Brad Dayley

ESSENTIAL CODE AND COMMANDS

# jQuery and JavaScript

## PHRASEBOOK

# jQuery and JavaScript

## PHRASEBOOK

*This page intentionally left blank*

# jQuery and JavaScript

## PHRASEBOOK

Brad Dayley

Addison-Wesley

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

*Library of Congress Control Number: 2013950281*



ISBN-13: 978-0-321-91896-3
ISBN-10: 0-321-91896-7

First printing: December 2013

**Acquisitions Editor**
Mark Taber

**Managing Editor**
Kristy Hart

**Project Editor**
Katie Matejka

**Copy Editor**
Karen Gill

**Indexer**
Publishing Services, WordWise, Larry Sweazy

**Proofreader**
Kathy Ruiz

**Technical Reviewer**
Phil Ballard

**Editorial Assistant**
Vanessa Evans

**Cover Designer**
Chuti Prasertsith

**Senior Compositor**
Gloria Schurick

# Dedication

*For D!*
*A & F*

# Contents

# Acknowledgments

I'd like to take this page to thank all those who made this title possible. First, I thank my wonderful wife and boys for giving me the inspiration and support I need. I'd never make it far without you. Thanks to Mark Taber for getting this title rolling in the right direction; Karen Gill for turning the ramblings of my techie mind into coherent text; Phil Ballard for ensuring the accuracy in the book and keeping me honest; Kathy Ruiz and Gloria Schurick for making sure the book is the highest quality; Larry Sweazy for making sure that the readers can actually find what they look for in the book; Tammy Graham and Laura Robbins for their graphical genius; Chuti Prasertsith for the stylish and sleek cover; and Katherine Matejka for all her hard work in making sure this book is the best it can be. You guys are awesome!

## About the Author

**Brad Dayley** is a senior software engineer with 20 years of experience developing enterprise applications. He has used HTML/CSS, JavaScript, and jQuery extensively to develop a wide array of web pages ranging from enterprise application interfaces to sophisticated rich Internet applications to smart interfaces for mobile web services. He is the author of *Python Developer's Phrasebook* and *Sams Teach Yourself jQuery and JavaScript in 24 Hours*.

## We Want to Hear from You!

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

You can email or write directly to let us know what you did or didn't like about this book—as well as what we can do to make our books stronger.

*Please note that we cannot help you with technical problems related to the topic of this book, and that due to the high volume of mail we receive, we might not be able to reply to every message.*

When you write, please be sure to include this book's title and author as well as your name and contact information.

Email: feedback@developers-library.info

Mail: Reader Feedback
Addison-Wesley Developer's Library
800 East 96th Street
Indianapolis, IN 46240 USA

## Reader Services

Visit our Web site and register this book at informit.com/register for convenient access to any updates, downloads, or errata that might be available for this book.

*This page intentionally left blank*

1

# Jumping into jQuery, JavaScript, and the World of Dynamic Web Development

JavaScript and its amped-up counterpart jQuery have completely changed the game when it comes to creating rich interactive web pages and web-based applications. JavaScript has long been a critical component for creating dynamic web pages. Now, with the advancements in the jQuery, jQuery UI, and jQuery Mobile libraries, dynamic web development has changed forever.

This chapter focuses on providing you with some concepts of dynamic web programming and getting set up to use JavaScript and jQuery in your web pages.

# Understanding JavaScript

JavaScript is a programming language much like any other. What separates JavaScript from other programming languages is that the browser has a built-in interpreter that can parse and execute the language. That means you can write complex applications running in the browser that have direct access to the browser, web page elements, and the web server.

This allows JavaScript code to dynamically add, modify, or remove elements from a web page without reloading it. Access to the browser gives you access to events such as mouse movements and clicks. This is what enables JavaScript to provide functionality such as dynamic lists as well as drag and drop. Figure 1.1 shows an example of downloading a web page from a server and then using JavaScript code to dynamically populate a `<ul>` element with `<li>` children.

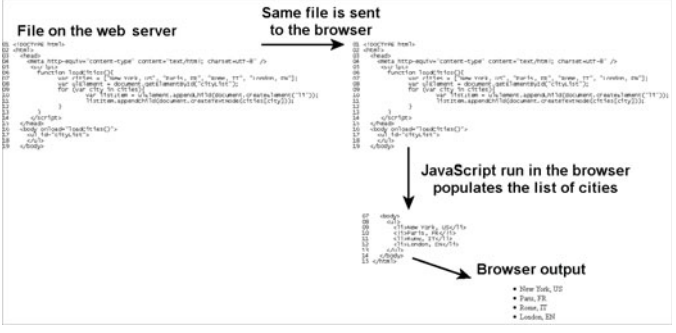


**Figure 1.1** JavaScript runs in the browser and can change the HTML elements on the web page without needing to retrieve additional pages from the web server.

# Adding JavaScript to an HTML Document

```
<head>
  <script type="text/javascript">
    alert("JavaScript is Enabled!");
  </script>
</head>
```

You can add JavaScript to an HTML document using HTML `<script>` tags. When the browser encounters a `<script>` tag, it parses the contents and then executes the JavaScript inside. Typically, the `<script>` tags are added to the `<head>` element, but you can also add them to the `<body>` element.

The web browsers currently default all scripts to `javascript`; however, it is a good idea to still set the type to `"text/javascript"` for future compatibility if that changes.

**Watch out!**

The order that `<script>` tags appear in the HTML document determines their load order. If you are loading multiple scripts, keep in mind that using the same variable and function names in subsequent scripts overwrites the ones already defined in previous ones.

## Loading JavaScript from External Files

```
<head>
  <script type="text/javascript"
    src="scripts/scriptA.js"></script>
  <script type="text/javascript"
    src="scripts/scriptB.js"></script>
</head>
```

As you create more and more complex JavaScript web applications, you will find that adding the JavaScript to your HTML files doesn't make much sense. The files become too big, and you will often want to reuse the scripts in other web pages.

Instead of including the JavaScript inside the `<script>` tag, you can specify a `src` location for the script to be loaded from. When the browser encounters a `src` attribute in the `<script>` tag, it downloads the script from the server and loads it into memory.

**Watch out!**

The browser downloads the external scripts asynchronously. That means you need to be careful if you reference one script from another because the second script may not be downloaded yet.

# Introducing jQuery

jQuery is a library that is built on JavaScript. The underlying code is actually JavaScript; however, jQuery simplifies a lot of the JavaScript code into simple-to-use functionality. The two main advantages to using jQuery are selectors and built-in functions.

Selectors provide quick access to specific elements on the web page, such as list and tables. Selectors also provide access to groups of elements, such as all paragraphs, or all paragraphs of a certain class. This allows you to quickly and easily access specific Document Object Model (DOM) elements.

jQuery also provides a rich set of built-in functionality that makes it easy to do a lot more with a lot less code. For example, tasks such as hiding an element on the screen or animating the resize of an element take just one line of code.

## Loading jQuery in Your Web Pages

```
<head>
  <script src="local/jquery-2.0.3.min.js"></script>
. . . or . . .
  <script src=
    "http://code.jquery.com/jquery-2.0.3.min.js">
  </script>
</head>
```

The jQuery library is just a `.js` file. You can load it just like any other JavaScript file. There are two ways to add jQuery to your web page.

The easiest method of adding jQuery to web pages is to use one of the several Content Discovery Network locations, or CDNs. A CDN allows you to load the libraries from a network of jQuery hosting servers spread globally. The benefit of this method is that the servers are spread globally so the downloads are distributed to multiple servers. Also, if the user has loaded another web page with a link to the CDN file, it may already have the jQuery library cached. The following are some examples of hosting CDNs:

```
//jQuery
<script src=
  "http://code.jquery.com/jquery-2.0.3.min.js">
</script>
<script src=
  "http://code.jquery.com/jquery-migrate-
➥1.1.0.min.js">
</script>
//google
<script src=
  "https://ajax.googleapis.com/ajax/libs/jquery/
➥2.0.3/jquery.min.js">
</script>
//Microsoft
<script src=
  "http://ajax.aspnetcdn.com/ajax/jQuery/jquery-
➥2.0.3.min.js">
</script>
```

The other option is to download the `.js` file from http://jquery.com/download and load it with your other JavaScript libraries. This has the advantage of being more closely tied to your site's content so you don't have to worry about possible unavailability issues or site-blocking issues with the external locations.

## Accessing jQuery in Your JavaScript

```
jQuery("#myElement")
. . . or . . .
$("#myElement")
```

jQuery is accessed using the jQuery object that is defined when the library is loaded. jQuery also provides a shortcut `$` character that you can use in the phrases throughout the book. For example, the following two jQuery statements are identical:

```
jQuery("#myElement")
$("#myElement")
```

# Introducing jQuery UI

jQuery UI is an additional library built on top of jQuery. The purpose of jQuery UI is to provide a set of extensible interactions, effects, widgets, and themes that make it easier to incorporate professional UI elements in your web pages.

jQuery UI is made up of two parts: JavaScript and Cascading Style Sheets (CSS). The JavaScript portion of jQuery UI extends jQuery to add additional functionality specific to adding UI elements or applying effects to those elements. The CSS portion of jQuery UI styles the page elements so that developers don't need to style the elements every time.

jQuery UI saves developers time by providing prebuilt UI elements, such as calendars and menus, with interactions, such as dragging and resizing, right out of the box.

## Getting the jQuery UI Library

To get started with jQuery UI, you need to download the library and add it to your web pages. You can download the jQuery library from http://jqueryui.com/download/ by selecting the options that you would like to include and clicking on the Download button at the bottom. This downloads the jQuery UI files.

When you download the jQuery UI library, you get a zip file. Inside that file are three main folders that you need to understand. They are

- **`js`**—This is the folder that contains the jQuery UI and jQuery libraries files. You need to deploy these files so you can load them in your web pages.
- **`css`**—This contains the theme-named folders that house the `.css` files and an `images` folder used by the jQuery UI library. The .css file and images/ folder need to be placed in the same location and accessible from your web pages.
- **`development-bundle`**—This folder contains the full source for jQuery UI. If you are not planning to modify the jQuery UI code, you can ignore this folder.

## Loading jQuery UI

```
<head>
  <link rel="stylesheet" type="text/css"
    href="local/jquery-ui-1.10.3.css">
  <script src="local/jquery-2.0.3.min.js"></script>
  <script src="local/jquery-ui-
➥1.10.3.min.js"></script>
. . . or  for CDN. . .
  <link rel="stylesheet" type="text/css"
    href="http://code.jquery.com/ui/1.10.3/themes/
➥base/jquery-ui.css">
  <script src=
    "http://code.jquery.com/jquery-2.0.3.min.js">
  </script>
  <script src=
    "http://code.jquery.com/ui/1.10.3/jquery-ui.js">
  </script>

</head>
```

To load jQuery UI, you first need to load the jQuery library. The jQuery UI is also a `.js` file. You can load it just like any other JavaScript file. Also, just like jQuery, you can load the script from an external CDN source or download the library and load it locally.

You also need to load the jQuery UI `.css` file using a `<link>` tag. This can be a local file or an external CDN location. For example:

```
<link rel="stylesheet" type="text/css"
  href="local/jquery-ui-1.10.3.min.css">
. . . or  for CDN. . .
<link rel="stylesheet" type="text/css"
  href="http://code.jquery.com/ui/1.10.3/themes/
➥base/jquery-ui.css">
```

# Introducing jQuery Mobile

Mobile devices are the fastest growing development platform. Much of that development is geared toward making web sites mobile friendly. It is much easier to implement and maintain a mobile web site than it is to maintain a mobile application.

jQuery mobile is an additional library built on top of jQuery. It is designed to streamline many of the necessary structural, UI, and event implementations to build mobile web sites. jQuery Mobile provides several advantages with developing mobile solutions, including

- Hiding some of the complexities of resizing page elements to a wide array of mobile devices.
- Providing simple UI components with mobile event interactions already built into them.

- Building on JavaScript and jQuery provides a common and well developed platform that is based on proven concepts.
- Developing a mobile web apps is simple to do and does not require any installation of the user's part. That is why they are becoming more and more popular.

## Getting the jQuery Mobile Library

To get started with jQuery Mobile, you need to download the library and add it to your web pages. You can download the jQuery library from http://jquerymobile.com/download/. You can also download jQuery Mobile from http://jquerymobile.com/download-builder/ by selecting the version and options that you would like to include and clicking on the Download button at the bottom. This downloads a zip file containing the jQuery Mobile library.

**Watch out!**

The `.css` files and the `images` folder that are included with the jQuery Mobile library come as a set. You need to make sure they are installed in the same location and you don't mix and match them from different custom downloads.

When you download the jQuery Mobile library, you get a zip. Inside the zip file are three main components that you need to put where your mobile web pages can load them. They are

- **`js` files**—There will be a `jquery.moble.###.js` as well as a minified version. This is the main library file, and one of these needs to be placed where you can add it to your project files in a `<script>` tag.
- **`css` files**—There will be `jquery.mobile.###.css`, `jquery.mobile.###.structure.css`, and `jquery.mobile.###.theme.css` files as well as their minified forms. This is all the styling code and should be placed in the same location as the `jquery.moble.###.js` file.
- **`images` folder**—This folder contains the images and icons used by jQuery Mobile to style the elements. This should also be placed in the same location as the `jquery.moble.###.js` file.

## Loading jQuery Mobile

```
<head>
  <link rel="stylesheet"
    href="local/jquery.mobile-custom.min.css" />
  <script src="local/jquery-2.0.3.min.js"></script>
  <script src="local/jquery.mobile-
➥custom.min.js"></script>
. . . or  for CDN . . .
  <link rel="stylesheet" href=
    "http://code.jquery.com/mobile/1.4.0/jquery.
➥mobile-1.4.0.min.css" />
  <script src=
   "http://code.jquery.com/jquery-
➥1.8.2.min.js"></script>
  <script src=
   "http://code.jquery.com/mobile/1.4.0/jquery.
➥mobile-1.4.0.min.js">
   </script>
</head>
```

Similar to what you did for jQuery UI, you need to load jQuery before loading jQuery Mobile. Once jQuery is loaded, you can load the jQuery Mobile `.js` file from an external CDN source or a locally downloaded version of the library.

You need to load the jQuery Mobile `.css` file as well using a `<link>` tag. For example, the following code loads the jQuery library first because it is required by jQuery Mobile and then loads the jQuery Mobile JS and CSS files:

```
<script type="text/javascript"
  src="local/jquery-2.0.3.min.js"></script>
<script type="text/javascript"
  src="local/jquery.mobile-custom.min.js"></script>
<link rel="stylesheet"
  href="local/jquery.mobile-custom.min.css" />
```

## Configuring Browser Development Tools

An important aspect of developing JavaScript and jQuery is using the web development tools incorporated in web browsers. These tools allow you to see the script files loaded, set breakpoints, step through code, and much, much more. It is beyond the scope of this book to delve too much into the browser tools. However, I wanted to provide you with the steps to enable them and encourage you to learn about them if you have not already done so.

## Installing Firebug on Firefox

Use the following steps to enable JavaScript debugging on Firefox:

1. Open Firefox.
2. Select Tools > Add-Ons from the main menu.
3. Type `Firebug` in the search box in the top right to search for Firebug. Then click the Install button to install it.
4. Type `FireQuery` in the search box in the top right to search for FireQuery. Then click the Install button to install it. FireQuery extends Firebug to also support jQuery.
5. When you reload Firefox, click on the Firebug button to display the Firebug console.

## Enabling Developer Tools in Internet Explorer

Use the following steps to enable JavaScript debugging on Internet Explorer:

1. Open IE.
2. Click on the Settings button and select Developer Tools from the drop-down menu. Or you can press the F12 key.
3. The Developer console is displayed.

## Enabling the JavaScript Console in Chrome

Use the following steps to enable JavaScript debugging in Chrome:

1. Open Chrome.
2. Click on the Settings button and select Tools > Developer Tools from the drop-down menu. Or you can press Ctrl+Shift+J on PCs or Cmd+Shift+J on Macs.
3. The JavaScript console is displayed.

2

# Using the JavaScript Language

The phrases in this chapter focus on various ins and outs of the JavaScript language. You need to know these to be able to fully utilize the full capabilities that jQuery and JavaScript provide in the HTML stack.

## JavaScript Syntax

As a programming language, JavaScript, and consequently jQuery since it is based on JavaScript, requires a specific syntax. This section gives you a quick primer on the JavaScript syntax before going into the language details.

You should be familiar with the following rules regarding JavaScript syntax:

- All JavaScript statements end with a semicolon.
- Statements execute in the order they appear in the script.

- Blocks of code are defined using `{CODE_BLOCK}` brackets.
- Expressions bound in `()` brackets are evaluated before they're applied to the rest of the statement.
- JavaScript is case sensitive.
- To break a code line in the middle of a string, \ must be the last character on the line.
- Strings in JavaScript must be enclosed in either single (`'string'`) or double (`"string"`) quotes.
- When adding a string to a number, the resulting value is a string.
- Variables must begin with a letter, `$`, or `_`.

# Defining and Accessing Data

One of the most important aspects of JavaScript is the ability to define variables that represent different forms of data, from individual numbers and strings to complex objects with properties and methods to arrays containing several items. Once data has been defined, you can use and reuse it for a variety of purposes. This section provides phrases that help you create the various data structures that you will need to use in JavaScript.

## Defining and Accessing Variables

```
var x = y = 5;
var z = 10;
var results = "x + y + z = " + (x+y+z);
var s1 = "jQuery";
var s2 = "JavaScript"
var s3 = s1 + " & " + s2;
```

JavaScript makes it easy to define variables. Variables are assigned using `var name = value;` syntax. The `var` keyword tells JavaScript that this is a new variable name. Once the variable has been assigned a value, you can access it using the variable name. You can define multiple variables on a single line using `var name1 = name2 = value;` syntax.

When you assign a variable to an expression, such as `x + y + z`, the expression is evaluated before assigning the variable a value. The following code snippet shows you how to define and access different variables.

```
<script>
  var x = y = 5;
  var z = 10;
  var results = "x + y + z = " + (x+y+z);
  var s1 = "jQuery";
  var s2 = "JavaScript"
  var s3 = s1 + " & " + s2;
  document.write(results);
  document.write("<br>");
  document.write(s3);
</script>
```

*ch0201.html*

```
x + y + z = 20
jQuery & JavaScript
```

*Output from ch0201.html*

## Creating Arrays

```
var x = y = 5;
var z = 10;
var results = "x + y + z = " + (x+y+z);
var s1 = "jQuery";
var s2 = "JavaScript"
var s3 = s1 + " & " + s2;
```

You can create arrays in a few different ways. You can create them in the definition using `[]` brackets, such as `var arr=[1,2,3]`. You can also create a new array object and add items to it using the `push()` method. You can access items in the array using the `arr[index]` method. The following code snippet shows an example of creating an array in each of these ways.

```
<script>
  var weekdays = ["Mon", "Tue", "Wed", "thur",
➥"Fri"];
  var weekend = new Array();
  weekend.push("Sat");
  weekend.push("Sun");
  document.write(weekdays.toString() + "<br>");
  document.write(weekend[0] + ", " + weekend[0]);
</script>
```

*ch0202.html*

```
Mon,Tue,Wed,thur,Fri
Sat,Sun
```

*Output from ch0202.html*

## Creating Objects

```
  var obj1 = {name:"Brad", title:"Author", "last-
➥name":"Dayley" };
  var obj2 = new Object();
  obj2.name = "Frodo";
  obj2.title = "Hobbit";
  obj2["last-name"] = "Baggins";
```

You can create objects in the definition using `{property:value, ..}` syntax, such as `var obj={name:"Brad", title:"Author"};`. You can also create a new object and add properties to it using standard dot syntax.

The following code snippet shows an example of creating an array in each of these ways.

```
<script>
  var obj1 = {name:"Brad", title:"Author", "last-
➥name":"Dayley" };
  var obj2 = new Object();
  obj2.name = "Frodo";
  obj2.title = "Hobbit";
  obj2["last-name"] = "Baggins";
  document.write(obj1.name + " " + obj1["last-name"]
➥+ ", " + obj1.title + "<br>");
  document.write(obj2.name + " " + obj2["last-name"]
➥+ ", " + obj2.title);
</script>
```

*ch0203.html*

```
Brad Dayley, Author
Frodo Baggins, Hobbit
```

*Output from ch0203.html*

# Defining Functions

```
function add(a, b){
  return a + b;
}
var result1 = add(5, 20);
```

You need to be familiar with creating and using functions in JavaScript. Functions are defined using the following syntax:

```
function function_name(arguments){function_block}
```

You can specify any number of arguments, and because JavaScript is loosely typed, you can pass in different object types each time you call the function. To call the function, simply reference it by name and pass in the desired arguments enclosed in `()` brackets. The code in the `function` block is then executed. You can use the `return` keyword to return a value from the function call.

The following code snippet shows examples of defining and calling functions:

```
<script>
  function add(a, b){
    return a + b;
  }
  var result1 = add(5, 20);
  var result2 = add("Java", "Script")
  document.write(result1 + "<br>");
  document.write(result2);
</script>
```

*ch0204.html*

```
25
JavaScript
```

*Output from ch0204.html*

# Manipulating Strings

Strings are one of the most important data structures in JavaScript. Strings represent data that is conveyed to the user on the web page, whether it is a paragraph element, the name on a button, a menu label, or a figure caption. Strings are also used in the background to define locations, filenames, and a variety of other values used in web elements.

There are several special characters to define values that cannot be represented otherwise, such as new lines and tabs. Table 2.1 lists the special characters in JavaScript strings.

Table 2.1 **String Object Special Character**

| Character | Description |
|---|---|
| `\'` | Single quote |
| `\"` | Double quote |
| `\\` | Backslash |
| `\n` | New line |
| `\r` | Carriage return |
| `\t` | Tab |
| `\b` | Backspace |
| `\f` | Form feed |

## Getting the Length of a String

```
var s = "One Ring";
s.length; //returns 8
var s2 = "To Rule\nThem All";
s2.length; //returns 16 because \n is only 1
➥character
```

String objects have a length attribute that contains the number of characters in the string. This includes the number of special characters as well, such as `\t` and `\n`. To get the length of the string, simply use the `.length` attribute on the string object.

## Finding What Character Is at a Specific Location in a String

```
var s = "In The Darkness Bind Them";
s.charAt(7); //returns 'D'
s[7]; // also returns 'D'
```

String objects provide the `charAt(offset)`, which returns the character contained at the offset specified. Also, strings in JavaScript are basically arrays of characters, so you can get the character at a specific offset using `stringName[offset]`. Keep in mind that the offsets are always zero based, so their first character is at offset 0.

## Converting Numbers to Strings

```
var n=16;
var a = n.toString();  //a = 16
var b = n.toString(16); //b = 10
```

To convert a number to a string in JavaScript, create a variable with the numerical value if you don't already

have one and then call `.toString(radix)` on the number. The optional `radix` parameter specifies the base to use when converting the number. You can specify base `2` up to base `36`.

## Converting Strings to Numbers

```
new Number("16");   //returns 16
new Number("0x20"); //returns 32
new Number("32.8"); //returns 32.8
```

To convert numerical-based strings into a number, create a new number object. JavaScript automatically detects whether the string is a number (even hex number with `0x##` formatting) and create a new number object. If the string is not a valid number format, a `NaN` object is returned.

## Combining Strings

```
var str1="jQuery";
var str2=" & ";
var str3="JavaScript";
var str4 = str1.concat(str2, str3); //str5 = "jQuery
➥& JavaScript"
var str5 = str4 + " Phrasebook";    //str5 = "jQuery
➥& JavaScript Phrasebook"
```

You can combine multiple strings using the `.concat(str, str, …)` method. Or you can just use the `str + str + str . . .` method.

## Changing String Case

```
var s1 = "jQuery and JavaScript";
var s2  = s1.toLowerCase();  //s2 = "jquery &
➥javascript"
var s3  = s1.toUpperCase(); //s3 = "JQUERY &
➥JAVASCRIPT"
```

Strings have built-in functions to change the case. The `.toLowerCase()` method returns a lowercase version of the string. The `.toUpperCase()` method returns an uppercase version of the string.

## Splicing Strings

```
var s1 = "The play's the thing";
var s2 = s1.splice(4,8); // s2 = "play"
```

You can carve a string into substrings using `.splice(start, end)` by specifying the starting index and the ending index. The section of the string starting with the character at the index specified by `start` and ending with the character just before `end` index is returned. All indexes are zero based.

## Splitting Strings

```
var s1 = "on-the-way-to-the/forum";
var arr = s1.split("-"); // arr = ["on", "the",
➥"way", "to", "the","forum"];
```

To split a string into chunks using a specific delimiter, use the `.split(separator [, limit])`. The separator defines where to split the string and is not included in the results. The limit specifies the number of items to return before stopping. An array of the split portions of the string is returned.

## Checking to See If a String Contains a Substring

```
var s1 = "I think therefore I am";
var a = s1.indexOf("think");     // a = 2
var b = s1.indexOf("thought"); // b = -1
```