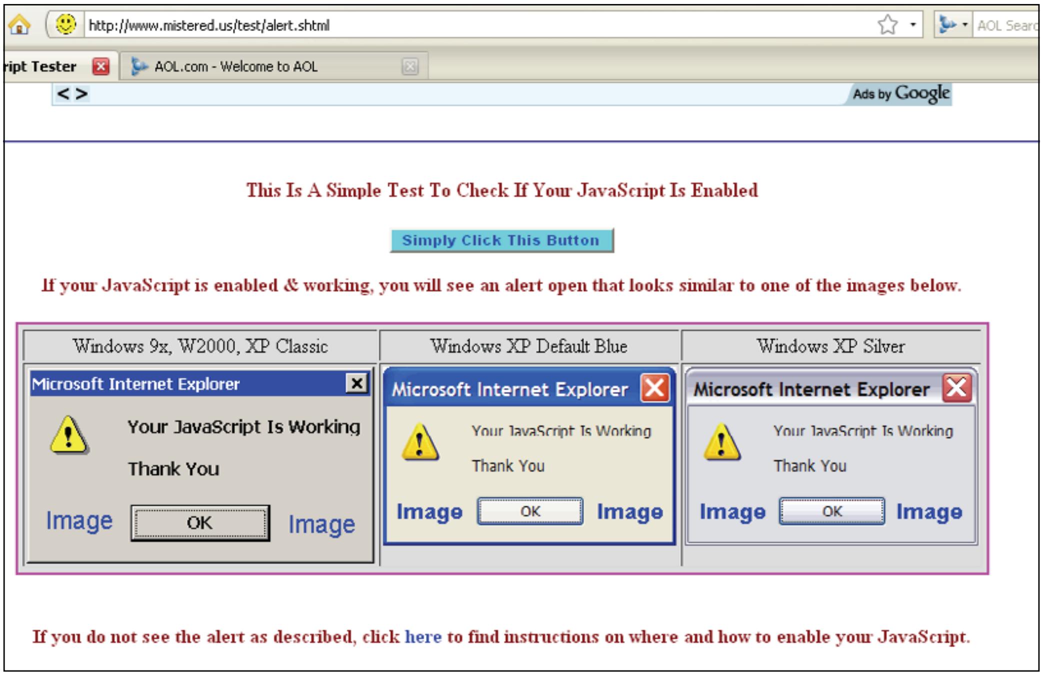
Security level for this zone
Allowed levels for this zone: Medium to High
Medium-high
- Appropriate for most websites
- Prompts before downloading potentially unsafe content
- Unsigned ActiveX controls will not be downloaded

Disable
[✓ Enable]
Prompt
Scripting
Active scripting
Disable
[✓ Enable]
Prompt
Allow Programmatic clipboard access
Disable
Enable
[✓ Prompt]
Allow status bar updates via script
[✓ Disable]
Enable
Allow websites to prompt for information using scripted windo

*Takes effect after you restart Internet Explorer

Reset to: Medium-high (default)

Figure 1.12 Enabling JavaScript on Microsoft Internet Explorer.



This Is A Simple Test To Check If Your JavaScript Is Enabled

If your JavaScript is enabled & working, you will see an alert open that looks similar to one of the images below.

Your JavaScript Is Working
Thank You

If you do not see the alert as described, click here to find instructions on where and how to enable your JavaScript.

Figure 1.13 Is your browser JavaScript enabled?

1.11 Where to Put JavaScript

Before learning JavaScript, you should be familiar with HTML and how to create an HTML document. This doesn't mean that you have to be an expert, but you should be familiar with the structure of HTML documents and how the tags are used to display various kinds of content on your browser. Once you have a static HTML document, then adding basic JavaScript statements is quite easy. (Go to *http://www.w3schools.com* for an excellent HTML tutorial.) In this text we have devoted a separate chapter to CSS. CSS allows you to control the style and layout of your Web page by changing fonts, colors, backgrounds, margins, and so on in a single file. With HTML, CSS, and JavaScript you can create a Web site with structure, style, and action.

Client-side JavaScript programs are embedded in an HTML document between HTML head tags *<head>* and *</head>* or between the body tags *<body>* and *</body>*. Many developers prefer to put JavaScript code within the *<head>* tags, and at times, as you will see later, it is the best place to store function definitions and objects. If you want text displayed at a specific spot in the document, you might want to place the JavaScript code within the *<body>* tags (as shown in Example 1.5). Or you might have multiple scripts within a page, and place the JavaScript code within both the *<head>* and *<body>* tags. In either case, a JavaScript program starts with a *<script>* tag, and ends with a *</script>* tag. And if the JavaScript code is going to be long and involved, or may be shared by multiple pages, it should be placed in an external file (text file ending in *.js*) and loaded into the page. In fact, once you start developing Web pages with JavasScript, it is customary to separate the HTML/CSS content from the programming logic (JavaScript) by creating separate files for each entity.

When a document is sent to the browser, it reads each line of HTML code from top to bottom, and processes and displays it. As JavaScript code is encountered, it is read and executed by the JavaScript interpreter until it is finished, and then the parsing and rendering of the HTML continues until the end of the document is reached.

EXAMPLE 1.5

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
          "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head><title>First JavaScript Sample</title></head>
  <body bgcolor="yellow" text="blue">
    <script type="text/javascript">
      document.write("<h2>Welcome to the JavaScript World!</h2>");
    </script>
    <big>This is just plain old HTML stuff.</big>
  </body>
</html>
```

EXPLANATION

1 The doctype declaration tells the Web browser what version of the markup language will be used for this page and should be the very first thing in an HTML document, and must be included in an XHTML document.The doctype declaration refers to a Document Type Definition (DTD). The DTD specifies the rules for the markup language, so that the browsers can render the content correctly. This document is declared to be HTML 4.01 Strict. HTML 4.01 Strict is a version of HTML 4.01 that emphasizes structure over presentation. Deprecated elements and attributes, frames, and link targets are not allowed in HTML 4 Strict. In most of the examples in this book, this declaration will be omitted just to save space, but when you create your own documents, you should include the doctype declaration.

2 This is the starting tag for an HTML document.

3 This is the HTML *<head>* tag. The *<head>* tags contain all the elements that don't belong in the body of the document, such as the *<title>* tags, as well as JavaScript tags.

4 The *<body>* tag defines the background color and text color for the document.

5 This *<script>* tag is the starting HTML tag for the JavaScript script, which consists of a mix of textual content and JavaScript instructions. JavaScript instructions are placed between this tag and the closing *</script>* tag. JavaScript understands JavaScript instructions, not HTML.

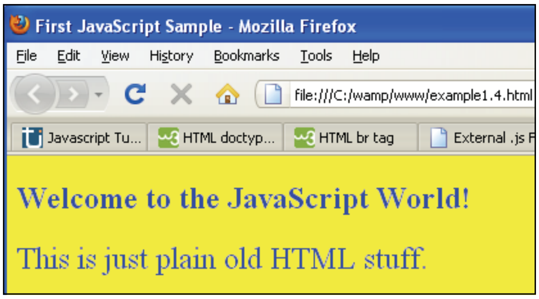
The JavaScript *writeln* method is called for the document. The string enclosed in parentheses is passed to the JavaScript interpreter. If the JavaScript interpreter encounters HTML content, it sends that content to the HTML renderer and it is printed into the document on the browser. The normal HTML parsing and rendering resumes after the closing JavaScript tag is reached.

6 This is the ending JavaScript tag. The output is shown in Figure 1.14.

7 HTML tags and text continue in the body of the document.

8 The body of the document ends here.

9 This is the ending tag for the HTML document.


First JavaScript Sample - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/wamp/www/example1.4.html
Javascript Tu...
HTML doctyp...
HTML br tag
External .js F

Welcome to the JavaScript World!

This is just plain old HTML stuff.

Figure 1.14 Example 1.5 output: JavaScript has been inserted in a document.

1.11.1 JavaScript from External Files

When scripts are long or need to be shared by other pages, they are usually placed in external files, separate from the HTML page. Keeping the JavaScript separate (unobtrusive JavaScript) from the HTML or CSS files is important when developing a Web site. It enables you to apply one set of functions to every page of the site, so that when you need to make a change, you can do it in one document rather than going through each individual page to apply the change. A JavaScript external file contains just plain JavaScript code and is saved as a .js file. The .js file is linked to the Web page by including it between the <*head*> tags of the HTML document and within its own <*script*> tags. The external JavaScript file is assigned to the *src* attribute of the <*script*> tag in the HTML file. The external file name includes the full URL if the script is on another server, directory path or just the script name if in the local directory. You can include more than one .js script in a file.

```
<script type="text/javascript"
        src="http://servername/JavaScriptfile.js">
</script>
```

The following examples, although very small, give you the idea of how external files are used. The welcome.js script contains a JavaScript function (see Chapter 7, “Functions”).

EXAMPLE 1.6

```
// The external file called "welcome.js"
function welcome(){
  alert("Welcome to JavaScript!");
}
```

EXAMPLE 1.7

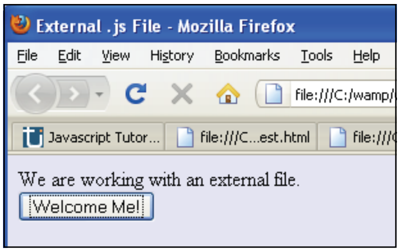
```
     <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
             "http://www.w3.org/TR/html4/strict.dtd">
     <html>
       <head>
         <title>External .js File</title>
1        <script type="text/javascript" src="welcome.js">
2        </script>
       </head>
       <body bgColor="lavender">
         We are working with an external file.<br />
3        <input type="button" onClick="welcome()" value="Welcome Me!" />
       </body>
     </html>
```

EXPLANATION

1 The JavaScript <*script*> tag's *src* attribute is assigned the name of a file (name must end in *.js*) that contains JavaScript code. The file's name is *welcome.js* and it contains a JavaScript program of its own containing a simple JavaScript function that will be called when the user clicks the "Welcome Me!" button. See Figure 1.15.

2 The JavaScript program ends here.

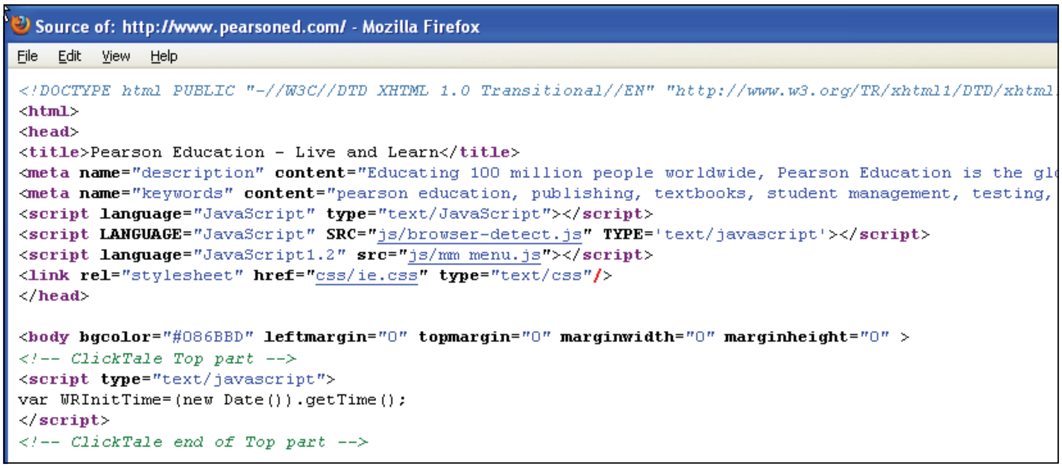
3 This is an HTML button input device. When the user clicks the button, a JavaScript function called *welcome()* will be called. It is defined in the external file, *welcome.js*.


External .js File - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/wamp/
Javascript Tutor...
file:///C...est.html
file:///
Welcome Me!

We are working with an external file.

Figure 1.15 After clicking the "Welcome Me!" button, a function from the external .js file is called.

As you can see in Figure 1.16, Pearson Education uses many external JavaScript files (.js) files to produce their Web site.



```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml
<html>
<head>
<title>Pearson Education - Live and Learn</title>
<meta name="description" content="Educating 100 million people worldwide, Pearson Education is the gl
<meta name="keywords" content="pearson education, publishing, textbooks, student management, testing,
<script language="JavaScript" type="text/JavaScript"></script>
<script LANGUAGE="JavaScript" SRC="js/browser-detect.js" TYPE='text/javascript'></script>
<script language="JavaScript1.2" src="js/mm menu.js"></script>
<link rel="stylesheet" href="css/ie.css" type="text/css"/>
</head>

<body bgcolor="#086BBD" leftmargin="0" topmargin="0" marginwidth="0" marginheight="0" >
<!-- ClickTale Top part -->
<script type="text/javascript">
var WRInitTime=(new Date()).getTime();
</script>
<!-- ClickTale end of Top part -->
```

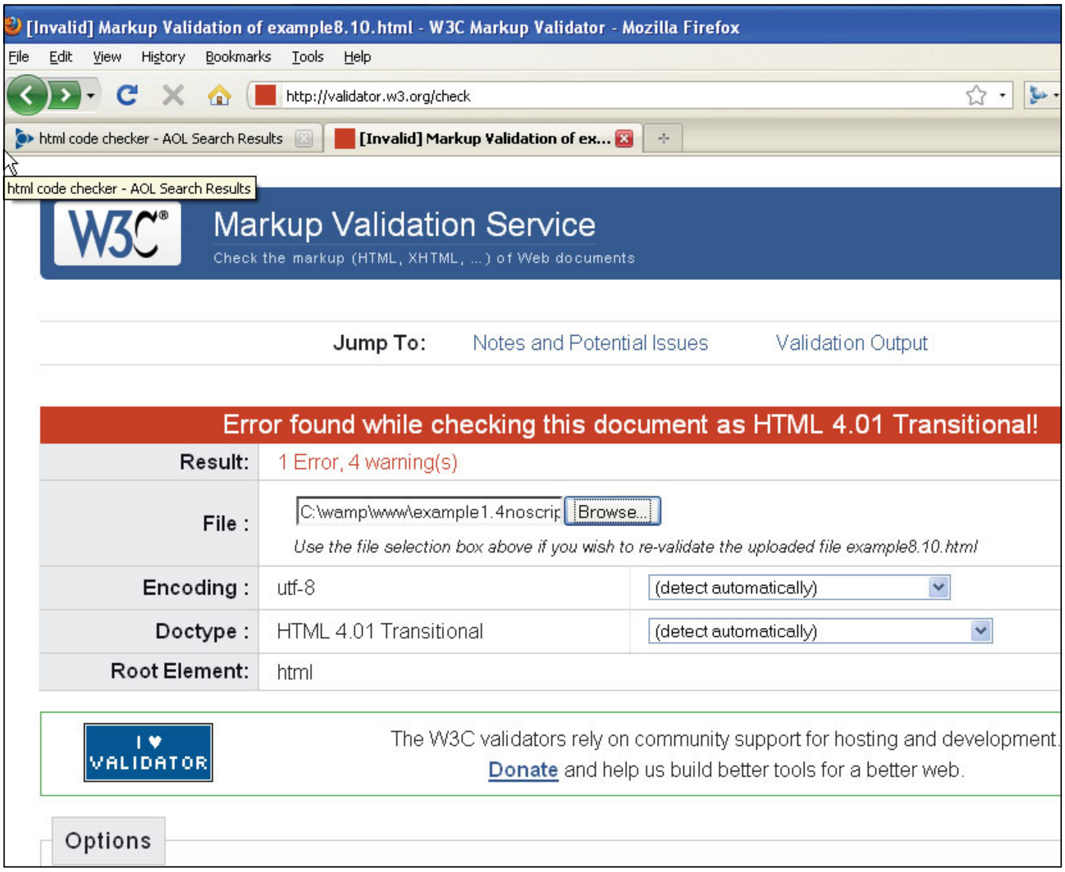
Figure 1.16 Viewing the source code for a Web site using external JavaScript files.

1.12 Validating Your Markup

Because your JavaScript code does not stand alone but is integrated with HTML/XHTML and CSS, it is important to find validation tools to verify that your markup is correct, especially when conforming to the DOM and W3C standards. There are a number of free tools on the Web to help you make sure your markup is valid.

1.12.1 The W3C Validation Tool

The W3C validation tool is shown in Figure 1.17. This tool allows you to validate by URI, file upload, or by direct input. This validator checks the markup validity of Web documents in HTML, XHTML, SMIL, MathML, and so on, but to evaluate specific content such as RSS/Atom feeds or CSS stylesheets, MobileOK content, or to find broken links, there are other validators and tools available.


[Invalid] Markup Validation of example8.10.html - W3C Markup Validator - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://validator.w3.org/check
html code checker - AOL Search Results
[Invalid] Markup Validation of ex...
html code checker - AOL Search Results
Markup Validation Service
Check the markup (HTML, XHTML, ...) of Web documents
Jump To: Notes and Potential Issues Validation Output
Browse...
I ♥ VALIDATOR
Options

Error found while checking this document as HTML 4.01 Transitional!

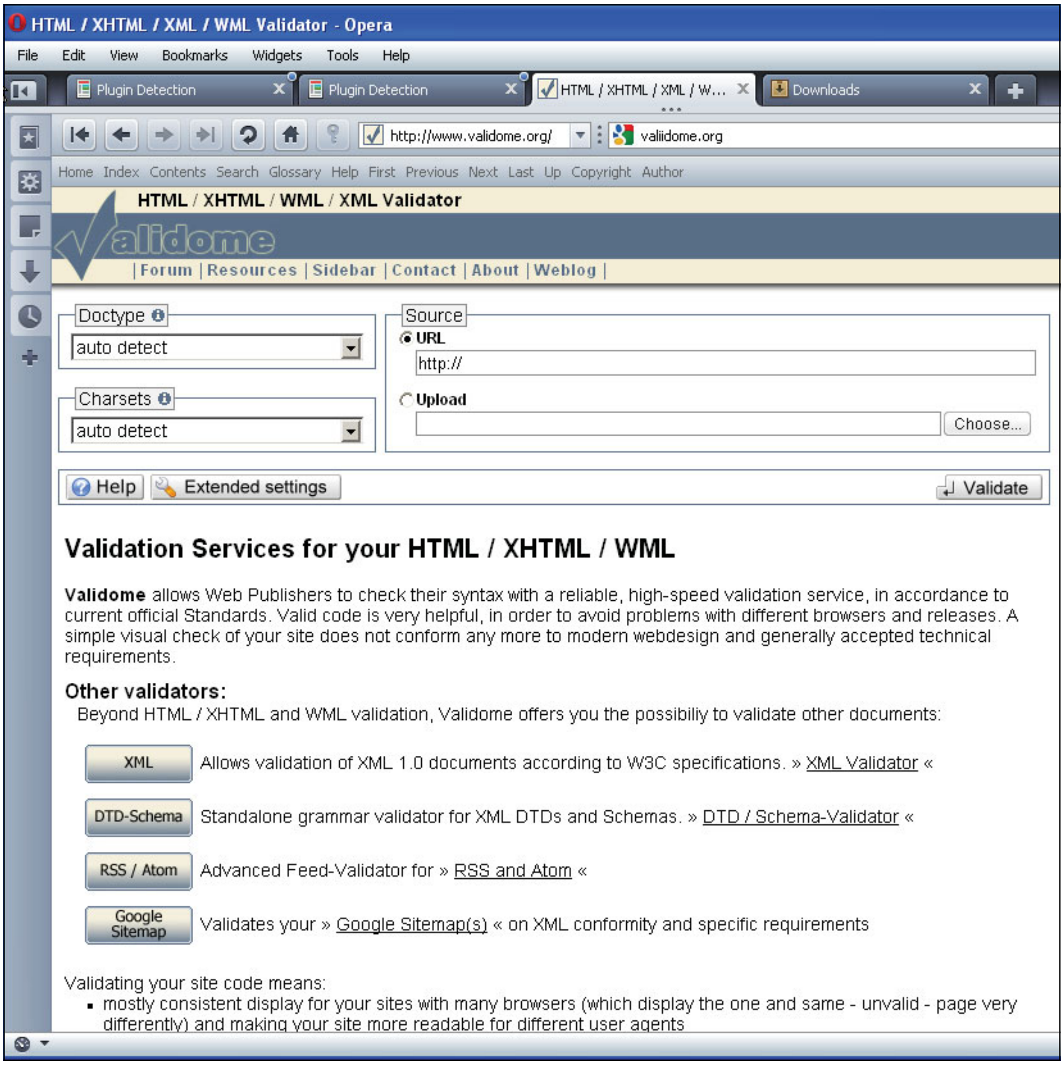
Result:	1 Error, 4 warning(s)	
File :	C:\wamp\www\example1.4noscrip Use the file selection box above if you wish to re-validate the uploaded file example8.10.html	
Encoding :	utf-8	(detect automatically)
Doctype :	HTML 4.01 Transitional	(detect automatically)
Root Element:	html	

The W3C validators rely on community support for hosting and development. Donate and help us build better tools for a better web.

Figure 1.17 The W3C markup validation of an HTML page.

1.12.2 The Validome Validation Tool

The validator at *www.validome.org* is another excellent tool for validating an online page or an XHTML document. Just go to the Validome Web page and select the URL or Upload option to get a file from your hard drive. Once you have selected a file, just click Validate (see Figure 1.18).


HTML / XHTML / XML / WML Validator - Opera
File Edit View Bookmarks Widgets Tools Help
Plugin Detection
Plugin Detection
HTML / XHTML / XML / W...
Downloads
http://www.validome.org/
validome.org
Home Index Contents Search Glossary Help First Previous Next Last Up Copyright Author
HTML / XHTML / WML / XML Validator
validome
| Forum | Resources | Sidebar | Contact | About | Weblog |
Doctype
auto detect
Charsets
auto detect
Source
URL
http://
Upload
Choose...
Help
Extended settings
Validate
XML
DTD-Schema
RSS / Atom
Google Sitemap

Validation Services for your HTML / XHTML / WML

Validome allows Web Publishers to check their syntax with a reliable, high-speed validation service, in accordance to current official Standards. Valid code is very helpful, in order to avoid problems with different browsers and releases. A simple visual check of your site does not conform any more to modern webdesign and generally accepted technical requirements.

Other validators:

Beyond HTML / XHTML and WML validation, Validome offers you the possibiliy to validate other documents:

XML	Allows validation of XML 1.0 documents according to W3C specifications. » XML Validator «
DTD-Schema	Standalone grammar validator for XML DTDs and Schemas. » DTD / Schema-Validator «
RSS / Atom	Advanced Feed-Validator for » RSS and Atom «
Google Sitemap	Validates your » Google Sitemap(s) « on XML conformity and specific requirements

Validating your site code means:

- mostly consistent display for your sites with many browsers (which display the one and same - unvalid - page very differently) and making your site more readable for different user agents

Figure 1.18 The Validome validation tool.

1.13 What You Should Know

This first chapter introduces you to the JavaScript programming language, its history, why it is important in Web development, and how it fits into a Web page. Before going further, you should know:

1. The difference between a compiled and scripting language.
2. Where JavaScript is defined, the client or server?
3. What JavaScript is used for.
4. How JavaScript makes a page interactive.
5. Where JavaScript programs are stored.
6. What a JavaScript program looks like.
7. What Ajax stands for and an example of how it is used.
8. What W3C stands for.
9. Why the DOM was standardized.
10. What is meant by unobtrusive JavaScript.
11. Where on the Web to find a good HTML tutorial.

Exercises

1. Describe the life cycle of a Web page.
2. What browser are you using? What version? How do you know?
3. What is an example of a JavaScript event handler? Copy Example 1.2 into your editor and run the program in your browser.
4. What is the difference between JavaScript and JScript?
5. Where do JavaScript tags go in an HTML page? Does JavaScript understand HTML?
6. What is the DOM?
7. Define the three layers of a Web page.
8. How do you set up JavaScript in an external file?
9. Write a JavaScript program that prints a welcome message in a large blue font. Check to see if JavaScript is enabled. Use comments to explain what you are doing.

This page intentionally left blank

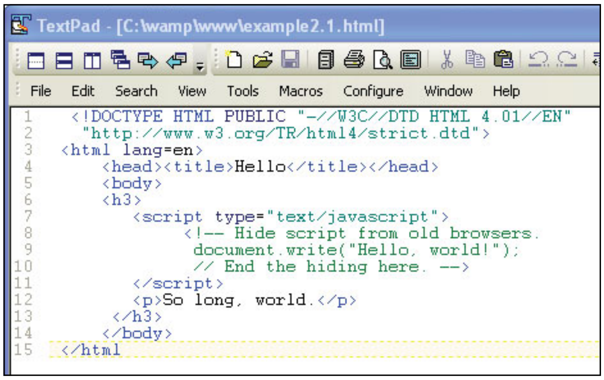
chapter

2

Script Setup

2.1 The HTML Document and JavaScript

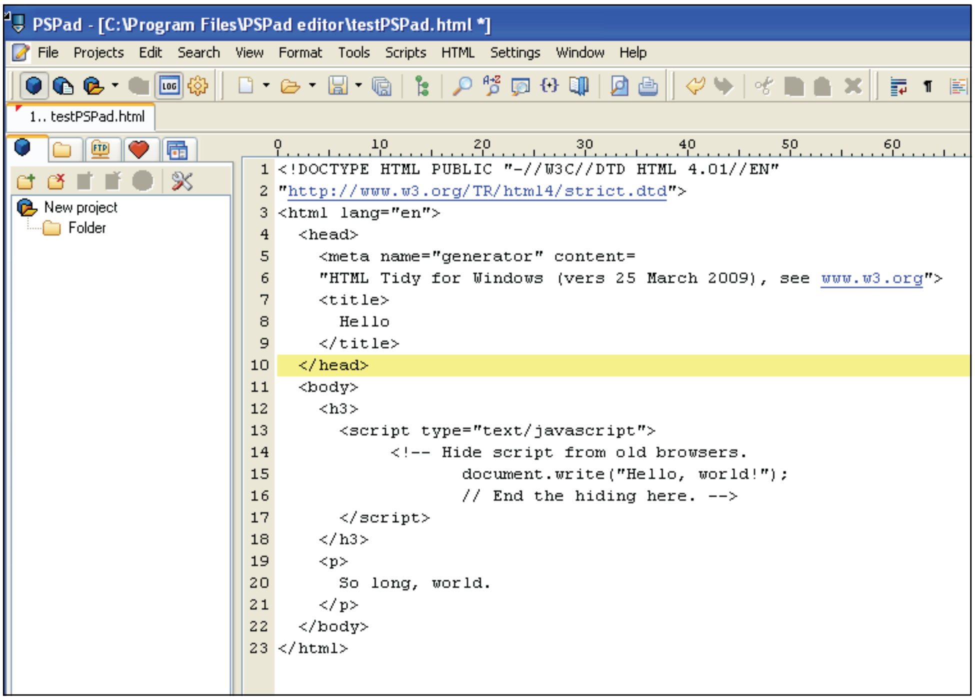
Unlike Perl and Python scripts, JavaScript scripts are not stand-alone programs. They are run in the context of an HTML document. When programming on the client side, the first step will be to create an HTML document in your favorite text editor, such as UNIX vi or emacs, or Windows Notepad,[1] WordPad, or TextPad (see Figure 2.1). There are a number of popular integrated development environments (IDEs) such as NetBeans, Komodo Edit, and Eclipse, with highlighting, validation, debugging features, and so on you might prefer to use. A list of recommended IDEs can be found at *http://JavaScript-ide.software.informer.com/downloads/*. Because the file you create is an HTML document, its name must include either an *.html* or *.htm* extension. JavaScript programs can be embedded within the HTML document between the *<script>* and *</script>* tags. Figures 2.1 and 2.2 show an HTML file containing JavaScript code.



```
  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
<html lang=en>
    <head><title>Hello</title></head>
    <body>
    <h3>
       <script type="text/javascript">
            <!-- Hide script from old browsers.
             document.write("Hello, world!");
             // End the hiding here. -->
       </script>
       <p>So long, world.</p>
      </h3>
    </body>
</html
```

Figure 2.1 JavaScript in TextPad editor.

1. If you are using Windows Notepad, be sure to turn off word wrap (under the Format menu) to avoid errors in your program.


PSPad - [C:\Program Files\PSPad editor\testPSPad.html *]
File Projects Edit Search View Format Tools Scripts HTML Settings Window Help
1.. testPSPad.html
New project
Folder

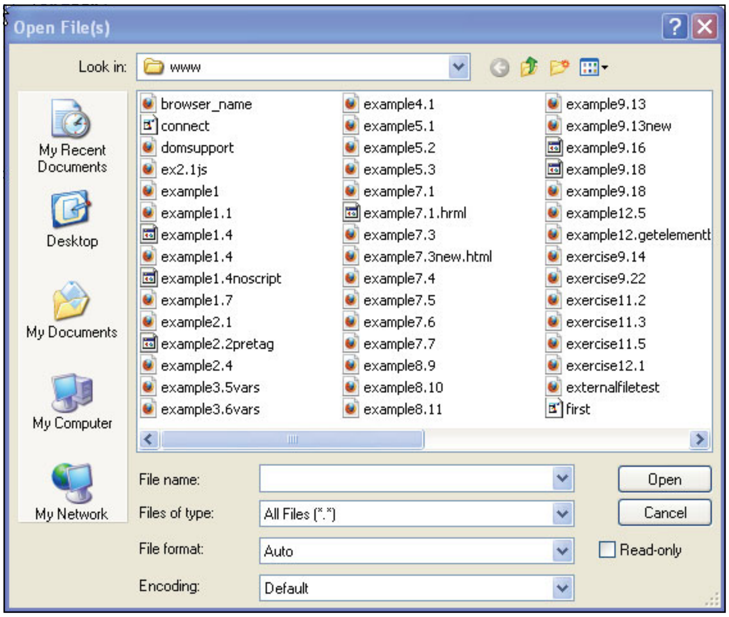
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html lang="en">
  <head>
    <meta name="generator" content=
    "HTML Tidy for Windows (vers 25 March 2009), see www.w3.org">
    <title>
      Hello
    </title>
  </head>
  <body>
    <h3>
      <script type="text/javascript">
           <!-- Hide script from old browsers.
                  document.write("Hello, world!");
                  // End the hiding here. -->
      </script>
    </h3>
    <p>
      So long, world.
    </p>
  </body>
</html>
```

Figure 2.2 JavaScript in PSPad Editor, a popular free IDE with many features.

2.1.1 Script Execution

Because a JavaScript program is embedded in an HTML document, you will execute it in your browser window. If you are using an IDE, the browser is part of the integrated environment, but either way, you can execute a JavaScript program directly in the browser. If using Mozilla Firefox, Opera, or Internet Explorer, follow these instructions:

1. Go to the File menu and open the HTML file by browsing for the correct one.



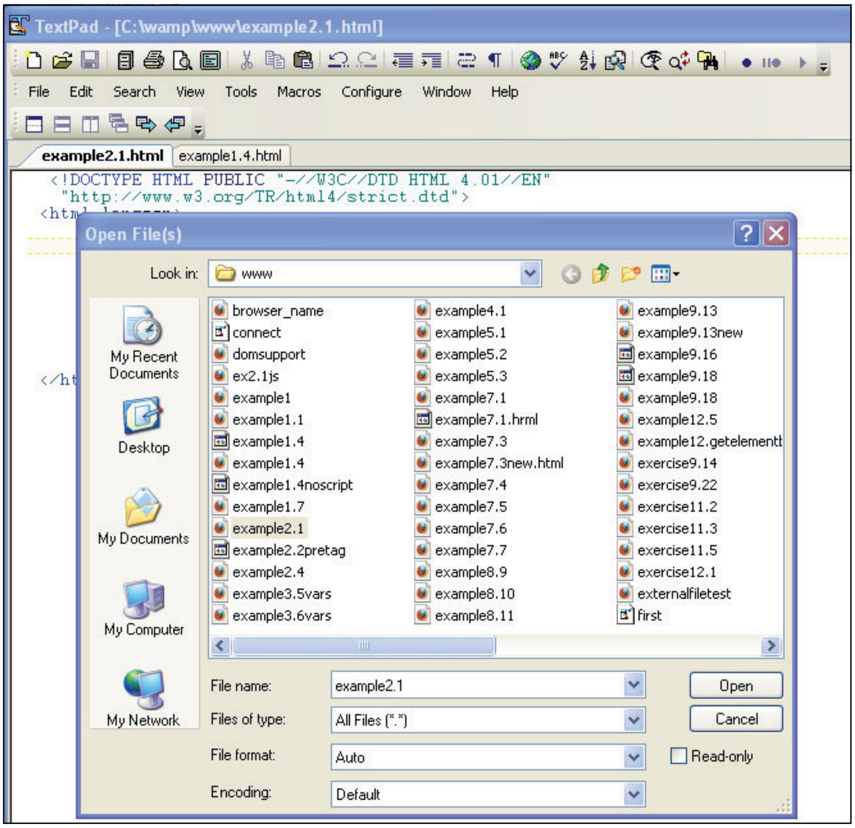
Look in: www

browser_name
connect
domsupport
ex2.1js
example1
example1.1
example1.4
example1.4
example1.4noscript
example1.7
example2.1
example2.2pretag
example2.4
example3.5vars
example3.6vars
example4.1
example5.1
example5.2
example5.3
example7.1
example7.1.hrml
example7.3
example7.3new.html
example7.4
example7.5
example7.6
example7.7
example8.9
example8.10
example8.11
example9.13
example9.13new
example9.16
example9.18
example9.18
example12.5
example12.getelementt
exercise9.14
exercise9.22
exercise11.2
exercise11.3
exercise11.5
exercise12.1
externalfiletest
first

File name:
Files of type: All Files (*.*)
File format: Auto
Encoding: Default

2. All files in the www folder are displayed. (The .html extension on the files will appear in the browser's URL.)

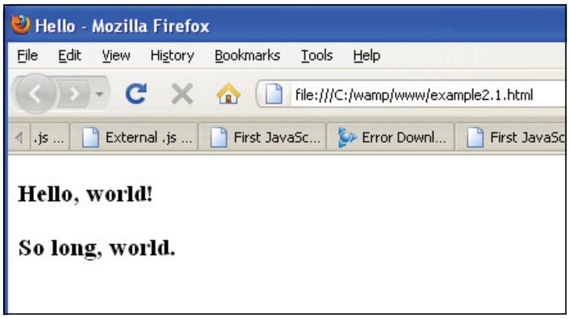
3. The file example 2.1.html is highlighted. (The .html extension is not visible.)



```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
```

browser_name, connect, domsupport, ex2.1js, example1, example1.1, example1.4, example1.4, example1.4noscript, example1.7, example2.1, example2.2pretag, example2.4, example3.5vars, example3.6vars, example4.1, example5.1, example5.2, example5.3, example7.1, example7.1.hrml, example7.3, example7.3new.html, example7.4, example7.5, example7.6, example7.7, example8.9, example8.10, example8.11, example9.13, example9.13new, example9.16, example9.18, example9.18, example12.5, example12.getelementt, exercise9.14, exercise9.22, exercise11.2, exercise11.3, exercise11.5, exercise12.1, externalfiletest, first

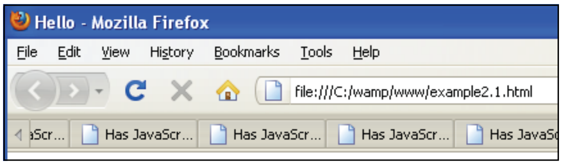
File name: example2.1
Files of type: All Files (*.*)
File format: Auto
Encoding: Default



Hello, world!

So long, world.

4. Or you can type the URL (complete address) in the navigation bar of your browser as shown here. (Note the .html extension appears in the URL.)


Hello - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/wamp/www/example2.1.html
Has JavaScr...

2.2 Syntactical Details

Rules, rules, rules. Just like English, French, or Chinese, all programming languages have their rules. Many of the rules are similar and many have individual quirks, but to do anything at all, you have to obey the rules, or your program simply won't work. If you have experience programming in other languages, you will find the JavaScript rules and syntax quite familiar. When you write JavaScript programs, you have to deal with HTML rules (and CSS rules), as well as JavaScript rules, because JavaScript does not stand alone.

2.2.1 Case Sensitivity

The HTML tags in a document are not case sensitive. If you type the *title* tag as <*title*>, <*Title*>, <*TItle*>, or any combination of upper or lowercase characters, the HTML renderer will not care. But JavaScript names, such as variables, keywords, objects, functions, and so on, are case sensitive. If, for example, you spell the Boolean value *true* with any uppercase letters (e.g., TrUE), JavaScript will not recognize it and will produce an error or simply ignore the JavaScript code. Although most names favor lowercase, some JavaScript names use a combination of upper and lowercase (e.g., *onClick*, *Math.floor*, *Date.getFullYear*).

2.2.2 Free Form and Reserved Words

JavaScript ignores whitespace (i.e., spaces, tabs, and newlines) if the whitespace appears between words. For example, a function name, such as *onMouseOver*(), *toLowerCase*(), or *onClick*(), cannot contain whitespace even though it consists of more than one word.

```
1. var name="Tom";          1 and 2 are equivalent statements
2. var    name   =
           "Tom";

3. onMouseOver()            3 and 4 are not the same
4. on  Mouse Over()
```

Whitespace is preserved when it is embedded within a string or regular expression. For example, the whitespace in the string, *"Hello   there"* will be preserved because it is enclosed within double quotes. Of course, you can't break up a word such as *switch*, *if*, *else*, *window*, *document*, and so on, because it would no longer be the same word. Because extra whitespace is ignored by the JavaScript interpreter, you are free to indent, break lines, and organize your program so that it is easier to read and debug.

There are a number of reserved words (also called keywords) in JavaScript. Being reserved means that keywords are special vocabulary for the JavaScript language and cannot be used by programmers as identifiers for variables, functions and labels, and the like. Words such as *if*, *for*, *while*, *return*, *null*, and *typeof* are examples of reserved words. Table 2.1 gives a list of reserved words.

Table 2.1 Reserved Keywords

abstract	boolean	break	byte	case	catch
char	class	const	continue	default	delete
do	double	else	extends	false	final
finally	float	for	function	goto	if
implements	import	in	instanceof	int	interface
long	native	new	null	package	private
protected	public	return	short	static	super
switch	synchronized	this	throw	throws	transient
true	try	typeof	var	void	volatile
while	with				

2.2.3 Statements and Semicolons

Just like sentences (which represent complete thoughts) in the English language, JavaScript statements are made up of expressions. The statements are executed top down, one statement at a time. If there are multiple statements on a line, the statements must be separated by semicolons. Although not a rule, it is good practice to terminate all statements with a semicolon to make it clear where the statement ends. Because JavaScript is free form, as long as statements are terminated with a semicolon, the lines can be broken, contain whitespace, and so on. A statement results in some action unless the statement is a null statement, in which case it does nothing.

The following two lines are both technically correct:

```
var name = "Ellie"     <- no semicolon, valid
var name = "Ellie";    <- better
```

The following line is incorrect:

```
var name = "Ellie"  document.write("Hi "+name); <- wrong, two statements
```

It should be:

```
var name = "Ellie";  document.write("Hi " + name); <- semicolon needed
                                                      to separate two
                                                      statements on the
                                                      same line
```

If the statements are grouped in a block of curly braces, they act as a single statement.

```
if ( x > y) { statement;  statement; } <- Statements enclosed in curly
                                          braces act as a single
                                          statement
```

2.2.4 Comments

A comment is text that describes what the program or a particular part of the program is trying to do and is ignored by the JavaScript interpreter. Comments are used to help you and other programmers understand, maintain, and debug scripts. JavaScript uses two types of comments: single-line comments and block comments.

Single-line comments start with a double slash:

```
// This is a comment
```

For a block of comments, use the /* */ symbols:

```
/* This is a block of comments
   that continues for a number of lines
*/
```

2.2.5 The *<script>* Tag

JavaScript programs must start and end with the HTML *<script>* and *</script>* tags, respectively. Everything within these tags is considered JavaScript code, nothing else. The script tag can be placed anywhere within an HTML document. If you want the JavaScript code to be executed before the page is displayed, it is placed between the *<head>* and *</head>* tags. This, for example, is where function definitions are placed (see Chapter 7, "Functions"). If the script performs some action pertaining to the body of the document, then it is placed within the *<body>* and *</body>* tags. A document can have multiple *<script>* tags, each enclosing any number of JavaScript statements.

FORMAT

```
<script>
  JavaScript statements...
</script>
```

EXAMPLE 2.1

```
<script>
  document.write("Hello, world!<br />");
</script>
```

Attributes. The *<script>* tag also has **attributes** to modify the behavior of the tag. The attributes are

- *language*
- *type*
- *src*

Any JavaScript-enabled browser can identify that the scripting language is JavaScript, if the *language* attribute is set to *JavaScript*[2] rather than, for example, *VBScript* or *JScript*. You normally set the language attribute as follows:

```
<script language="JavaScript">
```

According to the W3C recommendation, the value assigned to this attribute is an identifier for the scripting language, but because these identifiers are not standard, this attribute has been deprecated in favor of the *type* attribute.

The *language* attribute can be assigned a version number to specify what version of JavaScript is supported to view the page. If the browser doesn't recognize the version, the script will be totally ignored. You shouldn't have to worry about this if you are using the latest version of a particular browser, but just in case, here's how you specify a version number.

```
<script language="JavaScript1.5">
</script>
```

The *type* attribute is used to specify both the scripting language and the Internet content type. It is used mainly to validate JavaScript as part of a well-formed document and is the preferred way to start JavaScript in all modern browsers.

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

2. Although common to most scripts, the *language* attribute has been deprecated as of HTML 4.0 in favor of the *type* attribute.

The *src* attribute is used when the JavaScript code is in an external file, the file name ending with a *.js* extension. The *src* attribute is assigned the name of the file, which can be prefixed with its location (e.g., a directory tree or URL).

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

<script type="text/javascript"
   src="directory/sample.js">
</script>

<script type="text/javascript"
   src="http://hostname/sample.js">
</script>
```

2.3 Generating HTML and Printing Output

When you create a program in any language, the first thing you want to see is the output of the program displayed on a screen. In the case of JavaScript, you'll see your output in the browser window. Of course, browsers use HTML to format output. Although JavaScript doesn't understand HTML per se, it can generate HTML output with its built-in methods, *write()* and *writeln()*.

2.3.1 Strings and String Concatenation

A string is a character or set of characters enclosed in matching quotes. Because the methods used to display text take strings as their arguments, this is a good time to talk a little about strings. See Chapter 9, "JavaScript Core Objects," for a more complete discussion. All strings must be placed within a matched set of either single or double quotes; for example:

`"this is a string"` or `'this is a string'`

Double quotes can hide single quotes; for example:

```
"I don't care"
```

And single quotes can hide double quotes; for example:

```
'He cried, "Ahoy!"'
```

Either way, the entire string is enclosed in a set of matching quotes.

Concatenation is caused when two strings are joined together. The plus (+) sign is used to concatenate strings; for example:

`"hot" + "dog"` or `"San Francisco" + "<br />"`

For more information on strings, see Chapter 3, “The Building Blocks: Data Types, Literals, and Variables.”

2.3.2 The *write()* and *writeln()* Methods

One of the most important features of client-side JavaScript is its ability to generate pages dynamically. Data, text, and HTML itself can be written to the browser on the fly. The *write()* method is a special kind of built-in JavaScript function used to output HTML to the document as it is being parsed. When generating output with *write()* and *writeln()*, put the text in the body of the document (rather than in the header) at the place where you want the text to appear when the page is loaded.

Method names are followed by a set of parentheses. They are used to hold the arguments. These are messages that will be sent to the methods, such as a string of text, the output of a function, or the results of a calculation. Without arguments, the *write()* and *writeln()* methods would have nothing to write.

JavaScript defines the current document (i.e., the HTML file that contains the script) as a document object. (You will learn more about objects later.) For now, whenever you refer to the document object, the object name is appended with a dot and the name of the method that will manipulate the document object. In the following example the *write()* method must be prepended with the name of the document object and a period. The browser will display this text in the document’s window. The syntax is

```
document.write("Hello to you");
```

The *writeln()* method is essentially just like the *write()* method, except when the text is inserted within HTML *<pre>* or *<xmp>* tags, in which case *writeln()* will insert a newline at the end of the string. The HTML *<pre>* tag is used to enclose preformatted text. It results in “what you see is what you get.” All spaces and line breaks are rendered literally, in a monopitch typeface. The *<xmp>* tag is an obsolete HTML tag that functions much like the *<pre>* tag.

EXAMPLE 2.2

```
   <html>
     <head><title>Printing Output</title></head>
     <body bgcolor="yellow" text="blue">
       <big>
       <b>Comparing the <em>document.write</em> and
                      <em>document.writeln</em> methods</b><br />
       <script type="text/javascript">
1        document.write("One, ");  // No newline
2        document.writeln("Two, ");
```

EXAMPLE 2.2 (CONTINUED)

```
            document.writeln("Three, ");
3           document.write("Blast off....<br />");  // break tag
4           //document.write("The browser you are using is " +
            //                 navigator.userAgent + "<br />");
5         </script>
6         <pre>
7         <script type="text/javaScript">
            /*Lines are broken due to size of this page. If you cut
              and paste these programs into an editor, make sure
              strings start and end with qutoes!!!*/
8           document.writeln("With the <em>HTML &lt;pre&gt;
                             </em> tags, ");
            document.writeln("the <em>writeln</em> method produces a
                              newline.");
            document.writeln("Slam");
            document.writeln("Bang");
            document.writeln("Dunk!");
9         </script>
10        </pre>
          </big>
        </body>
      </html>
```

EXPLANATION

1 The *document.write()* method does not produce a newline at the end of the string it displays. HTML tags are sent to the HTML renderer as the lines are parsed.

2 The *document.writeln()* method doesn't produce a newline either, unless it is in an HTML *<pre>* tag.

3 Again, the *document.write()* method does not produce a newline at the end of the string. The *<br>* tag is added to produce the line break.

4 The *document.write()* method does not produce a newline. The *<br />* tag takes care of that. *userAgent* is a special *navigator* property that tells you about your browser.

5 The first JavaScript program ends here.

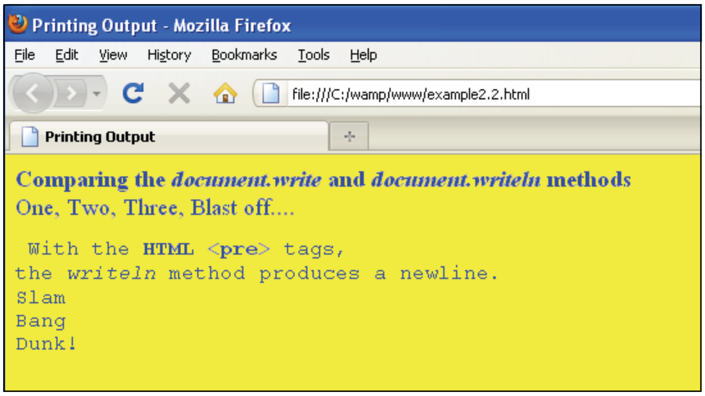
6 The HTML *<pre>* tag starts a block of preformatted text; that is, text that ignores formatting instructions and fonts.

7 This tag starts the JavaScript code.

8 When enclosed in a *<pre>* tag, the *writeln()* method will break each line it prints with a newline; otherwise, it behaves like the *write()* method (i.e., you will have to add a *<br>* tag to get a newline).

9 This tag marks the end of the JavaScript code.

10 This tag marks the end of preformatted text. The output is shown in Figure 2.3.


Printing Output - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/wamp/www/example2.2.html
Printing Output

Comparing the *document.write* and *document.writeln* methods
One, Two, Three, Blast off....

```
 With the HTML <pre> tags,
the writeln method produces a newline.
Slam
Bang
Dunk!
```

Figure 2.3 The output from Example 2.2 demonstrates the difference between the *document.write()* and *document.writeln()* methods.

2.4 About Debugging

Have you ever tried to draw a picture or do your resume for the first time without a mistake either in the layout, order, type, style, or whatever? In any programming language, it's the same story, and JavaScript is no exception. It's especially tricky with JavaScript because you have to consider the HTML as well as the JavaScript code when your page doesn't turn out right. You might get errors on the console or get a totally blank page. Finding errors in a script can get quite frustrating without proper debugging tools. Before we go any further, this is a good time to get acquainted with some of the types of errors you might encounter.

2.4.1 Types of Errors

Load or Compile Time. Load-time errors are the most common errors and are caught by JavaScript as the script is being loaded. These errors will prevent the script from running at all. Load-time errors are generally caused by mistakes in syntax, such as missing parentheses in a function or misspelling a keyword. You might have typed a string of text and forgotten to enclose the string in quotes, or you might have mismatched the quotes, starting with single quotes but ending with double quotes.

Runtime. Runtime errors, as the name suggests, are those errors that occur when the JavaScript program actually starts running. An example of a runtime error would be if your program references an object or variable that doesn't exist, or you put some code between the *<head></head>* tags and it should have been placed within the *<body></body>* tags, or you referenced a page that doesn't exist.

Logical. Logical errors are harder to find because they imply that you didn't anticipate an event or that you inadvertently misused an operator, but your syntax was okay.

For example, if you are checking to see if two expressions are equal, you should use the == equality operator, not the = assignment operator.

2.5 Debugging Tools

To see your where errors have occurred in your JavaScript programs, modern browsers provide an error console window.

Table 2.2 Browser Error Console

Browser	***How to Invoke Error Console***
Internet Explorer	Double-click the little yellow triangle in the left corner
Firefox	Tools/Error Console
Safari	Develop/Show Error Console
Opera9	Tools/Advanced/Error Console script Options/

2.5.1 Firefox

Error Console. You can bring up the error console for Firefox by going to "Tools/Error Console" The Console displays the lines containing the errors. Leave the console open and watch your errors build up. There is a "Clear" option to refresh the error console window. The following JavaScript program contains an error that will be displayed in the Error Console window as shown in Figure 2.4.

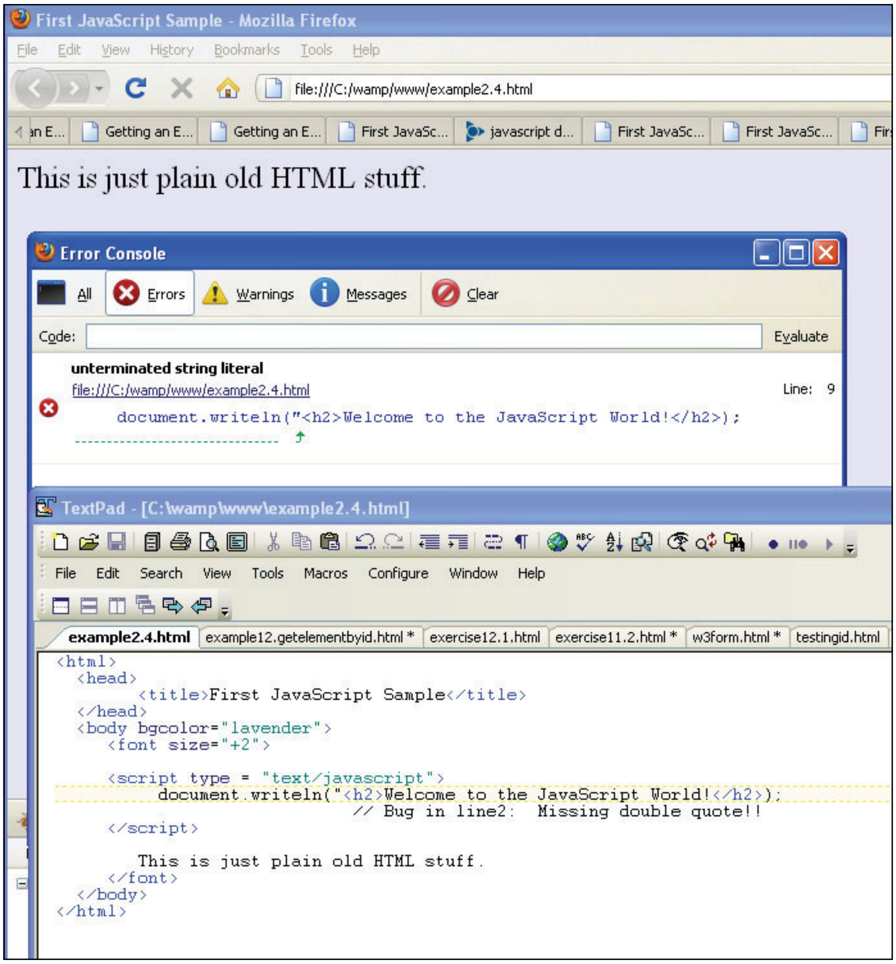
EXAMPLE 2.3

```
      <html>
        <head>
          <title>First JavaScript Sample</title>
        </head>
        <body bgcolor="lavender">
          <font size="+2">

1         <script type = "text/javascript">
2           document.writeln("<h2>Welcome to the JavaScript World!</h2>);
                             // Bug in line2:  Missing double quote!!
          </script>

            This is just plain old HTML stuff.
          </font>
        </body>
      </html>
```

EXPLANATION

1 JavaScript code starts here.

2 In this line, the string starts with a double quote, but doesn't terminate with one. Because the quotes are not matched, JavaScript produces an error. Each browser has a way of handling error messages. Figure 2.4 uses the Firefox error console to detect the error.

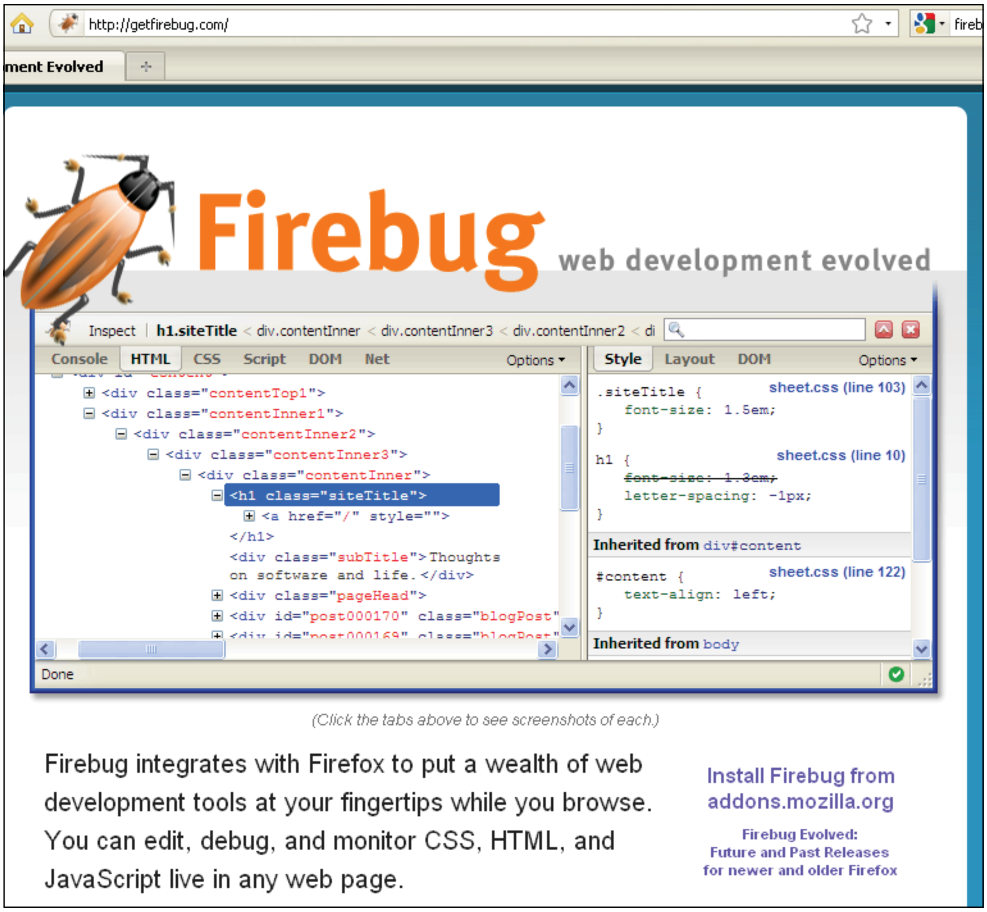

First JavaScript Sample - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/wamp/www/example2.4.html
This is just plain old HTML stuff.
Error Console
All Errors Warnings Messages Clear
Code: Evaluate
TextPad - [C:\wamp\www\example2.4.html]
File Edit Search View Tools Macros Configure Window Help
example2.4.html example12.getelementbyid.html * exercise12.1.html exercise11.2.html * w3form.html * testingid.html

```
unterminated string literal
file:///C:/wamp/www/example2.4.html                                    Line: 9
    document.writeln("<h2>Welcome to the JavaScript World!</h2>);
------------------------------- ^
```

```
<html>
  <head>
        <title>First JavaScript Sample</title>
  </head>
  <body bgcolor="lavender">
     <font size="+2">

     <script type = "text/javascript">
          document.writeln("<h2>Welcome to the JavaScript World!</h2>);
                              // Bug in line2:  Missing double quote!!
     </script>

        This is just plain old HTML stuff.
     </font>
  </body>
</html>
```

Figure 2.4 Firefox Error Console (go to Firefox Tools/Error Console).

Firebug. Firebug is a Firefox extension that has become very popular with Web developers for editing, debugging, and monitoring CSS, HTML, and JavaScript live in any Web page (see Figures 2.5 and 2.6). It is easy to download and can be found in the Firefox Tools menu.



```
⊞ <div class="contentTop1">
⊟ <div class="contentInner1">
    ⊟ <div class="contentInner2">
        ⊟ <div class="contentInner3">
            ⊟ <div class="contentInner">
                ⊟ <h1 class="siteTitle">
                    ⊞ <a href="/" style="">
                  </h1>
                  <div class="subTitle">Thoughts
                  on software and life.</div>
                ⊞ <div class="pageHead">
                ⊞ <div id="post000170" class="blogPost"
```

```
.siteTitle {                sheet.css (line 103)
   font-size: 1.5em;
}

h1 {                        sheet.css (line 10)
   ~~font-size: 1.3em;~~
   letter-spacing: -1px;
}
Inherited from div#content
#content {                  sheet.css (line 122)
   text-align: left;
}
Inherited from body
```

(Click the tabs above to see screenshots of each.)

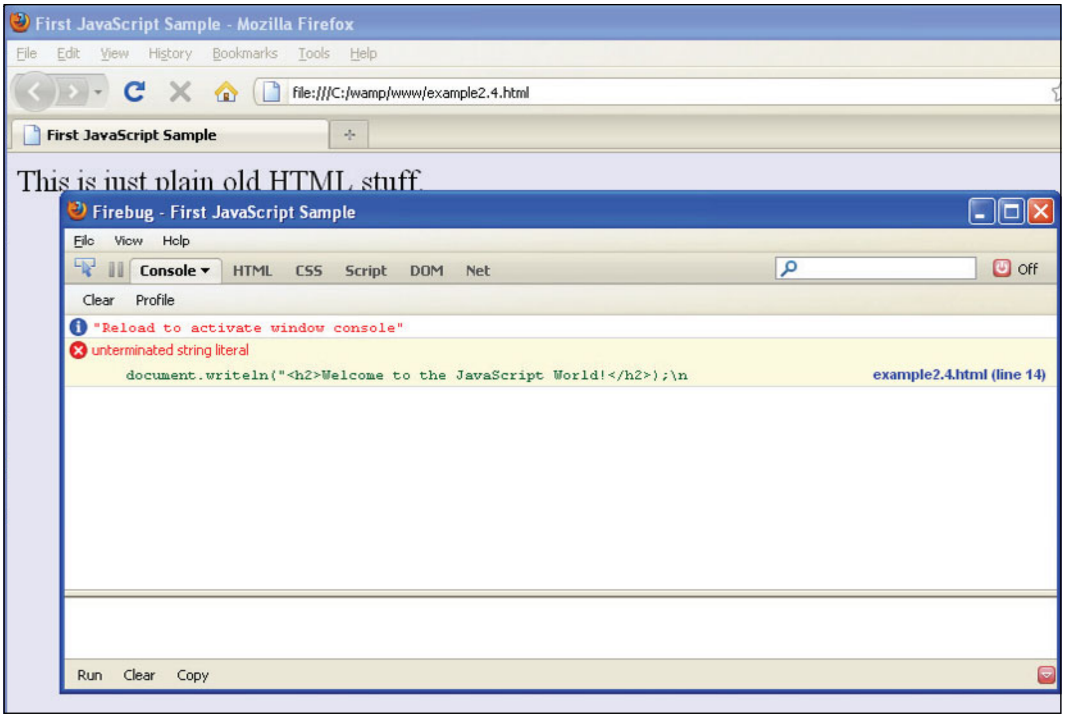
Firebug integrates with Firefox to put a wealth of web development tools at your fingertips while you browse. You can edit, debug, and monitor CSS, HTML, and JavaScript live in any web page.

Install Firebug from addons.mozilla.org

Firebug Evolved:
Future and Past Releases
for newer and older Firefox

Figure 2.5 The Firebug Web site.

You can also use a version of Firebug in Internet Explorer, Opera, and Safari called Firebug Lite. See *http://getfirebug.com/lite.html*.


First JavaScript Sample - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/wamp/www/example2.4.html
First JavaScript Sample
Firebug - First JavaScript Sample
File View Help
Console HTML CSS Script DOM Net
Off
Clear Profile
Run Clear Copy

This is just plain old HTML stuff.

```
"Reload to activate window console"
unterminated string literal
    document.writeln("<h2>Welcome to the JavaScript World!</h2>);\n        example2.4.html (line 14)
```

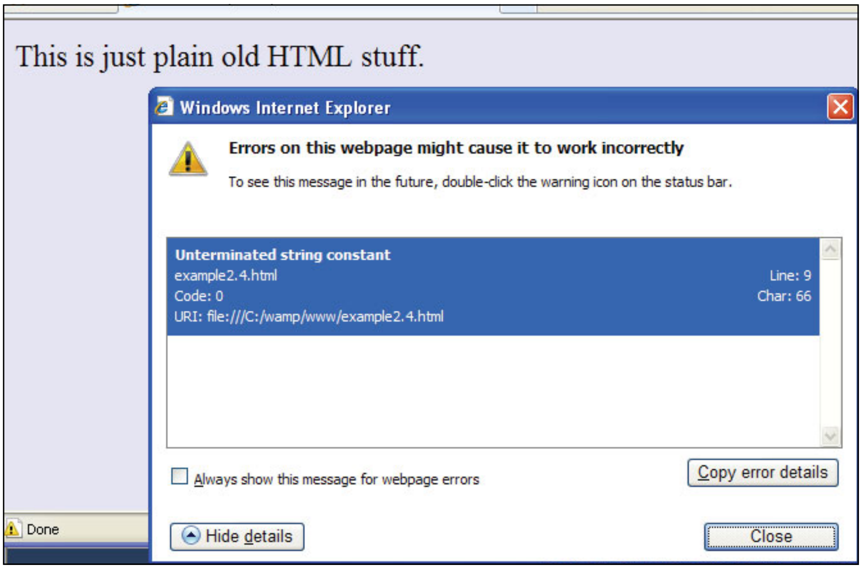
Figure 2.6 The Firebug Debugging window.

2.5.2 Debugging in Internet Explorer 8

When an error occurs in your JavaScript program, a little yellow triangle appears in the bottom left corner of the browser window. If you double-click the triangle, a debugging window opens explaining the error and the line number where it occurred (see Figure 2.7).

Internet Explorer Developer Tools. Every installation of Internet Explorer 8 comes with the Developer Tools for debugging JavaScript (Microsoft JScript), HTML, and CSS on the fly. It comes with a plethora of features including the ability to control script execution, set break points, inspect variables, profile performance, edit and prototype new designs, and so on. See *http://msdn.microsoft.com/en-us/library/dd565628(VS.85).aspx*.

To start debugging your JavaScript programs, open the Developer Tools and switch to the Script tab, then click Start Debugging. When starting the debugging process, the Developer Tools will refresh the page and you will have all the functionality you expect from a debugger (see Figure 2.8). Once you are done, click Stop Debugging. Go to Internet Explorer Tools/Developer Tools and the debugger window will appear. Click Script, and then restart your program in the browser.



This is just plain old HTML stuff.

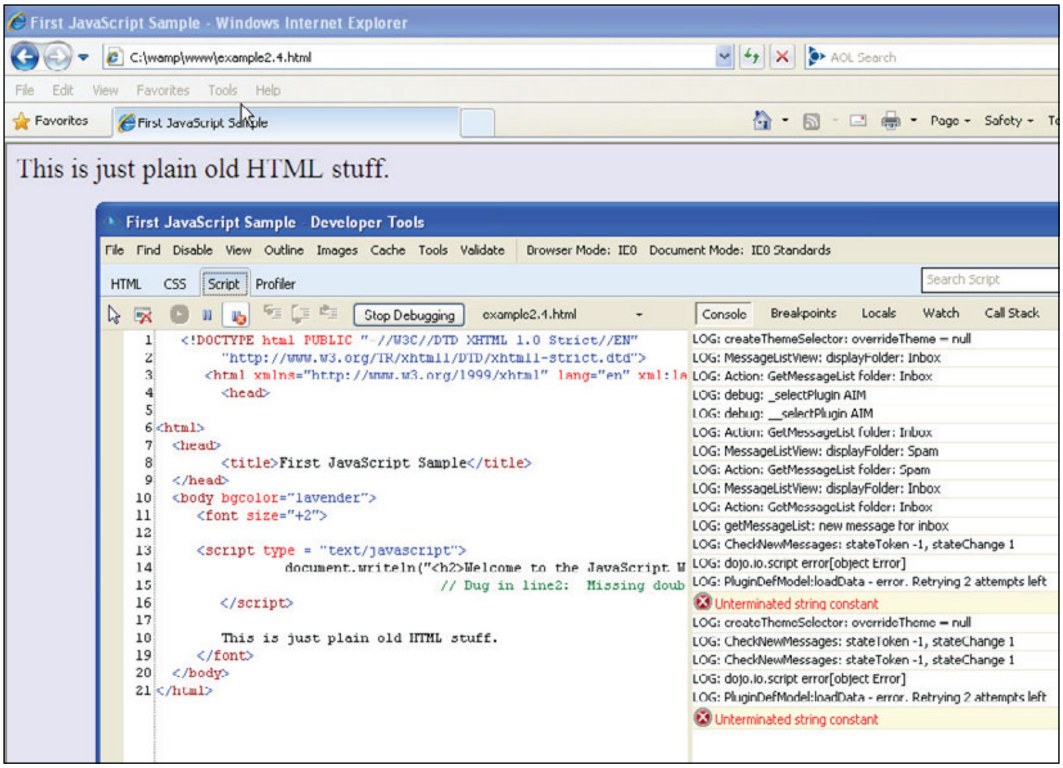
Errors on this webpage might cause it to work incorrectly

To see this message in the future, double-click the warning icon on the status bar.

```
Unterminated string constant
example2.4.html                                   Line: 9
Code: 0                                           Char: 66
URI: file:///C:/wamp/www/example2.4.html
```

Windows Internet Explorer
Always show this message for webpage errors
Copy error details
Hide details
Close
Done

Figure 2.7 Internet Explorer 8 after clicking the little yellow triangle in the bottom left corner.



First JavaScript Sample - Windows Internet Explorer
C:\wamp\www\example2.4.html
AOL Search
File Edit View Favorites Tools Help
Favorites
First JavaScript Sample
Page Safety
First JavaScript Sample Developer Tools
File Find Disable View Outline Images Cache Tools Validate Browser Mode: IE8 Document Mode: IE8 Standards
HTML CSS Script Profiler
Search Script
Stop Debugging
example2.4.html
Console Breakpoints Locals Watch Call Stack

This is just plain old HTML stuff.

```
  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
    <html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:la
      <head>

<html>
  <head>
      <title>First JavaScript Sample</title>
  </head>
  <body bgcolor="lavender">
     <font size="+2">

     <script type = "text/javascript">
            document.writeln("<h2>Welcome to the JavaScript W
                              // Bug in line2:  Missing doub
      </script>

      This is just plain old HTML stuff.
     </font>
  </body>
</html>
```

```
LOG: createThemeSelector: overrideTheme = null
LOG: MessageListView: displayFolder: Inbox
LOG: Action: GetMessageList folder: Inbox
LOG: debug: _selectPlugin AIM
LOG: debug: __selectPlugin AIM
LOG: Action: GetMessageList folder: Inbox
LOG: MessageListView: displayFolder: Spam
LOG: Action: GetMessageList folder: Spam
LOG: MessageListView: displayFolder: Inbox
LOG: Action: GetMessageList folder: Inbox
LOG: getMessageList: new message for inbox
LOG: CheckNewMessages: stateToken -1, stateChange 1
LOG: dojo.io.script error[object Error]
LOG: PluginDefModel:loadData - error. Retrying 2 attempts left
Unterminated string constant
LOG: createThemeSelector: overrideTheme = null
LOG: CheckNewMessages: stateToken -1, stateChange 1
LOG: CheckNewMessages: stateToken -1, stateChange 1
LOG: dojo.io.script error[object Error]
LOG: PluginDefModel:loadData - error. Retrying 2 attempts left
Unterminated string constant
```

Figure 2.8 Internet Explorer 8 Developer Tools (go to the Tools menu and then Developer Tools).

2.5.3 The *JavaScript:* URL Protocol

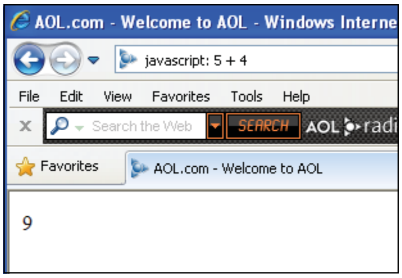
For simple debugging or testing code, you can use the URL pseudoprotocol, *JavaScript:* followed by any valid JavaScript expression or expressions separated by semicolons. The result of the expression is returned as a string to your browser window, as shown in Example 2.4 and Figures 2.9, 2.10, and 2.11.

FORMAT

```
JavaScript: expression
```

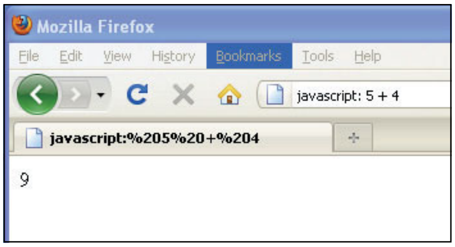
EXAMPLE 2.4

```
JavaScript: 5 + 4
```



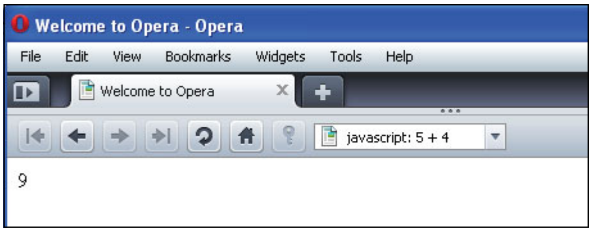
```
9
```

Figure 2.9 Internet Explorer and the *JavaScript:* protocol.



```
9
```

Figure 2.10 Mozilla Firefox and the *JavaScript:* protocol.



```
9
```

Figure 2.11 Opera and the *JavaScript:* protocol.

2.6 JavaScript and Old or Disabled Browsers

2.6.1 Hiding JavaScript from Old Browsers

Is JavaScript Enabled? The answer is most probably "yes." There are many versions of browsers available to the public and the vast majority of the public uses Firefox, Opera or Internet Explorer. So why worry? Well, just because a browser supports JavaScript does not mean that everyone has JavaScript enabled. There are also some older text browsers that don't support JavaScript, but today it's more likely that JavaScript has been disabled for security reasons or to avoid cookies than because the browser is old. Cell phones, Palm handhelds, and speech browsers for the visibly disabled provide browser support, but do not necessarily have JavaScript. There has to be some alternative way to address those Web browsers (see *http://www.internet.com*).

Hiding JavaScript. If you put a script in a Web page, and the browser is old and doesn't know what a <*script*> tag is, the JavaScript code will be treated as regular HTML. But if you enclose JavaScript code within HTML comment tags, it will be invisible to the HTML renderer and, therefore, ignored by browsers not supporting JavaScript. If the browser has JavaScript enabled, then any HTML tags (including HTML comments) inserted between the <*script*> </*script*> tags will be ignored. Hiding JavaScript in comment tags is a common practice used in JavaScript programs. It is introduced here so that when you notice these embedded comments in other's programs, you will understand why. See Example 2.5.

EXAMPLE 2.5

```
   <html>
     <head><title>Old Browsers</title></head>
     <body><font color="0000F">
       <div align=center>
1        <script type="text/javascript">
2          <!-- Hiding JavaScript from Older Browsers
3          document.write("<h2>Welcome to Maine!</h2>");
4          // Stop Hiding JavaScript from Older Browsers -->
5        </script>
         <img src="BaileyIsland.jpg" width="400" height="300" border=1>
         <br />Bailey's Island
       </div></font>
     </body>
   </html>
```

EXPLANATION

1 The JavaScript program starts here. Browsers that don't support JavaScript will skip over this opening <*script*> tag.

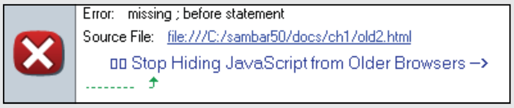
2 The <!-- symbol is the start of an HTML comment and will continue until --> is reached. Any browser not supporting JavaScript will treat this whole block as a comment. JavaScript itself uses two slashes or C-style syntax, /* */, and will ignore the HTML comment tags.

Continues

EXPLANATION (CONTINUED)

3 The *document.write* method displays this line in the page. Any HTML tags inserted in the quoted strings will be handled by the HTML renderer. JavaScript does not know how to interpret HTML by itself. If the browser supports JavaScript, the line *Welcome to Maine!* will appear just above the image. If the browser does not support JavaScript, or has it disabled, this section of code is ignored. See the two examples of output shown in Figures 2.12 and 2.13.

4 This line starts with two slashes, the start of a JavaScript comment. This is done so that if JavaScript is interpreting this section, it won't see the HTML closing comment tag, -->. Why don't we want JavaScript to see the closing tag if it could see the opening tag? Because JavaScript would see the double dash as one of its special operators, and produce an error. Netscape's error:



```
Error:   missing ; before statement
Source File:   file:///C:/sambar50/docs/ch1/old2.html
    □□ Stop Hiding JavaScript from Older Browsers -->
--------  ^
```

5 The JavaScript program ends here with its closing *</script>* tag.

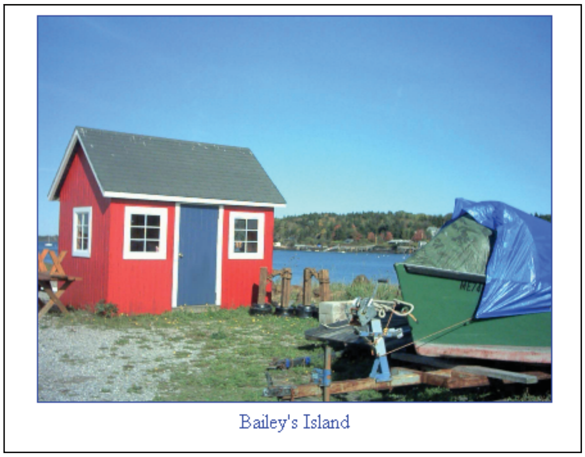


Figure 2.12 Example 2.5 output in a JavaScript-disabled browser.

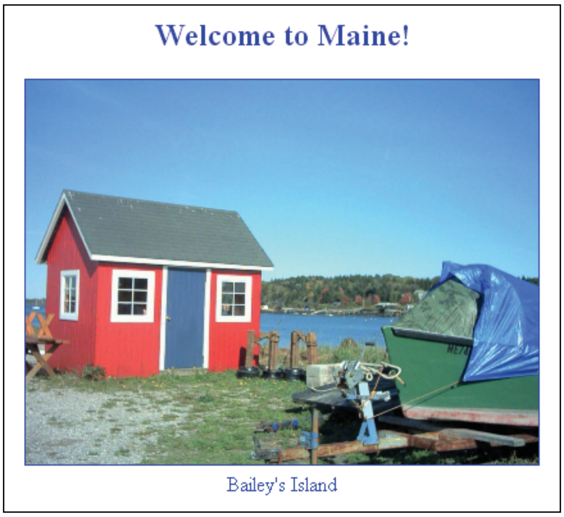


Figure 2.13 Example 2.5 output in a JavaScript-enabled browser.

The <*noscript*> Tag. Modern browsers provide a set of tags called <*noscript*></*noscript*> that enable you to provide alternative information to browsers that are either unable to read JavaScript or have it turned off. All JavaScript-enabled browsers recognize the <*noscript*> tag. They will just ignore whatever is between <*noscript*> and</*noscript*>. Browsers that do not support JavaScript do not recognize the <*noscript*> tags. They will ignore the tags but will display whatever is in between them. See Example 2.6.

EXAMPLE 2.6

```
    <html>
      <head>
        <title>Has JavaScript been turned off?</title>
      </head>
      <body bgColor="blue">
        <font color="white">
        <h3>
1       <script type="text/javascript" >
2         document.write("Your browser supports JavaScript!");
        </script>
3       <noscript>
          Please turn JavaScript on if you want to see this page!<br>
          <em>
4           Firefox &gt; Tools &gt; Options &gt; Content &gt;
              Enable JavaScript<br />
            IE &gt; Tools &gt; Internet Options &gt; Security &gt;
              Custom Level &gt;Security Setting &gt; Scripting &gt;
              Enable<br />
          </em>
5       </noscript>
```

Continues

EXAMPLE 2.6 (CONTINUED)

```
            </h3>
            </font>
        </body>
    </html>
```

EXPLANATION

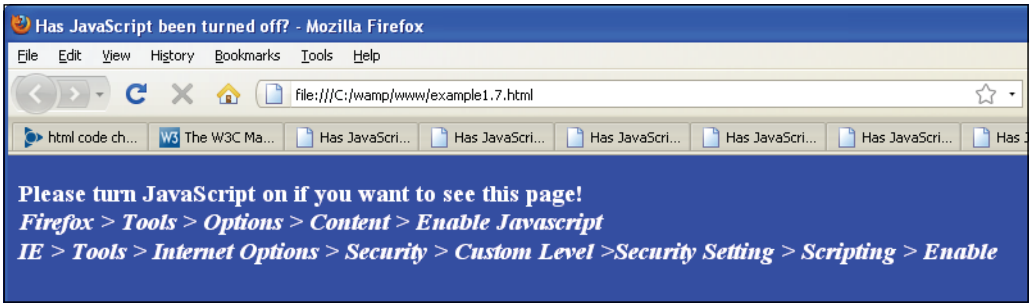
1 The JavaScript program starts here with the opening <*script*> tag.

2 This line is displayed on the Web page only if JavaScript is enabled.

3 The <*noscript*> tag is read by browsers that support JavaScript. They will ignore everything between the <*noscript*> and </*noscript*> tags. Disabled browsers will not recognize the <*noscript*> tag and thus ignore them, displaying all enclosed text.

4 JavaScript-disabled browsers will display the message shown in Figure 2.14.

5 The </*noscript*> tag ends here.


Has JavaScript been turned off? - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/wamp/www/example1.7.html
html code ch...
The W3C Ma...
Has JavaScri...
Has JavaScri...
Has JavaScri...
Has JavaScri...
Has JavaScri...
Has

Please turn JavaScript on if you want to see this page!
Firefox > Tools > Options > Content > Enable Javascript
IE > Tools > Internet Options > Security > Custom Level >Security Setting > Scripting > Enable

Figure 2.14 Output from Example 2.6.

2.7 What You Should Know

This chapter introduced you the JavaScript, the language. You should now be able to create a simple script and execute it in the browser window. Here are some things you should know:

1. How HTML and JavaScript coexist.
2. Understand the syntax, or how to write correct JavaScript statements.
3. How to execute a JavaScript program.
4. About case sensitivity, special words called reserved words or keywords, and how JavaScript handles whitespace, that it is free form.
5. Three ways to comment text that is used to explain what is going on in your program and ignored by the interpreter.
6. How to use <*script*> </*script*> tags, its attributes, and where to put them.

7. How and why you would create a .js file.
8. How to create and quote a string, and how to use the *writeln()* method.
9. Identify three types of errors and use your browser's debugging tools.
10. How to hide JavaScript from old browsers.

Exercises

1. What is a reserved word? Give an example.
2. Is JavaScript case sensitive?
3. What is the purpose of enclosing statements within curly braces?
4. What is the latest version of JavaScript? Where can you find this information?
5. What is the difference between the JavaScript *src* and *type* attributes?
6. How would you concatenate the following three strings with JavaScript?

   ```
   "trans"   "por"   "tation"
   ```

7. Write a script that demonstrates how concatenation works.
8. Create a JavaScript program that will print “Hello, world! Isn’t life great?” in an Arial bold font, size 14, and make the background color of the screen light green.
9. Add two strings to the first JavaScript program—your first name and last name—concatenated and printed in a blue sans serif font, size 12.
10. In the Location field of your browser, test the value of an expression using the *JavaScript:* protocol.
11. Find the errors in the following script:

    ```
    <html>
    <head>
       <title>Finding Errors</title>
    </head>
    <body bgcolor="yellow" text="blue">
       <script type="text/javascript"
          document.writeln("Two, ")
          document.writeln ("Three, ")
          document.write('Blast off....<br />");
       </script>
    </body>
    </html>
    ```

chapter

3 The Building Blocks: Data Types, Literals, and Variables

3.1 Data Types

A program can do many things, including calculations, sorting names, preparing phone lists, displaying images, validating forms, ad infinitum. But to do anything, the program works with the data that is given to it. Data types specify what kind of data, such as numbers and characters, can be stored and manipulated within a program. JavaScript supports a number of fundamental data types. These types can be broken down into two categories, primitive data types and composite data types.

3.1.1 Primitive Data Types

Primitive data types are the simplest building blocks of a program. They are types that can be assigned a single literal value such as the number *5.7*, or a string of characters such as *"hello"*. JavaScript supports three core or basic data types:

- numeric
- string
- Boolean

In addition to the three core data types, there are two other special types that consist of a single value:

- null
- undefined

Numeric Literals. JavaScript supports both integers and floating-point numbers. Integers are whole numbers and do not contain a decimal point, such as 123 and –6. Integers can be expressed in decimal (base 10), octal (base 8), and hexadecimal (base 16), and are either positive or negative values. See Example 3.1.

Floating-point numbers are fractional numbers such as 123.56 or –2.5. They must contain a decimal point or an exponent specifier, such as 1.3e–2. The letter "*e*" for exponent notation can be either uppercase or lowercase.

JavaScript numbers can be very large (e.g., 10^{308} or 10^{-308}).

EXAMPLE 3.1

`12345`	integer
`23.45`	float
`.234E-2`	scientific notation
`.234e+3`	scientific notation
`0x456fff`	hexadecimal
`0x456FFF`	hexadecimal
`0777`	octal

String Literals and Quoting. String literals are rows of characters enclosed in either double or single quotes.[1] The quotes must be matched. If the string starts with a single quote, it must end with a matching single quote, and likewise if it starts with a double quote, it must end with a double quote. Single quotes can hide double quotes, and double quotes can hide single quotes:

```
"This is a string"
'This is another string'
"This is also 'a string' "
'This is "a string"'
```

An empty set of quotes is called the null string. If a number is enclosed in quotes, it is considered a string; for example, *"5"* is a string, whereas *5* is a number.

Strings are called constants or literals. The string value *"hello"* is called a string constant or literal. To change a string requires replacing it with another string.

Strings can contain escape sequences (a single character preceded with a backslash), as shown in Table 3.1. Escape sequences are a mechanism for quoting a single character.

Table 3.1 Escape Sequences

Escape Sequence	***What It Represents***
\'	Single quotation mark
\"	Double quotation mark
\t	Tab

1. Any string without quotation marks surrounding it is considered the name of a variable.

Table 3.1 Escape Sequences (continued)

Escape Sequence	***What It Represents***
\n	Newline
\r	Return
\f	Form feed
\b	Backspace
\e	Escape
\\\\	Backslash
Special Escape Sequences	
\XXX	The character with the Latin-1 encoding specified by up to three octal digits XXX between 0 and 377. \251 is the octal sequence for the copyright symbol.
\xXX	The character with the Latin-1 encoding specified by the two hexadecimal digits XX between 00 and FF. \xA9 is the hexadecimal sequence for the copyright symbol.
\uXXXX	The Unicode character specified by the four hexadecimal digits XXXX. \u00A9 is the Unicode sequence for the copyright symbol.

EXAMPLE 3.2

```
     <html>
        <head><title>Escape Sequences</title></head>
        <body>
1          <pre>
           <big>
2             <script type="text/javascript">
                 <!-- Hide script from old browsers.
3                document.write("\t\tHello\nworld!\n");
4                document.writeln("\"Nice day, Mate.\"\n");
5                document.writeln('Smiley face:<font size="+3"> \u263a\n');
                 //End hiding here. -->
              </script>
           </big>
           </pre>
        </body>
     </html>
```

EXPLANATION

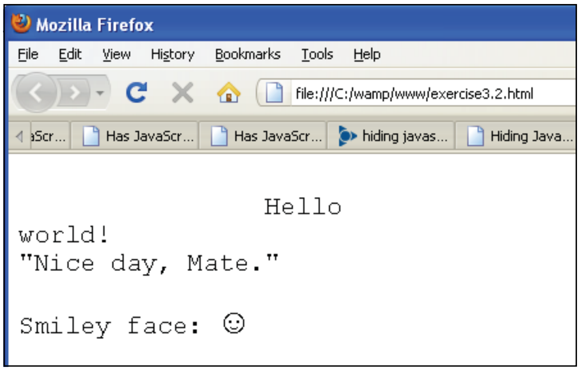
1 The escape sequences will work only if in a *<pre>* tag or an alert dialog box.

2 The JavaScript program starts here.

3 The *write()* method sends to the browser a string containing two tabs (\t\t), *Hello*, a newline (\n), *world!*, and another newline (\n).

4 The *writeln()* method sends to the browser a string containing a double quote (\”), *Nice day, Mate.*, another double quote (\”), and a newline (\n). Because the *writeln()* method automatically creates a newline, the output will display two newlines: the default value and the \n in the string.

5 This string contains a backslash sequence that will be translated into Unicode. The Unicode hexadecimal character 263a is preceded by a \u. The output is a smiley face. See Figure 3.1.


Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/wamp/www/exercise3.2.html
Has JavaScr... Has JavaScr... hiding javas... Hiding Java...

```
                        Hello
world!
"Nice day, Mate."

Smiley face: ☺
```

Figure 3.1 Escape sequences.

Putting Strings Together. The process of joining strings together is called concatenation. The string concatenation operator is a plus sign (+). Its operands are two strings. If one string is a number and the other is a string, JavaScript will still concatenate them as strings. If both operands are numbers, the + will be the addition operator. The following examples output *“popcorn”* and *“Route 66”*, respectively.

```
document.write("pop" + "corn");
document.write("Route " + 66);
```

The expression *5* + *100* results in *105*, whereas “*5*” + *100* results in “*5100*”.

Boolean Literals. Boolean literals are logical values that have only one of two values, *true* or *false*. You can think of the values as yes or no, on or off, or 1 or 0. They are used to test whether a condition is true or false. When using numeric comparison and equality operators, the value *true* evaluates to 1 and *false* evaluates to 0. (Read about comparison operators in Chapter 5, “Operators.”)

```
answer1 = true;
```

or

```
if (answer2 == false) { do something; }
```

The *typeof* Operator. The *typeof* operator returns a string to identify the type of its operand (i.e., a variable, string, keyword, or object). The values returned can be "number", "string", "boolean", "object", "null", and "undefined". You can use the *typeof* operator to check whether a variable has been defined because if there is no value associated with the variable, the *typeof* operator returns *undefined*.

FORMAT

```
typeof operand
typeof (operand)
```

EXAMPLE

```
typeof(54.6)
typeof("yes")
```

EXAMPLE 3.3

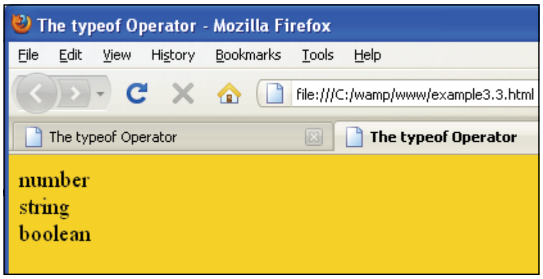
```
<html>
  <head>
     <title>The typeof Operator</title>
  </head>
  <body bgcolor="gold">
     <big>
     <script type="text/javascript">
1      document.write(typeof(55),"<br />");   // Number
2      document.write(typeof("hello there"),"<br />");   // String
3      document.write(typeof(true),"<br />");  // Boolean
     </script>
     </big>
  </body>
</html>
```

EXPLANATION

1 The integer, *55*, is a *number* type.

2 The text *"hello there"* is a *string* type.

3 The *true* or *false* keyword represent a *boolean* type. See Figure 3.2.


The typeof Operator - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/wamp/www/example3.3.html
The typeof Operator
The typeof Operator

```
number
string
boolean
```

Figure 3.2 Output from Example 3.3.

***Null* and *Undefined*.** The difference between null and undefined is a little subtle. The *null* keyword represents "no value," meaning "nothing," not even an empty string or zero. It is a type of JavaScript object (see Chapter 8, "Objects"). It can be used to initialize a variable so that it does not produce errors or to clear the value of a variable, so that there is no longer any data associated with that variable, and the memory used by it is freed. When a variable is assigned *null*, it does not contain any valid data type.

A variable that has been declared, but given no initial value, contains the value *undefined* and will produce a runtime error if you try to use it. (If you declare the variable and assign *null* to it, *null* will act as a placeholder and you will not get an error.) The word *undefined* is not a keyword in JavaScript. If compared with the == equality operators, *null* and *undefined* are equal, but if compared with the identity operator, they are not identical (see Chapter 5, "Operators").

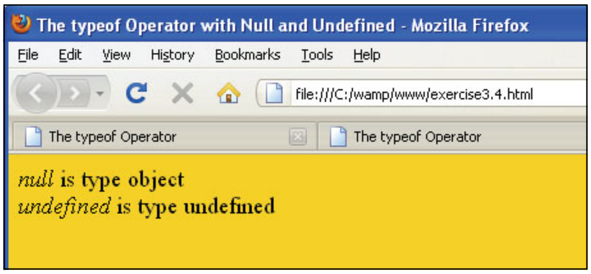
EXAMPLE 3.4

```
    <html>
      <head>
        <title>The typeof Operator with Null and Undefined</title>
      </head>
      <body bgColor="gold">
        <big>
          <script type="text/javascript">
              document.write("<em>null</em> is type "+
1                                 typeof(null),"<br />");
              document.write("<em>undefined</em> is type "+
2                                 typeof(undefined),"<br />");
          </script>
        </big>
      </body>
    </html>
```

EXPLANATION

1 The *null* keyword is a type of object. It is a built-in JavaScript object that contains no value.

2 Undefined is returned when a variable has been given no initial value or when the *void* operator is used (see Table 5.19 on page 120). See output in Figure 3.3.


The typeof Operator with Null and Undefined - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/wamp/www/exercise3.4.html
The typeof Operator
The typeof Operator

null is type object
undefined is type undefined

Figure 3.3 Output from Example 3.4.

3.1.2 Composite Data Types

We mentioned that there are two types of data: primitive and composite. This chapter focuses on the primitive types: numbers, strings, and Booleans—each storing a single value. Composite data types, also called complex types, consist of more than one component. Objects, arrays, and functions, covered later in this book, all contain a collection of components. Objects contain properties and methods, arrays contain a sequential list of elements, and functions contain a collection of statements. The composite types are discussed in later chapters.

3.2 Variables

Variables are fundamental to all programming languages. They are data items that represent a memory storage location in the computer. Variables are containers that hold data such as numbers and strings. Variables have a **name**, a **type**, and a **value**.

```
num = 5;            // name is "num", value is 5, type is numeric
friend = "Peter";   // name is "friend", value is "Peter",
                    // type is string
```

The values assigned to variables can change throughout the run of a program whereas constants, also called literals, remain fixed. JavaScript variables can be assigned three types of data:

- numeric
- string
- Boolean

Computer programming languages like C++ and Java require that you specify the type of data you are going to store in a variable when you declare it. For example, if you are going to assign an integer to a variable, you would have to say something like:

```
int n = 5;
```

And if you were assigning a floating-point number:

```
float x = 44.5;
```

Languages that require that you specify a data type are called "strongly typed" languages. JavaScript, conversely, is a dynamically or loosely typed language, meaning that you do not have to specify the data type of a variable. In fact, doing so will produce an error. With JavaScript, you would simply say:

```
n = 5;
x = 44.5;
```

and JavaScript will figure out what type of data is being stored in *n* and *x*.

3.2.1 Valid Names

Variable names consist of any number of letters (an underscore counts as a letter) and digits. The first character must be a letter or an underscore. Because JavaScript keywords do **not** contain underscores, using an underscore in a variable name can ensure that you are not inadvertently using a reserved keyword. Variable names are case sensitive; for example, *Name, name*, and *NAme* are all different variable names. Refer to Table 3.2.

Table 3.2 Valid and Invalid Variable Names

Valid Variable Names	***Invalid Variable Names***
name1	*10names*
price_tag	*box.front*
_abc	*name#last*
Abc_22	*A-23*
A23	*5*

3.2.2 Declaring and Initializing Variables

Variables must be declared before they can be used. To make sure that variables are declared first, you can declare them in the head of the HTML document. There are two ways to declare a variable: with or without the keyword *var*. Although laziness might get the best of you, it is a better practice to always use the *var* keyword.

You can assign a value to the variable (or initialize a variable) when you declare it, but it is not mandatory, unless you omit the *var* keyword. If a variable is declared but not initialized, it is "undefined."

FORMAT

```
var variable_name = value;   // initialized
var variable_name;           // uninitialized
variable_name;               // wrong
```

To declare a variable called *firstname*, you could say

```
var first_name="Ellie"
```

or

```
first_name ="Ellie";
```

or

```
var first_name;
```

You can declare multiple variables on the same line by separating each declaration with a comma. For example, you could say

```
var first_name, middle_name, last_name;
```

EXAMPLE 3.5

```
  <html>
    <head><title>Using the var Keyword</title>
      <script type="text/javascript">
1       var language="English";    // Variable is initialized
2       var name;             // OK, undefined variable
3       age;                  //  Not OK! var keyword missing ERROR!
      </script>
    </head>
    <body bgcolor="silver">
      <big>
        <script type="text/javascript">
          document.write("Language is " + language + "<br />");
          document.write("Name is "+ name + "<br />");
4         document.write("Age is "+ age + "<br />");
        </script>
      </big>
    </body>
  </html>
```

EXPLANATION

1 The variable called *language* is defined and initialized. The *var* keyword is not required here, but is recommended.

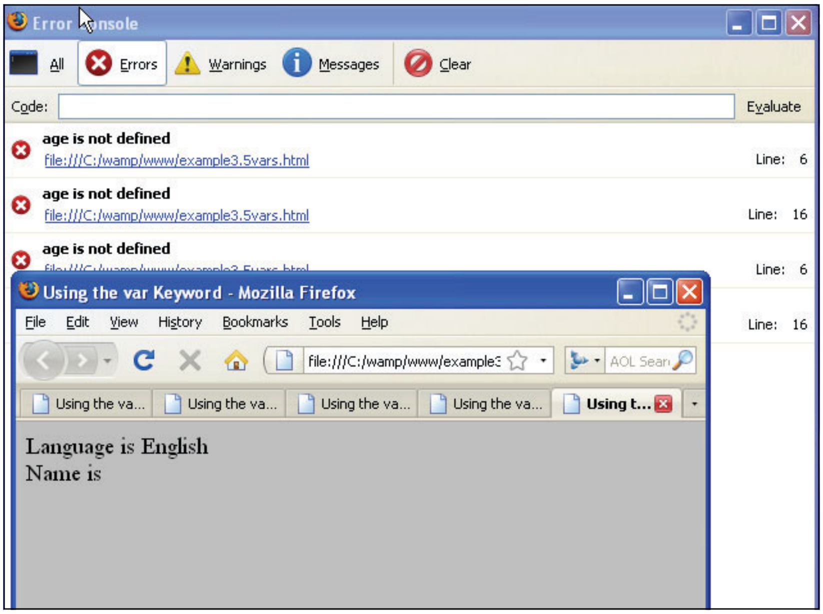
2 Because the variable called *name* is not initialized, the *var* keyword is required here.

Continues

EXPLANATION (CONTINUED)

3 The variable called *age* is not assigned an initial value. The *var* keyword is required. Without it, the program produces errors, shown in the output for Firefox and Explorer, in Figure 3.4 and Figure 3.5 (on page 63), respectively.

4 This line will not be printed until the variable called *age* is defined properly. Just use the *var* keyword as good practice, even if it isn't always required!


Error Console
All
Errors
Warnings
Messages
Clear
Code:
Evaluate
Using the var Keyword - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/wamp/www/example3
AOL Search
Using the va...
Using t...

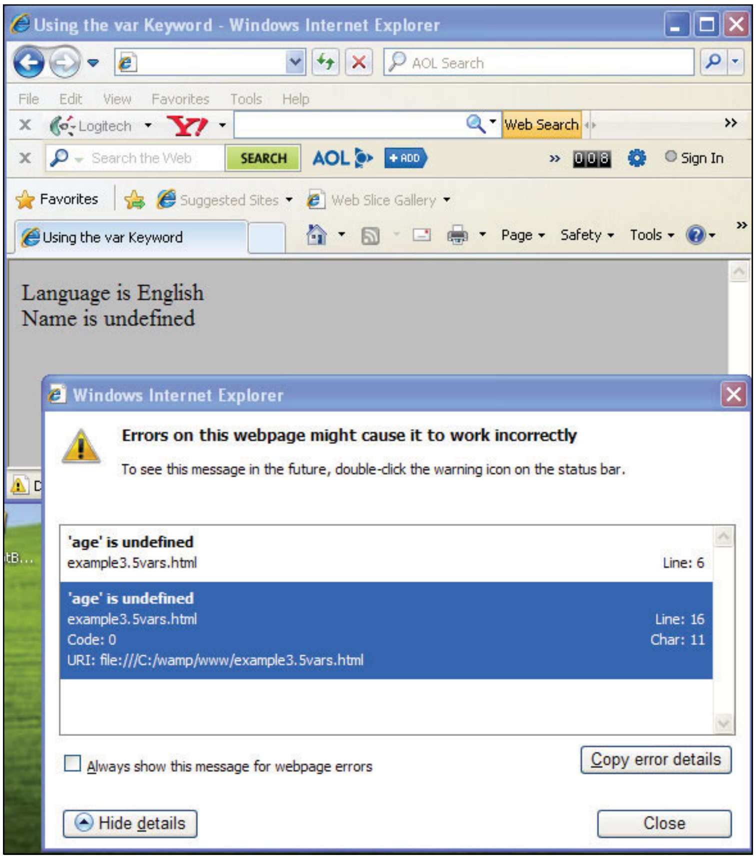
```
age is not defined
file:///C:/wamp/www/example3.5vars.html          Line: 6
age is not defined
file:///C:/wamp/www/example3.5vars.html          Line: 16
age is not defined
                                                 Line: 6
                                                 Line: 16
```

```
Language is English
Name is
```

Figure 3.4 Firefox error (JavaScript Error Console). The variable *age* was referenced twice on lines 6 and 16 in the actual program (lines numbered 3 and 4 in Example 3.5). Program was tested twice.

3.2.3 Dynamically or Loosely Typed Language

Remember, strongly typed languages like C++ and Java require that you specify the type of data you are going to store in a variable when you declare it, but JavaScript is loosely typed. It doesn't expect or allow you to specify the data type when declaring a variable. You can assign a string to a variable and later assign a numeric value. JavaScript doesn't care and at runtime, the JavaScript interpreter will convert the data to the correct type. Consider the following variable, initialized to the floating-point value of 5.5. In each successive statement, JavaScript will convert the type to the proper data type (see Table 3.3).


Using the var Keyword - Windows Internet Explorer
AOL Search
File Edit View Favorites Tools Help
Favorites
Suggested Sites
Web Slice Gallery
Using the var Keyword
Page
Safety
Tools

Language is English
Name is undefined

Windows Internet Explorer

Errors on this webpage might cause it to work incorrectly

To see this message in the future, double-click the warning icon on the status bar.

'age' is undefined
example3.5vars.html Line: 6

'age' is undefined
example3.5vars.html Line: 16
Code: 0 Char: 11
URI: file:///C:/wamp/www/example3.5vars.html

Always show this message for webpage errors
Copy error details
Hide details
Close

Figure 3.5 Internet Explorer error (Example 3.5).

Table 3.3 How JavaScript Converts Data Types

Variable Assignment	***Conversion***
`var item = 5.5;`	Assigned a float
`item = 44;`	Converted to integer
`item = "Today was bummer";`	Converted to string
`item = true;`	Converted to Boolean
`item = null;`	Converted to the null value

EXAMPLE 3.6

```
       <html>
         <head><title>JavaScript Variables</title>
 1         <script type="text/javascript">
 2           var first_name="Christian"; // first_name is assigned a value
 3           var last_name="Dobbins";    // last_name is assigned a value
 4           var age = 8;
 5           var ssn;     // Unassigned variable
 6           var job_title=null;
           </script>
 7       </head>
 8       <body bgcolor="lightgreen">
           <big>
 9           <script type="text/javascript">
10             document.write("<b>Name:</b> " + first_name + " "
                                 + last_name + "<br />");
11             document.write("<b>Age:</b>  " + age + "<br />");
12             document.write("<b>SSN:</b>  " + ssn + "<br />");
13             document.write("<b>Job Title:</b> " + job_title+"<br />");
14             ssn="xxx-xx-xxxx";
15             document.write("<b>Now SSN is:</b> " + ssn , "<br />");
             </script>
             <p>
               <img src="Christian.gif" /></body>
             </p>
           </big>
         </body>
       </html>

       Output:

       10   Name: Christian Dobbins
       11   Age: 8
       12   SSN: undefined
       13   Job Title: null
       15   Now Ssn is: xxx-xx-xxx
```

EXPLANATION

1 This JavaScript program is placed within the document head. Because the head of the document is processed before the body, this assures you that the variable definitions will be defined first.

2 The string *"Christian"* is assigned to the variable called *first_name*.

3 The string *"Dobbins"* is assigned to the variable called *last_name*.

4 The number *8* is assigned to the variable called *age*.

5 The variable called *ssn* is not assigned any value at all. It is an uninitialized variable. The return value is *undefined*.

EXPLANATION

6 The value *null* is assigned to the variable called *job_title*. *Null* is used to set a variable to an initial value different from other valid types, but if used in an expression the value of *null* will be converted to the appropriate type.

7 The document head ends here.

8 The body of the document starts here.

9 A new JavaScript program starts here. All the variables declared in the head of the document are available here. Variables that are available throughout the entire document are called global variables.

10 The *document.write()* method concatenates the values of the strings with the + sign and sends them to the browser to display on the screen.

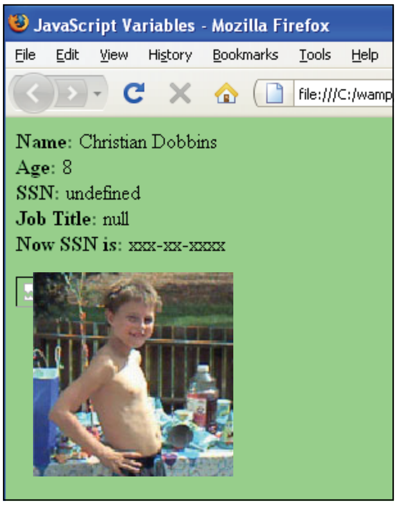
11 The value of the variable called *age* is displayed.

12 The variable called *ssn* was declared, but not initialized. It has no value, which JavaScript calls *undefined*.

13 The variable *job_title* was assigned *null*, a placeholder value. The *null* string is returned.

14 The variable *ssn* is assigned a string value. It is no longer *undefined*. Even though the variable was declared in the head of the document, as long as it was declared, it can be assigned a value anywhere else in the document.

15 The value of the variable *ssn* is displayed. Figure 3.6 shows the output in Internet Explorer.


JavaScript Variables - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/wamp

```
Name: Christian Dobbins
Age: 8
SSN: undefined
Job Title: null
Now SSN is: xxx-xx-xxxx
```

Figure 3.6 Declaring and displaying variables.

3.2.4 Scope of Variables

Scope describes where a variable is visible, or where it can be used, within the program. JavaScript variables are either of global or local scope. A global variable can be accessed from any JavaScript script on a page, as shown in Example 3.6. The variables we have created so far are global in scope.

It is often desirable to create variables that are private to a certain section of the program, thus avoiding naming conflicts and accidentally changing a value in some other part of the program. Private variables are called local variables. Local variables are created when a variable is declared within a function. Local variables must be declared with the keyword, *var*. They are accessible only from within the function from the time of declaration to the end of the enclosing block, and they take precedence over any global variable with the same name. (See Chapter 7, "Functions.")

3.2.5 Concatenation and Variables

To concatenate variables and strings together on the same line, the + sign is used. The + sign is an operator because it operates on the expression on either side of it (each called an operand). Sometimes the + sign is a string operator and sometimes it is a numeric operator when used for addition. Addition is performed when both of the operands are numbers. In expressions involving numeric and string values with the + operator, JavaScript converts numeric values to strings. For example, consider these statements:

```
var temp = "The temperature is " + 87;
 // returns "The temperature is 87"
var message = 25 + " days till Christmas";
// returns "25 days till Christmas"
```

But, if both operands are numbers, then addition is performed:

```
var sum = 10 + 5; // sum is 15
```

EXAMPLE 3.7

```
<html>
  <head><title>Concatenation</title></head>
  <body>
    <script type="text/javascript">
      var x = 25;
      var y = 5 + "10 years";
      document.write( x + " cats" , "<br />");
      document.write( "almost " + 25 , "<br />");
      document.write( x + 4, "<br />");
      document.write( y, "<br />");
      document.write(x  +  5 + " dogs" , "<br />");
      document.write(" dogs"  + x + 5 , "<br />");
    </script>
```

EXAMPLE 3.7 (CONTINUED)

```
      </body>
    </html>

Output:

3  25 cats
4  almost 25
5  29
6  510 years
7  30 dogs
8  dogs255
```

EXPLANATION

1 Variable *x* is assigned a number.

2 Variable *y* is assigned the string *510 years*. If the + operator is used, it could mean the concatenation of two strings or addition of two numbers. JavaScript looks at both of the operands. If one is a string and one is a number, **the number is converted to a string** and the two strings are joined together as one string, so in this example, the resulting string is *510 years*. If one operand were *5* and the other *10*, addition would be performed, resulting in *15*.

3 A number is concatenated with a string. The number *25* is converted to a string and concatenated to "*cats*", resulting in *25 cats*. (Note that the *write()* method can also use commas to separate its arguments. In these examples the **<br>** tag is not concatenated to the string. It is sent to the *write()* method and appended.)

4 This time, a string is concatenated with a number, resulting in the string *almost 25*.

5 When the operands on either side of the + sign are numbers, addition is performed.

6 The value of *y*, a string, is displayed.

7 The + operators works from left to right. Because *x* and *y* are both numbers, addition is performed, *25* + *5*. *30* is concatenated with the string "*dogs*".

8 Because the + works from left to right, this time the first operand is a string being concatenated to a number, the number is converted to string *dogs25* and concatenated with string *5*.

3.3 Constants

The weather and moods are variable; time is constant, and so are the speed of light, midnight, *PI*, and *e*. In programming, a constant is a special kind of placeholder with a value that cannot be changed during program execution. Many programming languages use a special syntax to define a constant to distinguish it from a variable. JavaScript declares constants with the *const* type (which replaces *var*) and the name of the constant is in uppercase letters, by convention only. Constants are assigned values at the time of the declaration, and it is impossible to modify them during the run of the program. Caveat: Firefox, Opera, Safari, Netscape, and many browsers support the JavaScript reserved keyword *const* to declare constants (see Figure 3.7). It is important to note that Internet Explorer 7 and 8 do not support it, as shown in Figure 3.8.

EXAMPLE 3.8

```
      <html>
        <head><title>Using the const Keyword</title>
          <script type="text/javascript">
1           const NOON = 12;
2           const FREEZING = 32;    // Can't change
          </script>
        </head>
        <body bgcolor="silver">
          <big>
            <script type="text/javascript">
              document.write("Farenheit temp is " + FREEZING + ".<br />");
3             FREEZING = 32 + 10;
4             NOON = NOON + " noon";
5             document.write("Make it warmer " + FREEZING + ".<br />");
              document.write("See you at ", NOON, ".<br />");
            </script>
          </big>
        </body>
      </html>
```

EXPLANATION

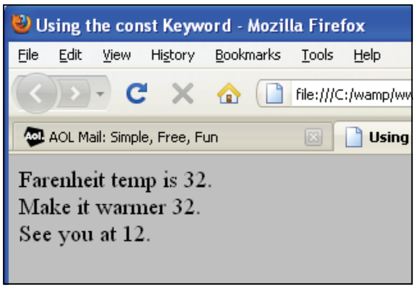
1 The constant NOON is assigned 12, a value that will not change throughout the execution of this program.

2 The constant FREEZING is assigned 32, a value that will not change throughout the execution of this program.

3 Now if we try to add 10 to the constant, the value of the constant doesn't change. It's still 32.

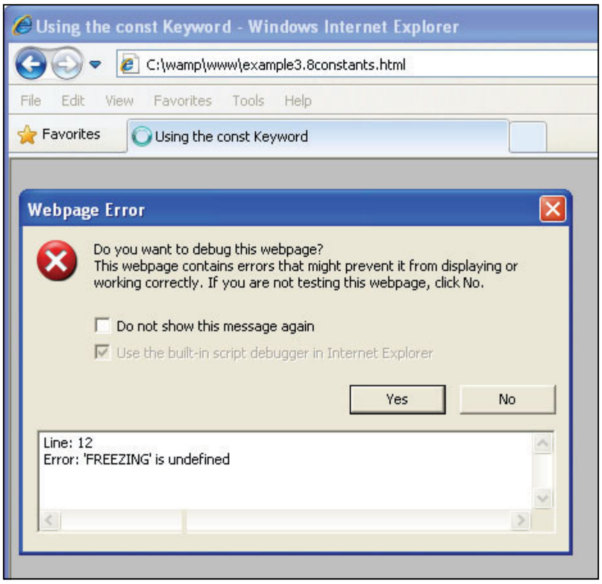
4 This time, we try to concatenate a string to the constant NOON. It will not be changed.

5 The constants FREEZING and NOON are displayed. They were not changed.


Using the const Keyword - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/wamp/ww
AOL Mail: Simple, Free, Fun
Using

```
Farenheit temp is 32.
Make it warmer 32.
See you at 12.
```

Figure 3.7 Firefox 3.5.7 supports the *const* keyword.


Using the const Keyword - Windows Internet Explorer
C:\wamp\www\example3.8constants.html
File Edit View Favorites Tools Help
Favorites
Using the const Keyword
Webpage Error
Do not show this message again
Use the built-in script debugger in Internet Explorer
Yes
No

Do you want to debug this webpage?
This webpage contains errors that might prevent it from displaying or working correctly. If you are not testing this webpage, click No.

```
Line: 12
Error: 'FREEZING' is undefined
```

Figure 3.8 Internet Explorer 8 does not support the *const* keyword.

3.4 Bugs to Watch For

Try to declare all your variables at the beginning of the program, even if you don't have values for them yet. This will help you find misspelled names faster. Watch that you use proper variable names. Don't used reserved words and words that are too long to remember or type easily. Remember that variable names are case sensitive. *MyName* is not the same as *myName*. Avoid giving two variables similar names, such as *MyName* and *myNames*. Avoid one-character differences in variable names, such as *Name1* and *Names1*. Even though you aren't always required to use the *var* keyword, do it anyway. It's safer. And, of course, be sure that the variables you use are spelled properly throughout the script.

When you use strings don't forget to enclose the strings in either double or single quotes. Quoting will get the best of programmers every time!

3.5 What You Should Know

This chapter introduced you to the fundamental building blocks of the JavaScript language; that is, the kinds of data that can be stored and manipulated within a program, such as strings and numbers. To proceed to the next chapters, you should know:

1. What is meant by primitive data types.
2. What numeric literals are.
3. How to concatenate strings.
4. The two values for a Boolean.
5. What the *typeof* operator returns.
6. What is meant by *null*, *undefined*.
7. The difference between a variable and a constant.
8. The difference between loosely typed and strongly typed languages.
9. What scope is.
10. What types can be assigned to a variable.
11. How to name a variable.
12. When *var* is used.

Exercises

1. Create a script that uses the three primitive data types and prints output for each type. In the same script, print the following:

   ```
   She cried, "Aren't you going to help me?"
   ```

2. Go to *http://www.unicode.org/charts/PDF/U2600.pdf* and find a symbol. Use JavaScript to display one of the symbols in a larger font (+5).

3. Write a script that displays the number 234 as an integer, a floating-point number, an octal number, a hexadecimal number, and the number in scientific notation.

4. When is it necessary to use the *var* keyword?

5. Write a script that contains four variables in the head of the document: The first one contains your name, the second contains the value 0, the third one is declared but has no value, and the last contains an empty string. In the body of the document, write another script to display the type of each (use the *typeof* operator).

This page intentionally left blank

chapter

4 Dialog Boxes

4.1 Interacting with the User

Programs like to talk, ask questions, get answers, and respond. In the previous chapter, we saw how the *write()* and *writeln()* methods are used to send the document's output to the browser. The document is defined in an object and *write()* and *writeln()* are methods that manipulate the document, make it do something. The document object is defined within a window. The window is also an object and has its own methods.

The window object uses dialog boxes to interact with the user. The dialog boxes are created with three methods:

- *alert()*
- *prompt()*
- *confirm()*

4.1.1 The *alert()* Method

The window's *alert()* method is used to send a warning to the user or alert him or her to do something. For example, you might let the user know he or she has not entered his or her e-mail address correctly when filling out a form, or that his or her browser doesn't support a certain plug-in, and so on. The alert box is also commonly used for debugging to find out the results of a calculation, if the program is executing in an expected order, and so on.

The *alert()* method creates a little independent window—called a dialog box—that contains a a user-customized message placed after a small triangle, and beneath it, an OK button. See Figure 4.1. When the dialog box pops up, all execution is stopped until the user clicks the OK button in the pop-up box. The exact appearance of this dialog box might differ slightly on different browsers, but its functionality is the same.

Unlike the *write()* method, the *alert()* method doesn't require the window object name in front of it as *window.alert()*. Because the window is the top-level browser object, it doesn't have to be specified. This is true with any window object methods you use.