

Chapter 17: Introduction to JavaScript

Learning Objectives

- Understand what JavaScript is
- See the methods for debugging code errors
- Learn the methods of putting JavaScript into an HTML document
- Learn two basic JavaScript functions

17.1 What is JavaScript

Formally, JavaScript is an interpreted, object-oriented programming language. It is often confused with the programming language Java, but they are completely independent languages¹. JavaScript has a full set of capabilities that make it just as powerful as many other popular programming languages. To fully cover all the features of JavaScript would require about 1,000 pages of detailed reference text. We will take a more limited approach to the language, covering basic functionality for use in web pages.

For our purposes, we will be using JavaScript as a scripting language that lets us create dynamic HTML. Instead of static images, JavaScript can be used to make them change based on the users' actions. Instead of programming HTML that will be the same for every user, JavaScript can be used to make the text and appearance different for each person. JavaScript can be used to add interactive features to pages, and add finer control over events. Using JavaScript cookies, data can be stored and passed between pages for personalization or to facilitate more advanced functionality. This is all called Client Side JavaScript, because the code goes directly in the HTML page.

JavaScript is implemented in Netscape and Mozilla browsers, Safari, the Mac OS X browser, and some other less popular browsers. JavaScript is currently in version 1.5, which is supported in Netscape 6 and later. The previous version of Netscape – version 4.5 – supports JavaScript version 1.3. The language features introduced here are very basic and will generally work in both old and new browsers. When writing JavaScript code, you should consider what browser versions your target audience will be using and use that to make decisions about which features to use.

Microsoft has chosen to implement their own, slightly different version of the language that they call JScript. Though mostly the same as JavaScript, there are some significant differences. Unlike HTML, which works pretty much the same way in every browser, it

¹ JavaScript and Java actually can interact quite well on web pages. Since this text does not cover Java, readers should check out some of the many online resources available to explain these methods of interaction.

is more difficult to get JavaScript working in both Internet Explorer and Mozilla/Netscape. In these chapters, the term "JavaScript" will be used as a general term to refer to the core set of code that works in both browser versions. While we will only look at features that work in both IE and other browsers, it is especially important to check your JavaScript in all of the major browsers because the chance of errors is much higher.

The good news is that JavaScript is moving toward becoming a standard. The European standards group ECMA as approved ECMA-262, which is a standard language derived from Netscape JavaScript. That standard has also been approved by the International Standards Organization as ISO-16262. JavaScript has been fully compliant with the ECMA standard since version 1.3. More information about this standardization effort can be found online at the websites for Netscape, ECMA, and ISO.

17.2 Getting Started with JavaScript

There is a lot to learn before JavaScript can become an effective and powerful part of your web pages. The next chapter presents all of the syntax basics, but the truth is that learning JavaScript is a process that will produce a lot of errors along the way. Unlike HTML which, at worst, looks a bit strange when there are big coding errors, JavaScript is not forgiving. The smallest error will prevent the code from working at all. Learning to identify these problems, or *bugs*, in your code will be the most valuable skill you learn. This section will introduce several features of browsers and the JavaScript language that will be immeasurably useful as you progress.

17.2.1 The JavaScript Console

When an error is found in the code, there must be some way to find it. Thankfully, the browsers are designed to recognize an error in the code and point it out. In Mozilla and Netscape, this is done with the JavaScript Console. The console is a window that lists any JavaScript errors. It can be brought up by going to the Tools menu, selecting Web Development, and then clicking on JavaScript Console (as shown in figure 17.1), or by typing "javascript:" in the URL bar. If using the latter option, be sure to include the colon after the word "javascript" to bring up the console.

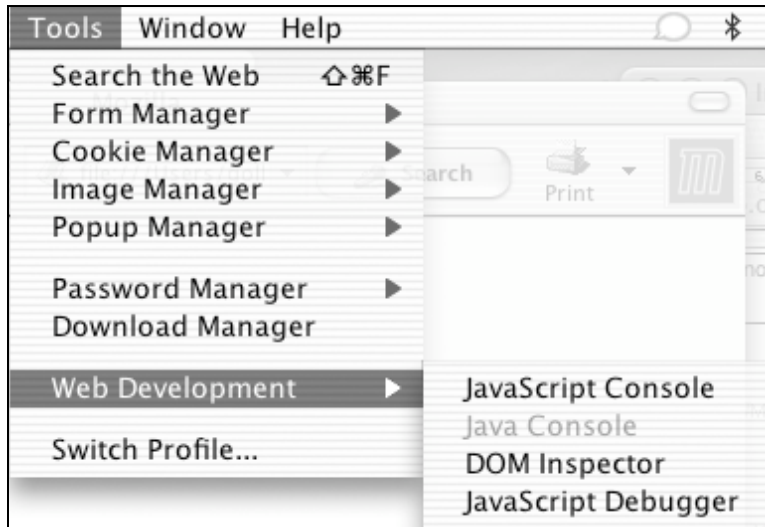


Figure 17.1: Bringing up the JavaScript Console in Mozilla.

The console will show each error with the number of the line it appears on and a description of what is the likely cause of the error. In figure 17.2, the console has one error logged. Examining the code shows that the ending quote from "this is a test" is missing. A missing quote is what causes the "unterminated string literal" error.



Figure 17.2: The JavaScript console showing an error on Line 6 of this test file.

In Internet Explorer, each error is brought up as an individual alert to the user, instead of being logged onto a console behind the scenes. Though the format is slightly different, the error still shows a description of the problem ("unterminated string constant"), the line number, and the number of the character where the error begins.

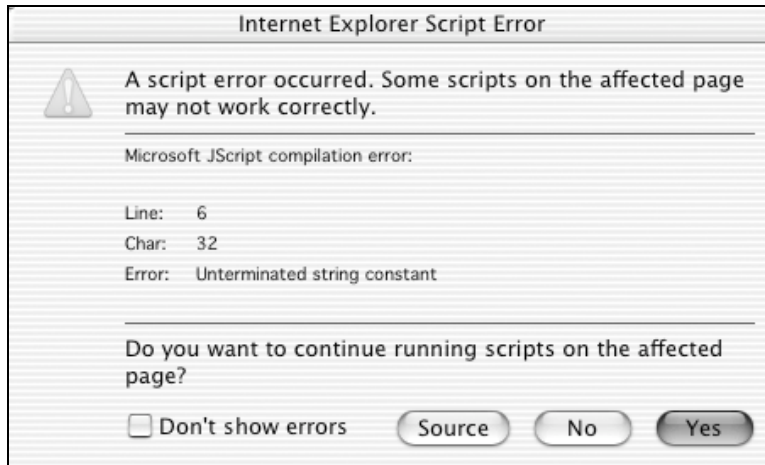


Figure 17.3: The pop-up alert that Internet Explorer shows when a JavaScript error is found.

Sometimes, depending on the error, the error messages will show the wrong line or indicate the wrong place for a problem. The important thing to remember is that the computer is doing its best to figure out what is wrong with code it cannot understand. Usually the errors are pretty close, if not right on. When the error does not look right, it is very likely that the actual error is found on the previous line. Part of learning to code and debug is to understand how to use error messages effectively – most of the time, they are not simple lists of corrections.

17.2.2 Embedding JavaScript on the Page

Unlike HTML, which is a *markup language*, JavaScript is a programming language. The syntax is entirely different from HTML. Just like CSS code (with its own, unique syntax) is inserted into web pages, JavaScript is also contained *within* HTML to add its functionality to the page. Last chapter, we saw that there are three ways Cascading Style Sheets can be embedded in an HTML document: external, document level, and inline. JavaScript can be embedded in the same three ways.

While CSS used the `<style>` tag, JavaScript is enclosed in the `<script>` tag. An external JavaScript file contains just JavaScript code – no HTML – and usually has the ".js" extension. The script tag should contain the "language" attribute which should be set equal to "JavaScript" so the browser knows what kind of script is being encoded. Some authors choose to include the version of JavaScript (e.g. "javascript1.3") to be more specific. Generally, browsers will have no problems understanding code without the "language" attribute, but it is good practice to include it. To import an external JavaScript, the "src" attribute is used in the `<script>` tag, and is set equal to the URL of the file to be imported:

```
<head>
  <script language="JavaScript" src="outsideFunctions.js">
</script>
</head>
```

Document level JavaScript also uses the `<script>` tag. The JavaScript code is contained between the start and end tags. It is not required that this be placed in the header section, though. The general usage of the `<script>` tag is as follows:

```
<script language="JavaScript">
    JAVASCRIPT GOES HERE
</script>
```

A `<script>` and `</script>` can go in the body of the document to list a set of actions that should be completed at that particular point in the page loading. JavaScript will execute wherever it is encountered in the document. That means a series of commands contained in the `<script>` tag in the header will execute before the body of the page loads, since they occur in the document before the `<body>` tag. If code is contained in the document, it will be executed as it is encountered. The text before the `<script>` tag will load and appear in the window, then the JavaScript will execute, and once the execution is complete, the rest of the document will load. The exception to this is when JavaScript commands are contained in functions, but that will be presented in the next chapter.

The `alert()` command is part of JavaScript, and causes an alert window to pop up in the user's browser. We will look at it more in detail in the next section, but for now, just consider it to be a line of code. A document level JavaScript could appear something like this:

```
<html>
<body>

<h1>Welcome to this page</h1>

<script language="JavaScript">
    alert("Hello");
</script>

<h1>I am learning JavaScript</h1>

</body>
</html>
```

When this page loads, the first line, "Welcome to this page," will appear, then the alert will pop up. After the user dismisses the alert by clicking the "Ok" button, the second line, "I am learning JavaScript," will appear.



Figure 17.4: The alert loading from a section of document-level JavaScript.

Finally, JavaScript commands can be inserted into individual tags in a way similar to inline style sheets. There is no parallel to the "style" attribute, though. Instead, tags have a set of *action attributes* that can be used to execute JavaScript when the user performs a certain action. There are a fixed number of action attributes, but they cannot all be used with every tag. Appendix F lists these action attributes and the tags with which each can be used. As we progress in our knowledge of JavaScript, we will use more and more of these. For now, we will just look at the "onclick" attribute. This attribute is most often seen with the capitalization "onClick". All of the action attributes begin with "on", and it has become most common for authors to capitalize the first letter after the "on". This practice, however, is not XHTML compliant. XHTML requires all characters in the attribute to be lower case.

The general syntax for using onclick is as follows:

```
<a href="http://example.com"
    onclick="SOME JAVASCRIPT">
```

This attribute is usually used with form buttons and links, and it is set equal to a set of JavaScript commands. When the user clicks the object that has this attribute, the commands are executed. For example, consider this link with an "onclick":

```
<a href="http://www.google.com"
  onclick="alert('You are now going to Google');">
  Visit Google
</a>
```

When the user clicks this link, the alert will pop up first, displaying the text. Once the user clicks "OK" on the alert, the browser will follow the link and load the Google page.

One note in this example is that the text in the alert is enclosed in single quotes (') while the whole value of the "onclick" attribute is enclosed in double quotes ("). This is because the browser needs to know which quotes go together. If double quotes were used in all of the cases, the browser would think that the set which begins the alert text would actually be closing the quotes that start the attribute. Any text after that would simply confuse the browser.

```
onclick=alert("You are now going to Google");">
```

The line above is formatted to show this effect. The browser would consider the bold section to be the value of the attribute, since it is contained in a set of double quotes. Everything after that, shown in italics, would be ignored since it does not contain valid attribute name-value pairs. Using single quotes inside the alert removes this ambiguity so the code executes correctly.

17.3 Useful Beginner Syntax

17.3.1 Built-in Functions

There are two built-in JavaScript commands that are very useful as we learn how to use JavaScript. One is the `alert()` function presented above. The other, `document.write()`, will write text directly into the document.

Although we have seen the alert function work, it is worth describing it more clearly. It is called a *function* because it does a more complicated process when it is called. We do not need to tell the browser to make a window of a certain size, add an "OK" button and an alert icon, and show a certain line of text, nor do we have to be involved in the process of shifting focus from the window to the alert, or keeping it there until the user dismisses the alert. All of these processes have to be done, but the function hides all of those details for us.

The only information we need to pass to the alert function is a *string* that contains the text that should appear on the alert. A string is just a series of characters put together. Strings are contained in quotes as is seen in the above examples. That string goes between the parentheses after the word "alert". This is called *passing* the string to the alert function.

The second useful function is `document.write()`. Like the alert function, we need to pass in a string. Instead of appearing in an alert, the string is written directly into the document. For example, the following HTML uses `document.write()`.

```
<html>  
<body>  
This line is plain text. <p>
```

```
<script>
    document.write("This line came from some JavaScript");
</script>

</body>
</html>
```

Figure 17.5 shows that the two lines of text appear as though they were generated in the same way. Though this may not show why using "document.write()" is useful over plain HTML, the more advanced features covered in the text chapter will take advantage of this. Since JavaScript allows us to store information and make calculations, this function gives us the ability to create pages that have information which is different for each user.

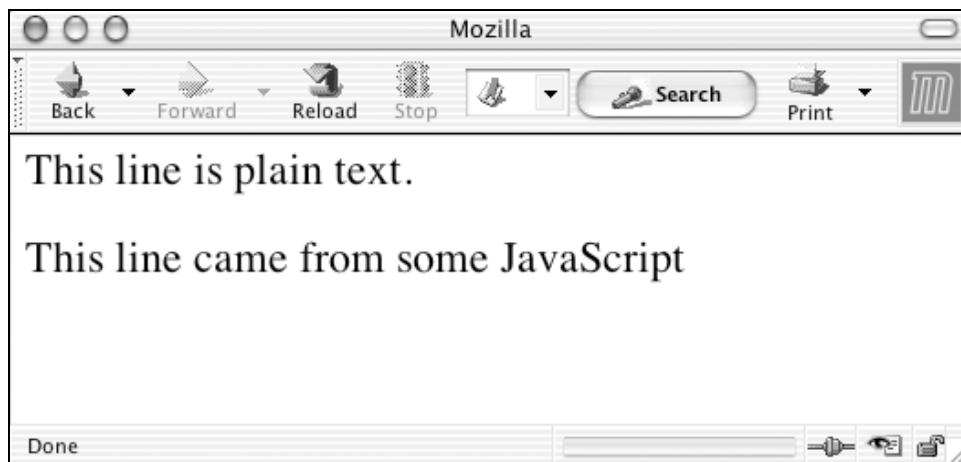


Figure 17.5: Two lines of text generated from the code example using "document.write()".

With these two functions, we have created our first pages using JavaScript. These functions will provide the foundation for our further work and serve as excellent tools for diagnosing errors.

17.3.2 Comments

Comments will be the most important feature that helps keep track of what is happening within your code. Comments are lines within the JavaScript that are not executed, but rather serve as notes about what is going on in the code. They can also be used to temporarily prevent a section of code from being executed, which is helpful in isolating errors.

A single line comment is created by putting two forward slashes before the comment. This comments out the code for the rest of the line, preventing it from being executed:

```
//This next line prints text into the document
document.write("This line came from some JavaScript");
```

```
alert("The text has been printed"); //this line shows an
                                     //alert so we know
                                     //things worked
```

Notice that comments can start at the beginning of a line or in the middle of one. Multiple lines can also be commented out. The beginning a multi-line comment is marked with a slash and an asterisk. The comment is ended with an asterisk followed by a slash. Everything between the `/*` and `*/` is commented out.

```
alert("The text has been printed"); /*this line shows an
                                     alert so we know
                                     things worked */
```

17.4 Review

- JavaScript is a programming language. The code can be placed within HTML documents and executed in the browser.
- JavaScript errors are reported in the JavaScript console in Netscape and Mozilla, or through error alerts in Internet Explorer.
- Internet Explorer implements a separate version of JavaScript, called Jscript. The difference between the languages requires extra attention to make sure code works in both IE and Netscape / Mozilla.
- JavaScript is placed within a page using the `<script>` tag for external and document level scripts, or with action attributes for inline scripts.
- Action attributes are attributes of tags that are set equal to code that should be executed when a specific action occurs.
- Two basic JavaScript functions are `alert()` and `document.write()`.
- The `alert()` function pops up a window in the browser with a line of text. The user has to click the "OK" button on the window to get rid of the alert.
- The `document.write()` function writes a line of text directly into the HTML document.
- A string is a series of characters put together and is usually written in quotes.
- Comments are lines in the JavaScript that are not executed. They can be used to make notes in the code or temporarily prevent code from being executed.
- Single line comments begin with `//`. Multi-line comments are contained between `/*` and `*/`.

17.5 Questions

1. What type of language is JavaScript?
2. How can JavaScript code from an external file be imported into an HTML document?
3. What does the `alert()` function do?
4. Give the code that will raise an alert that says "Illegal Error!"
5. What does `document.write()` do?
6. Where would `document.write()` be included in the code of an HTML page?
7. Comments

- a. What is a comment?
 - b. Can a comment be on the same line as functional code?
 - c. Can a comment come *before* functional code on the same line? If so, give an example. If not, explain why.
 - d. Can multiple lines of code be commented out? If so, give an example. If not, explain why.
 - e. Give two ways that comments are useful in code.
8. What is a string?
 9. How is a string written in JavaScript?
 10. Look at the JavaScript Console in your version of Netscape or Mozilla. How many errors are there from your general surfing habits? Have you noticed problems with any of the sites that show up as having JavaScript errors?

17.6 Exercises

1. Correct the errors with the following code (Hint: there are at least 5 errors):

```
<a href="http//google.com" onclick="alert("HEY!");>
```

2. Create a simple HTML webpage where all of the text and tags are written into the document using `document.write()` functions.
3. Add comments to the code from Exercise 3. Include both single line and multi-line comments.
4. How are external, document level, and inline JavaScript codes included in an HTML document? Give an example of each.
5. Give an example using `document.write()` that will print a level 1 heading (`<h1>`) with the text "Greetings!".