IN THIS CHAPTER

- Working with images
- Playing audio
- Watching videos

Chapter 5
HTML5 and Multimedia

Multimedia has taken over the Internet. Thanks to the popularity of websites like YouTube, these days if your website doesn't support some type of multimedia content, your visitors will consider it old school and may pass it by. This chapter examines the multimedia features available in HTML5 and shows you how to implement images, audio, and video in your dynamic web applications.

Working with Images

The most basic type of multimedia to put on a web page is a picture. The old saying "a picture is worth a thousand words" is somewhat true, especially in the web world. Placing images on your web page can help break up the monotony of plain text, as well as help add to your content in an attractive manner. Often, the first thing a new website visitor will notice are the images.

The HTML standard has always supported placing images within web pages, but there are a few new tricks that you can try using HTML5 and CSS3 to make your images stand out. This section shows just how to do that.

Placing images

The *img element* allows you to place an image file on the web page. The img element uses a one-sided tag, `<img>`, that uses attributes to define the image and how the browser should display it.

Here's the basic format for the `<img>` tag:

```
<img src="location" alt="text" width="x" height="y">
```

The `src` attribute defines the location of the image file to display. You can specify the location as a relative or absolute file path for images stored on the same server as the web page, or you can use a URL to reference images stored on another server.

The `alt` attribute defines alternative text that appears if the browser can't display the image, such as if the image file is missing if the browser doesn't support displaying images (such as a text-based browser), if your site visitor is using a screen reader, or if your website is being read by a search engine. For all your images, it's a good idea to provide a good description of not only the image, but also any action that occurs in the image. You do this in the alternative text attribute.

By default, the browser displays the image at full size in the browser window. That may not always be what you want, or you may just want more control over how or where the image appears. To help control that you use the `width` and `height` attributes to define a specific viewing area for the image to fit into.

Alternatively, instead of using the `width` and `height` attributes, you can use the `style` attribute and define the `width` and `height` as style properties:

```
<img src="myimage.jpg" alt="My image" style="width: x; height: y;">
```

Either method is allowed in HTML5, although using an inline style will help prevent accidental styling of the image from an external style sheet.

WARNING

Browsers are able to display most image types these days, but some image types are more suited for web pages than others. The JPEG image type is commonly used on web pages because it compresses the image to a smaller file size, making it quicker to download to the client browser. Using image files that are too large may ruin the experience for some of your website visitors, especially those who are using mobile data connections. No one likes having to wait for an image to load on a web page.

Styling images

The CSS3 standard defines some additional styles that you can apply to the images on your web page to make them stand out even more. In Book 2, Chapter 4, I demonstrate how you can use the CSS3 shadow effect on elements to help give them a 3D effect. You can use that effect on images on the web page, too. That adds a nice touch to help make the image pop out from the web page.

Another handy style added by the CSS3 standard is the `transform` property. The `transform` property allows you to alter how an image appears on the web page, such as scale it, rotate it, or even skew it! There are functions for both 2D and 3D manipulation of the images. Table 5-1 lists the 2D transform effects that are available.

TABLE 5-1 The CSS3 2D Transform Effects

Effect	Description
`matrix(a,b,c,d,e,f)`	Combines the translation, scale, skew, and rotation effects in one property
`rotate(angle)`	Rotates the object clockwise by the specified angle
`scale(x,y)`	Resizes the object by a factor of *x* horizontally and *y* vertically
`scaleX(x)`	Resizes the object horizontally only by a factor of *x*
`scaleY(y)`	Resizes the object vertically only by a factor of *y*
`skew(x,y)`	Offsets the object horizontally by an angle of *x* and horizontally by an angle of *y*
`skewX(x)`	Offsets the object horizontally only by an angle of *x*
`skewY(y)`	Offsets the object vertically only by an angle of *y*
`translate(x,y)`	Moves the object *x* pixels to the right and *y* pixels down
`translateX(x)`	Moves the object *x* pixels to the right
`translateY(y)`	Moves the object *y* pixels down

The `rotate()` function is one of my favorites. Just by adding the `rotate()` function to a standard image, you can help make it stand out from the text content on the web page. You can try that out yourself by following these steps:

1. **Open your favorite text editor, program editor, or integrated development environment (IDE) package.**

2. **In the editor window, type the following code:**

```
<!DOCTYPE html>
<html>
<head>
<title>Image Rotation Test</title>
<style>
    #img1 {
        float: left;
        transform: rotate(30deg);
        box-shadow: black 10px 5px 15px;
    }

    #img2 {
        float: left;
        transform: rotate(-30deg);
        box-shadow: black 10px 5px 15px;
    }
</style>
</head>
<body>
<h1>Testing the image rotation feature</h1>
<header>
<h1>My vacation photos</h1>
</header>
<section>
<img id="img1" src="image1.jpg" width="50" height="50">
<img id="img2" src="image2.jpg" width="50" height="50">
</section>
</body>
</html>
```

3. **Save the file as `imagetest.html` in the `DocumentRoot` folder for your web server.**

   For XAMPP on Windows, that's `c:\xampp\htdocs`. For XAMPP on macOS, that's `/Applications/XAMPP/htdocs`.

4. **Find two of your favorite image files and copy them to the same folder as the `imagetest.html` file.**

   You'll need to either rename them as `image1.jpg` and `image2.jpg` or change the code in the `imagetest.html` file to match your image filenames.

5. **Start the web server if necessary.**

6. **Open a browser and go to the URL for the file.**

   If you're using the XAMPP server set to TCP port 8080, use the following:

```
http://localhost:8080/imagetest.html
```

7. **Close the browser and shut down the web server.**

   The code in the `imagetest.html` file places two images on the web page. The first image is rotated by a negative value so that it rotates counterclockwise; the second image is rotated by a positive value so that it rotates clockwise. Figure 5-1 shows the results using my test image.

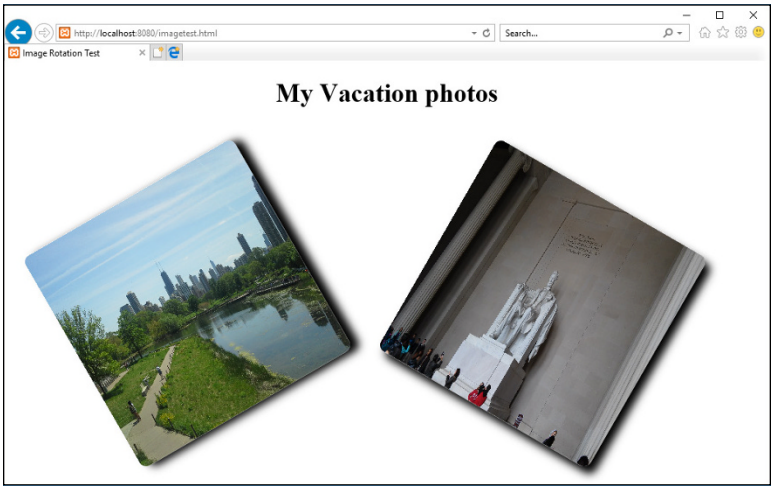

FIGURE 5-1: Rotating images on the web page.

That's a great start to a professional-looking website!

Linking images

You can also use images as links to other web pages or locations. You do that by embedding the `<img>` tag inside an anchor element:

```
<a href="childrens.html">
<img src="children.jpg" alt="Children's clothes"
style="width: 50px; height: 50px; border: 0px;">
</a>
```

If the website visitor clicks anywhere on the image, the browser responds just as if the anchor element was a hypertext link, redirecting the browser to the destination defined by the `href` attribute.

TIP

It's also a good idea to add the `border` style property to the anchor style element and set it to 0 pixels to prevent the browser from drawing an ugly border around the image to indicate that it's a link.

Working with image maps

Linked images are nice, but how about those fancy map images that allow you to click in different parts of the map to go to different locations? You do that by using image maps. An *image map* allows you to define sections of an image that act just like a hyperlink. You can define each section to redirect the visitor's browser to a different location.

Creating an image map requires that you first define the map and then apply it to your image. To define the map, you use the *map element:*

```
<map name="mapname">
   map area definitions
</map>
```

The `name` attribute is important, because you'll use that to reference the map from the image `<img>` tag in the `usemap` attribute:

```
<img src="location" usemap="#mapname">
```

After you define the map, you need to define one or more map areas. Each map area defines a specific location on the image to create a *hotspot* (clickable region). You define the map areas using the `<area>` tag:

```
<area shape="shape" cords="coordinates" href="location" alt="text">
```

The combination of the shape and coordinates defines the area in the image for the hotspot. Table 5-2 shows how to match those up.

TABLE 5-2 Defining the Area Element Hotspots

Shape Value	Description
`circle`	Defines the *x* and *y* location of the circle center, as well as the radius value
`poly`	Defines multiple *x* and *y* locations for each point of the polygon
`rect`	Defines the *x* and *y* coordinates for the upper-left corner and the lower-right corner
`default`	No coordinates necessary; uses the remaining unmapped area of the image

Defining an image map can be somewhat difficult. You'll need to know the exact size of the image on the web page and be able to define the exact location for each area hotspot. This is where a good image manipulation tool such as Photoshop or GIMP can come in handy. Anything that allows you to count pixels in the image will help you plot out the hotspots.

Here's an example of defining an image map for an image:

```
<map name="storemap">
<area shape="rect" coords="0, 0, 100, 500" href="books.html" alt="shop our
   books">
<area shape="rect" coords="101, 0, 200, 500" href="furniture.html" alt="shop our
   furniture">
<area shape="rect" coords="201, 0, 300, 500" href="clothes.html" alt="shop our
   clothes">
<area shape="rect" coords="301, 0, 400, 500" href="tools.html" alt="shop our
   selection of tools">
<area shape="rect" coords="401, 0, 500, 500" href="food.html" alt="shop for some
   groceries">
</map>
```

After you define the image map, you associate it to an image by adding the `usemap` attribute to the `<img>` tag:

```
<img src="store.html" alt="our store" usemap="#storemap">
```

The `#storemap` value references the `storemap name` attribute, so the browser applies that image map to the image on the web page. Clicking each individual section takes you to the associated `href` location defined for that area.

Using HTML5 image additions

Besides the standard HTML image features, HTML5 adds a couple of new image features that you can use in your web pages.

Figures and captions

It's common to want to place captions around images that you display on the web page. You can do that with standard HTML and CSS, but it takes some calculating to get the positioning correct, and if anything on the web page moves, the image and caption may get out of sync.

HTML5 adds the *figure element* to match images and captions together. The figure element encloses the image, along with a figcaption element, creating a single

object that you can position and move around on the web page. Here's the general format for all that:

```
<figure>
<img src="image.jpg" alt="Figure 1">
<figcaption>Figure 1: Creating a web page</figcaption>
</figure>
```

You can now add styles for the figure element to position both the image and its associated caption on the web page together as a single object. In this example, the caption will appear under the image. If you prefer, you can place the `<figcaption>` tag above the image, too, by just listing it before the `<img>` tag.


TIP

You can use the figure element to link other objects with captions, too. For example, use the p element instead of the img element for embedding quotes inside a text section and linking them to the citation for the quote in the figcaption element.

The picture element

The `<img>` tag, along with the transform CSS property, allows you to scale images to fit a specific area on the web page:

```
img {
   transform: scale(80,60);
}
```

This solution doesn't always produce the best-quality image for the device. With website visitors using a multitude of different devices, each with a different aspect ratio and screen size, it's hard to get one image to work in all situations.

The HTML5 standard has a solution for that problem. Instead of trying to scale one image to fit everywhere, you can define multiple versions of an image to display for different environments. You just need to define the environment parameters for the browser to test to know which image to display. You do that using the *picture element*.

The HTML5 picture element allows you to define one or more sources for the image, along with defining media rules to determine when each source should be used. I cover media rules in Book 2, Chapter 4, where I discuss how to use media rules to load different style sheets based on different properties of the website visitor's device. This is the same concept.

The picture element uses the `<picture>` tag, along with one or more source elements. Each source element defines a media rule and the image to use if the device meets the media rule criteria. The format for all that looks like this:

```
<picture>
<source media="(min-width: 1000px)" srcset="large.jpg">
<source media="(min-width: 500px)" srcset="small.jpg">
<img src="original.jpg" alt="My image">
</picture>
```

When the browser sees the picture element, it evaluates each of the source elements inside, from the first to the last. The first source element that matches the media environment is used to display the image defined in the `srcset` attribute. If none of the media tests defined in the source elements passes, the browser uses the image defined in the `<img>` tag.

Playing Audio

The original HTML standard didn't account for playing audio clips in web pages. That created a free-for-all of methods developed to incorporate audio. Many different solutions were created along the way.

One such solution is to reference an audio file stored on the server using a standard anchor element:

```
<a href="myaudio.mp3">Click to play</a>
```

When the site visitor clicks the hypertext link, the browser downloads the audio file and opens an appropriate audio player from the workstation to play it. That's a pretty clunky way of trying to incorporate audio into a web page.

The following sections discuss better ways of playing audio files in your web pages.

Embedded audio

The next step in the evolution of playing audio in web pages was the plugin. A *plugin* is a separate program that runs inside the browser to support additional features. Over the years, several different audio plugins had been developed, but the three most common were

- **QuickTime:** A plugin developed by Apple, used mainly in the Safari web browser.
- **RealAudio:** A vendor-neutral attempt to create an audio plugin. It only supports its own proprietary audio file format.
- **Flash:** Developed by Adobe, Flash became a popular format for playing both audio and video files in browsers.

To play an audio file using a plugin, you had to use the *embed element*, which signals to the browser that some type of external file is embedded inside the web page and to find the appropriate plugin to handle the embedded file. The embed element uses the one-sided `<embed>` tag, with the following format:

```
<embed src="location" type="mime" width=x height=y>
```

The `src` attribute defines the location of the audio file, either as an absolute or relative path on the local server, or as a URL to point to an audio file stored on a remote server.

The `width` and `height` attributes are used if the plugin requires space on the web page to display an interface. Some audio plugins provide an interface to stop, start, and pause playing the audio file.

The `type` attribute defines the type of multimedia file. It uses the standard *Multimedia Internet Mail Extension* (MIME) type names to identify the audio file type. As the name suggests, MIME types were originally developed for sending binary files through email, but they're also used by web browsers for embedded binary files in web pages. The browser uses the MIME type to determine just what plugin to use to process the embedded file. Different audio file formats (such as QuickTime or RealAudio) require a different plugin to play. This is where it helps to know the different formats available for digital audio files.

Digital audio formats

Since the transition from vinyl records to the digital world, many different methods had been used for converting analog sound to digital media. The process of converting sound to digital signals consists of three elements:

- **Sampling rate:** How often the sound amplitude is measured and quantified to a digital value
- **Sample resolution:** How many bits of data are used for each sample digital value
- **Compression:** How the final digital data is compressed to make an audio file

The combination of the sampling rate and sample resolution result in the *bit rate* used for the digital recording. The larger the bit rate, the more accurately the digital playback will produce the original analog signal. However, the larger the bit rate, the more digital data that is generated and, thus, required to store the audio file.

A standard CD format for digital audio uses a bit rate of 1411 Kbps, which results in an extremely accurate reproduction, but also an extremely large file size (around 10MB) for each audio track. It's impractical to send a standard CD audio track across the Internet to play on client browsers.

This is where compression comes into play. Because the original audio files are too large to use on the web, we need to incorporate some type of compression scheme to make them more manageable. Unfortunately, over the years, many different companies have developed proprietary compression techniques, resulting in our current myriad of different audio file types. Table 5-3 lists the more common audio file types in use, along with their file extensions and MIME types.

TABLE 5-3 **Audio File and MIME Types**

Audio	File	MIME	Description
AAC	`.aac`	audio/aac	Apple audio coding
AU	`.au`	audio/basic	Sun Microsystems standard for Unix systems
MIDI	`.mid`	audio/mid	Musical Instrument Digital Interface standard for recording instruments
MP3	`.mp3`	audio/mpeg	Motion Picture Experts Group standard
Ogg Vorbis	`.ogg`	audio/ogg	Open-source standard
RealAudio	`.ra`	audio/x-pn-realaudio	Real Media standard for streaming audio
SND	`.snd`	audio/basic	SouND format, developed by Apple based on the AU audio format
Shockwave Flash	`.swf`	audio/x-shockwave-flash	Adobe proprietary format
WAV	`.wav`	audio/wav	Waveform Audio format developed by Microsoft for uncompressed audio

Each of these audio file formats has its own pros and cons to deal with. Different audiophiles have different opinions on which method is the best. To make matters more complicated, many of these audio file formats are proprietary and require a license to embed a player in a browser. The MP3 audio type has become the de facto

standard over the years due to its high compression rate and high audio quality, but many developers don't like using MP3 because of its proprietary nature.

The Ogg Vorbis audio type is an open-source standard, free to use in any environment. However, it hasn't been widely adopted by all browsers yet (including Internet Explorer and Safari) due to its perceived lack of audio quality compared to other compression methods.


TIP

At the time of this writing, the only audio file type supported by all the major browsers is MP3. However, if your site visitor is using a Linux workstation, due to licensing restrictions, the MP3 codecs are often not loaded by default on all Linux distributions.

If you embed an audio file into your web page and a visitor doesn't have the appropriate plugin to handle it, the browser will react in one of three ways:

- Display an error message in place of the embedded audio file
- Display a pop-up message indicating the plugin required to play the audio file
- Redirect the web page to the web page for the required plugin

All three of these results are less than optimal for your web page. Fortunately, the HTML5 standard has produced a better way to incorporate sounds into web pages.

Audio the HTML5 way

The key to embedding audio files into your web pages is similar to how you handle displaying images — it's best to have multiple versions available and let each site visitor's browser decide which one to use. The HTML5 standard provides a way for you to do that with the *audio element.*

The audio element works just like the picture element (see the "Using HTML5 image additions" section, earlier in this chapter). You use the `<audio>` tag to open a list of audio sources, defined using the `<source>` tag. Each source specifies a different audio file format for the browser to try. The first one in the list that the browser supports is what gets requested and played by the browser. That looks like this:

```
<audio>
<source src="myaudio.mp3" type="audio/ogg">
<source src="myaudio.ogg" type="audio/mpeg">
<source src="myaudio.wav" type="audio/wav">
</audio>
```

For the source elements, you use the `src` attribute to indicate the location of the audio file to play, as well as the `type` attribute to indicate the MIME type of the audio file. It's up to the browser to decide which one to use. In this example, the browser attempts to play the MP3 version first. Then for Linux workstations where that's not available, it automatically attempts to load and play the Ogg Vorbis version. If that's not available, it'll try to use the WAV version of the audio file.

WARNING

The HTML5 standard only supports the MP3, Ogg Vorbis, and WAV audio file types. Don't try to use the audio element to embed a QuickTime or RealAudio audio file. You'll need to use the embed element to do that.

You can also place a short message after the `<source>` tag list for the browser to display if it can't support any of the listed MIME types:

```
<audio>
<source src="myaudio.mp3" type="audio/ogg" controls>
<source src="myaudio.ogg" type="audio/mpeg" controls>
Sorry, your browser doesn't support MP3 or OGG audio
</audio>
```

The `<audio>` tag has a few attributes to help alter how the browser handles the audio file. Table 5-4 shows the available attributes.

TABLE 5-4 The <audio> Tag Attributes

Attribute	Description
`autoplay`	The browser should start playing the file as soon as the web page loads.
`controls`	The browser should display a standard set of audio controls, such as Play, Stop, and Pause buttons.
`loop`	The browser should continually loop the audio file.
`muted`	The browser should mute the audio track immediately.
`preload`	Specifies whether the audio file should be loaded when the page loads or when the Play button is clicked.
`src`	Specifies the URL of an audio file when not using additional `<source>` tags.

The `controls` attribute is recommended, because it provides an interface for the website visitor to have control over how or when the audio file plays. Each browser has its own way of displaying audio controls. Figure 5-2 shows how the controls appear in the Internet Explorer browser.

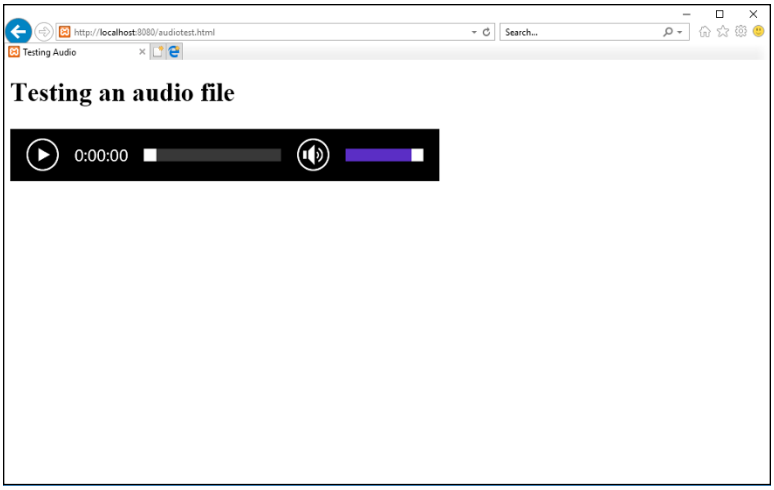

FIGURE 5-2: The audio controls in the Internet Explorer browser.

The audio controls shown by browser are fairly simplistic — a Play/Pause button, a Mute button, a slider to control the location in the audio file, and a sound level slider. Don't expect any fancy EQ settings to bump up the bass in your tunes!

WARNING

In the past, it was somewhat commonplace to embed an audio file in a web page and set it to automatically play with the loop feature enabled. This is a surefire way to annoy your site visitors, and it may even cause issues for visitors who use a screen reader to process your web page. I don't recommend using this method. Allow your site visitors the option of whether to play the audio embedded on your web pages.

Watching Videos

These days, the world is full of video content. You can find videos on just about every topic under the sun, including how to make your own videos! This section walks through how you can embed videos in your web pages, but first, a quick look at the different types of video files you may have to deal with.

Paying attention to video quality

Just like in the world of film, videos are composed of a series of individual images (called *frames*) played at a set rate of speed (called the *frame rate*). The higher the frame quality, the better the video quality. The higher the frame rate, usually the better the video quality (with exceptions, as noted in this section).

You can use any frame size for the video images, but there are standard frame sizes that are commonly used:

- **160 x 120:** Low-quality video using the 3:4 aspect ratio
- **320 x 240:** Higher-quality video, but still using the 3:4 aspect ratio
- **640 x 480:** Highest-quality video using the 3:4 aspect ratio
- **1280 x 720:** High-definition (HD) video using the 16:9 aspect ratio
- **1920 x 1080:** HD-quality video using the 16:9 aspect ratio

As you can guess, a higher quality of frame images means a larger video file.

The frame rate used for television video is 60 frames per second (fps). That frame rate would create a huge video file. DVD-quality videos use 24fps as a compromise between video quality and file size.

A fast frame rate isn't necessarily a good thing with web video. As the video is sent from the server to the client browser, the network gets in the way. Usually, to help with a smooth playback, most browsers use a buffer to hold video data as it downloads. When the buffer area is filled enough that the browser feels it can play the video at the designated frame rate and keep up with the download, it plays the video.

However, if the download slows down and the buffer starts to catch up with the real-time data, video playback will appear choppy and reduce the viewing experience of your site visitors. This is why a high frame rate doesn't necessarily equate to a better video quality.

Looking at digital video formats

Just as with audio files, many different companies have devised different methods of compressing videos for storage and playback. Unfortunately, this has resulted in a hodge-podge of different video formats that we have to work with. Table 5-5 lists the more popular video formats you'll most likely run into.

Just as with the audio files, each video file format has its own pros and cons, making it difficult to decide which one to use. Although the WebM video standard was developed by a consortium of browser developers, it's actually one of the lesser-used standards.

TABLE 5-5 Common Video Formats

Format	File	MIME	Description
AVI	`.avi`	video/x-msvideo	Audio Video Interleave. Developed by Microsoft.
Flash	`.flv`	video/x-flv	Adobe Flash video.
MPEG	`.mpg`	video/mpg	The original Motion Pictures Expert Group standard for digital video.
MPEG-4	`.mp4`	video/mp4	Updated MPEG standard, currently in use.
Ogg Theora	`.ogg`	video/ogg	Open-source video standard.
QuickTime	`.mov`	video/quicktime	Developed by Apple.
RealVideo	`.rm`	video/x-pn-realvideo	Developed by Real Media for video streaming.
WebM	`.webm`	video/webm	Developed by browser developers as a common video format.
WMV	`.wmv`	video/x-ms-wmv	Microsoft standard video format.

Putting videos in your web page

With the original version of HTML there was no standard way of embedding video content in your web pages. Proprietary methods became popular, and the Adobe Flash plugin became the de facto standard in web video.

However, HTML5 has changed that, by including a way to embed videos into your web pages without requiring the use of a separate plugin. The new *video element* is what does that.

As you can probably guess, the video element works the same way as the audio element does. It allows you to provide a list of source elements that define different videos using different MIME types. The basic format for that looks like this:

```
<video>
<source src="mymovie.mp4" type="video/mp4">
<source src="mymovie.ogg" type="video/ogg">
Sorry, your browser is unable to play the video
</video>
```

The browser attempts to play the first listed video, and if that fails, it tries the second listed video. Then if that fails, the browser will display the text specified.

WARNING

The HTML5 standard only defines support for the MP4, Ogg, and WebM video formats. If you need to play another video format you can try to use the embed element.

The `<video>` tag supports some attributes that allow you to control the viewing experience for your site visitors. Table 5-6 shows what attributes are available.

TABLE 5-6 The <video> Tag Attributes

Attribute	Description
`autoplay`	Starts the video as soon as the web page loads
`controls`	Displays a set of icons for controlling the video (such as Play, Stop, and Pause)
`height`	Sets the height of the video display area in the web page
`loop`	Specifies that the browser should continually loop through the video
`muted`	Starts the video with muted audio
`poster`	Specifies an image URL to show while the video is downloading
`preload`	Loads the video when the web page loads instead of when the Play button is clicked
`src`	Specifies the location of the video file
`width`	Specifies the width of the video display area in the web page

For videos, it's imperative that you specify the `height` and `width` attributes to maintain control of how the video appears in the web page. The browser will limit the video to display within the area you specify. Figure 5-3 shows playing a video with controls in the Internet Explorer web browser.

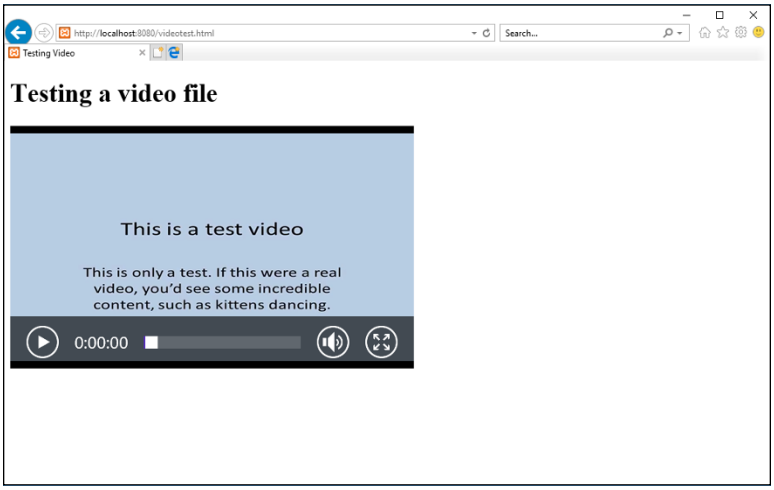

FIGURE 5-3: Playing a video in the Internet Explorer browser.

The video viewer in Internet Explorer provides a Play/Pause button, a status indicator showing the current position in the video file, a Mute button that also displays a volume control when you click it, and a button that allows you to switch to viewing the video in full-screen mode.

WARNING

The same warning that I gave you about automatically playing audio files applies to video files. Never assume that your site visitors will want to view the video as soon as the web page loads, even if you include the `muted` attribute. Playing videos takes a lot of processor power from the workstation; for site visitors using less powerful devices, that may cause issues.

Getting Help from Streamers

Trying to provide your own videos in the correct format to display correctly in all browsers can be somewhat of a challenge. Sometimes it's best to cry "uncle," and let the professionals handle it. By "professionals," I mean the myriad of commercially available video-streaming services, such as YouTube, Vimeo, and LiveStream. Most of these services allow you to register for free trials, and some even allow you to host small videos for free in your own channel.

The beauty of using a streaming service is that usually you only need to upload your video in one format; then the streaming service takes care of reformatting the video to match other video formats or quality required by your website visitors. No more having to reformat videos yourself and worrying about how they'll appear in different browsers, at different bandwidth speeds.

To embed a video from a streaming service requires that you use the old HTML *iframe element.* The iframe element was popular in the early days of HTML as a way of dividing a web page into separate sections. However, the iframe method was clunky, and soon CSS provided a much better way of dividing web pages.

However, the iframe element has had something of a comeback as a container for displaying streaming videos. The format uses the two-sided `<iframe>` tag:

```
<iframe width=x height=y src="location">
</iframe>
```

As you would expect, the `width` and `height` attributes are necessary to control the size of the iframe area in your web page. The `src` attribute points to the custom URL your streaming provider assigns to your uploaded video.

3 JavaScript

Contents at a Glance

IN THIS CHAPTER

- Defining JavaScript
- Adding JavaScript to your web pages
- Working with data
- Looking at JavaScript control structures
- Creating JavaScript functions

Chapter 1
Introducing JavaScript

The previous minibook shows you how to use HTML5 and CSS3 to create some pretty fancy-looking web pages. That's the first step to creating your dynamic web applications, but there are a few more parts to add. This minibook tackles the next piece you'll need to add to your web programs: client-side programming.

This chapter focuses on the JavaScript programming language, the most popular client-side programming language in use today. First, I cover the basics of how to add JavaScript code to your web pages. Then I explore some of the basics of the JavaScript language.

Knowing Why You Should Use JavaScript

HTML5 and CSS3 work together to create web pages. The HTML5 code produces the content that appears on the web page, and the CSS3 code helps style it to change the format and location of the web page elements. So, what exactly does JavaScript do to help augment those languages?

JavaScript is program code that you embed into the HTML5 code. The web server sends the JavaScript program code to your site visitors' web browsers, which in turn detect and run the JavaScript code. The JavaScript code can alter features of the web page that the HTML5 and CSS3 code produce. This section explains what you can do with JavaScript code.

Changing web page content

In your HTML5 code, you no doubt will have lots of text that appears in separate sections of your web page. For example, you may have a sidebar element that lists the day's news events related to your website topic, or you may have a header element that displays the current time and temperature for your city.

All that is great, but you need a way for that information to change dynamically, each time your site visitors load the web page. This is where JavaScript comes in.

JavaScript code allows you to alter the text that appears on your web page "on the fly," without requiring your site visitors to reload the web page. You can create JavaScript code that retrieves updated news articles even as your site visitors are viewing your web pages. The information will change right before their eyes — like magic!

Changing web page styles

Book 2, Chapter 2, explains how you add CSS3 styles to your web pages to apply styles to text and elements, or to place elements in specific locations on the web page. The CSS3 code you create is placed inline in the HTML5 elements, internally in the head element of the web page, or as an external style sheet.

JavaScript code allows you to dynamically alter any style or position that you define for an HTML5 element in your CSS3 code. That's right — you can use JavaScript to turn blue backgrounds yellow, orange text green, or even move an entire section of text from one side of the web page to another! That's a lot of control to have at your fingertips.

One of the coolest features of JavaScript is the ability to dynamically make HTML5 elements appear out of nowhere! Each HTML5 element supports the `display` style property, which you use to determine how or if the element appears on the web page.

With JavaScript code you can dynamically alter the `display` style property for any element on the web page to make it appear as needed or disappear when not needed. That gives you the ability to dramatically alter the layout of a web page at any time while your site visitor is interacting with the web page. This helps hide sections that may be distracting to site visitors at times, then make them appear when the site visitor needs to interact with them.

Seeing Where to Put Your JavaScript Code

Now that I've sold you on the benefits of using JavaScript code in your web pages, let's take a look at how you include JavaScript code in your HTML5 code. There are two ways of including JavaScript code in web pages:

- Embedding the JavaScript code directly into the HTML5 code for the web page
- Creating an external JavaScript file that the browser downloads and runs

This section walks through how to use both methods of working with JavaScript code in your HTML5 code.

Embedding JavaScript

You embed JavaScript code directly into the HTML5 code for your web pages by using the *script element.* The script element is a two-sided element, so it has an opening and closing tag that surrounds your JavaScript code:

```
<script>
    JavaScript code
</script>
```

The script element informs the browser that there's code to run as part of the web page. Most browsers will recognize the JavaScript code that appears in the script element and run the code using an internal JavaScript interpreter. However, some programmers like to identify the type of code embedded in the script element using the `type` attribute:

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

This isn't required in HTML5, but you're more than welcome to use this format if it helps you to remember that the embedded code is JavaScript. This can be especially helpful when you start embedding server-side programming languages, such as PHP, in your HTML5 code as well.


TIP

You can place script elements anywhere in the HTML5 code. The browser will process the JavaScript code as it parses the HTML5 code for the web page. However, that affects how the JavaScript code runs and how any output generated by the JavaScript code appears. The following sections demonstrate this.

Embedding in the head element

If you place the script element inside the head element of the web page, the browser will run the JavaScript code before it processes the code to build the web page. Follow these steps to see how this works:

1. **Open your favorite text editor, program editor, or integrated development environment (IDE) package.**
2. **Enter the following code:**

```
<!DOCTYPE html>
<html>
<head>
<title>Testing JavaScript in the Head Section</title>
<script>
alert("This is the JavaScript program!");
</script>
</head>
<body>
<h1>This is the web page</h1>
</body>
<html>
```

3. **Save the code as `scriptheadtest.html` in the `DocumentRoot` folder for your web server.**

   That's `c:\xampp\htdocs` for XAMPP on Windows or `/Applications/XAMPP/htdocs` for XAMPP on macOS.

4. **Start the XAMPP Control Panel and start the Apache web server.**
5. **Open your browser and enter the following URL:**

```
http://localhost:8080/scriptheadtest.html
```

   You may have to alter the TCP port to match your web server setup.

6. **Stop the Apache web server and exit from the XAMPP Control Panel.**

The `scriptheadtest.html` code embeds a script element inside the head element of the web page. Because this appears before the body section of the web page code, the browser processes the JavaScript code before the body element section. The script element contains a single line of JavaScript code:

```
alert("This is the JavaScript program!");
```

The `alert()` function displays text in a pop-up dialog box, separate from the main web page window of the browser.

When you run the program, you should see the alert dialog box pop-up, but no text appears on the web page, as shown in Figure 1-1. That's because the `alert()` function stops the browser from processing any more code until the site visitor clicks the OK button in the dialog box. The code is frozen in time, waiting for the browser to continue processing the rest of the code.

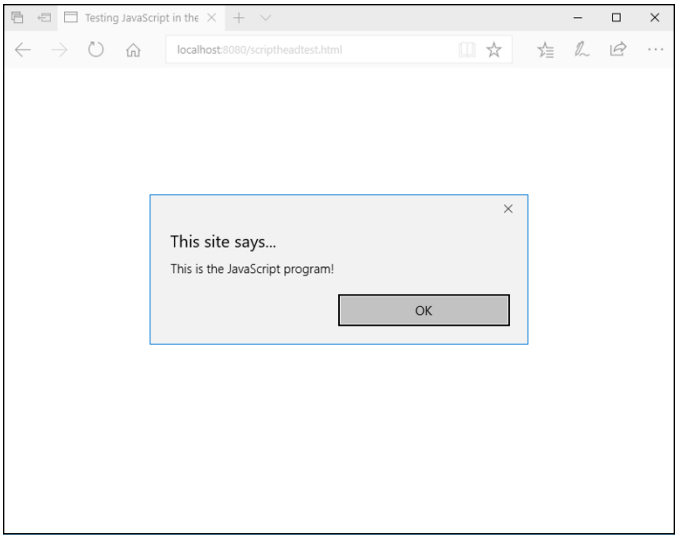

FIGURE 1-1: Testing the `script headtest.html` program file.

WARNING

When you run the test, your browser may not run the JavaScript code or it may prompt you to allow the code to run. Some browsers have built-in security features to block running JavaScript code embedded in a web page. You'll need to consult your browser documentation on how to enable JavaScript code, at least from the localhost address, so your test programs can run.

Embedding in the body element

Alternatively, you can place the script element inside the body element section of the web page. When you do this, the browser runs the JavaScript code when it gets to the script element as it parses the HTML5 code to build the web page.

Follow these steps to test this out:

1. **Open your favorite text editor, program editor, or IDE package.**
2. **Enter the following code:**

```
<!DOCTYPE html>
<html>
<head>
<title>Testing JavaScript in the Body Section</title>
</head>
<body>
<h1>This is the web page</h1>
<script>
alert("This is the JavaScript program!");
</script>
<h1>This is the end of the web page</h1>
</body>
<html>
```

3. **Save the code as `scriptbodytest.html` in the `DocumentRoot` folder for your web server.**

   That's `c:\xampp\htdocs` for XAMPP on Windows or `/Applications/XAMPP/htdocs` for XAMPP on macOS.

4. **Start the XAMPP Control Panel and start the Apache web server.**
5. **Open your browser and enter the following URL:**

```
http://localhost:8080/scriptbodytest.html
```

   You may have to alter the TCP port to match your web server setup.

6. **Stop the Apache web server and exit from the XAMPP Control Panel.**

When you run the test, you should see the content from the first h1 element appear on the web page, and the `alert()` function dialog box, but not the content from the second h1 element. Figure 1-2 shows that result.

The browser processes the first h1 element in the body section and then stops to run the JavaScript `alert()` function. After you click the OK button in the alert dialog box, the browser displays the section h1 element content.


TIP

Embedding JavaScript code inside the body element of a web page can slow down how the web page loads. If the code location is not crucial, it's best to place the script element at the end of the body element, after the normal HTML5 code.

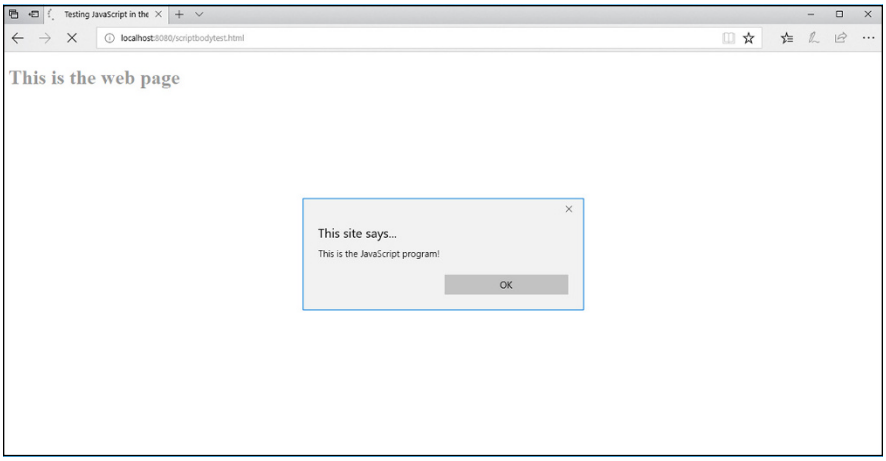

FIGURE 1-2: Running JavaScript code in the body section.

Using external JavaScript files

If you have JavaScript code that you need to embed in all of your web pages, having to retype the same code in each web page file can become tedious. And on top of that, if you need to change anything in the code, you have to revisit every single web page file that uses the code!

To solve that problem, you can use an external JavaScript file. The `<script>` opening tag supports the `src` attribute, which allows you to define an external location for the JavaScript code:

```
<script src="myjavascript.js"></script>
```

The `src` attribute can point to an absolute or relative file path on the local server, or you can use a full URL to point to a JavaScript file stored on a remote server. Note that although it's not mandatory, it's very common to use the `.js` file extension to identify JavaScript files.

WARNING

You place the JavaScript code inside the external file just as it would appear within the script element. Be careful though — don't include the `<script>` and `</script>` tags in the external JavaScript file.

The Basics of JavaScript

Now that you've seen where to put your JavaScript code in your web pages, you can dive into coding using JavaScript. This section goes through the basics for getting started with JavaScript coding.

Working with data

Data is the key to any program, and JavaScript programs are no exception. You'll need to work with different types of data in your dynamic web applications, everything from bowling scores to employee records. Being able to manipulate that data is an important function.

To manipulate data, the JavaScript program needs a way to temporarily store it somewhere so that it can retrieve the data later on, manipulate it, and then display it to the site visitor running the program. JavaScript does that using variables. *Variables* are names that represent storage locations in the computer memory where your JavaScript program can temporarily store data values. When your JavaScript code places a value into a variable, the JavaScript interpreter converts that action into the physical action of storing the data into the computer memory for future use.


WARNING

The downside to using variables to store data is that they only retain their values for the duration of the program. You can't save a data value to a variable in one web page and then retrieve it in another web page. After your site visitor leaves the web page, those values (and their data values) are gone forever. That's why we need some help from our MySQL database server — to have a place for storing data long term!

For all the JavaScript variables that you use in your programs, you must assign each one a unique name to represent the different memory locations. There are a few rules you'll need to remember when creating JavaScript variable names:

- Variable names can contain letters, numbers, underscores, and dollar signs.
- Variable names must begin with a letter, an underscore, or a dollar sign.
- Variable names are case sensitive.
- You can't use JavaScript keywords as variable names.

Before you can use a variable in your JavaScript code, you must first declare it as a variable using the `var` statement:

```
var test;
```

This format tells the JavaScript interpreter to set aside a place in memory for storing data and call that location `test`. For now, you haven't assigned a specific value to the `test` variable, so it contains what's called an *undefined value.*

You can then use the JavaScript *assignment operator* to assign a value to the variable:

```
test = 10;
```

If you prefer, you can both declare a variable and assign it a value in one statement:

```
var test = 10;
```

When you need to reference the value you stored in the variable later on in the program code, you just refer to it using the variable name:

```
alert(test);
```

The JavaScript interpreter retrieves the value stored in the location the variable represents and uses it just as if you had entered the value in the statement.

Data types

JavaScript variables can hold different types of data. The two basic data types are

- **Numbers:** Either integer values (such as 5) or floating point values (such as 3.1419)
- **Strings:** A series of characters, strung together in memory one after the other (thus the term *strings*)

Declaring a number value is somewhat straightforward:

```
var age = 20;
```

This statement places the numeric value of 20 into the memory location pointed to by the `age` variable.

Declaring a string value is a little bit trickier:

```
var name = "Rich Blum";
```

You must enclose the string value in quotes. That delineates the start and end of the string value. If you forget the quotes, you'll get a JavaScript error message.

One interesting feature of JavaScript is that it uses *dynamic data typing.* With dynamic data typing, you don't need to tell JavaScript what type of data a variable contains ahead of time, like some other programming languages require. Instead, JavaScript will automatically try to figure out the type of data from the values you use.

With dynamic data typing, you can also use the same variable name to hold different data types at different times. For example, after declaring the `age` variable with a number, later in the program you could then do the following:

```
age = "really old";
```

JavaScript won't complain that you started out storing a number in the `age` variable and then shifted to storing a string value, it just happily changes the value stored in that variable.


WARNING

Although dynamic data types can come in handy, they can also cause problems if you're not careful. If you try to perform a mathematical operation on a variable that contains a string value, you won't get what you might have been expecting. Always keep track of what type of data you're storing in variables in your programs.

Arrays of data

JavaScript allows you to store multiple values in a single variable. These variables are called *arrays.*

If you have an application that contains a list of items, it can often be somewhat clunky to specify each item as a separate variable:

```
var score1 = 100;
var score2 = 110;
var score3 = 105;
```

If you need to perform any type of operation on the variables, you need to know exactly how many variables are used to contain the list of items.

Arrays allow us to store an entire list of items into a single variable:

```
var scores = [100, 110, 105];
```

The scores array variable contains three items (called *elements*). You reference an individual element value by using an *index* value. You specify the index using brackets with the array variable:

```
scores[0]
```

This array variable contains the first element of the array (the value 100 in this example). Note that the first element of the array is at index 0, a very unfortunate fact that's important to remember when working with arrays in JavaScript!

You can change an individual array element value using the index in a standard assign statement:

```
scores[1] = 120;
```

This replaces the 110 value in the array with a value of 120.

JavaScript treats arrays as *objects.* An object has properties, as well as methods that you can run against the object. Properties return features for the array, such as the `length` property:

```
var games = scores.length
```

Methods are used to manipulate the values within the object:

```
scores.sort();
```

You can add new values to an existing array by using the `push()` method:

```
scores.push(115);
```

This allows you to dynamically store and retrieve data values from a single variable location in your program, without having to know exactly how many data values it will need to retain.

Operators

JavaScript provides different *operators* for working with data. An operator performs some type of manipulation of the data provided. Table 1-1 shows the basic math operators that JavaScript uses.

TABLE 1-1 JavaScript Math Operators

Operator	Description
+	Addition
–	Subtraction
*	Multiplication
/	Division
%	Modulus (the remainder of a division operation)
++	Increment (increases the value by 1)
––	Decrement (decreases the value by 1)

You use the math operators as part of an assignment statement:

```
result = 10 + 5;
```

The JavaScript interpreter performs the operation on the right side of the equal sign and then assigns the result to the variable declared on the left side. You can use variables with the operators as well:

```
var side1 = 10;
var side2 = 5;
var area = side1 * side2;
```

Again, JavaScript performs the operation on the right side of the assignment operator first and then assigns the result to the variable declared on the left side. In this example, JavaScript retrieves the value stored in both the `side1` and `side2` variables, performs the multiplication, and then stores the result in the `area` variable.

Don't think of the assignment as a mathematical equation. You can have an assignment statement that looks like this:

```
counter = counter + 1;
```

As a mathematical equation, this is impossible — you can't have a value equal to itself plus 1. What's happening here is that the interpreter adds 1 to the value stored in the `counter` variable and then stores the result back into the `counter` variable memory location.

TIP

JavaScript also provides the incrementor operator, `++`. This adds 1 to the variable without all the extra text: `counter++;`.

Besides the math operators, JavaScript also supports logical Boolean operators, as shown in Table 1-2.

TABLE 1-2 JavaScript Boolean Operators

Operator	Description
&&	logical AND
||	logical OR
!	logical NOT

Boolean operators are most commonly used in condition tests in control statements, as described a little later in the "Controlling Program Flow" section of this chapter.

There is also a string operator that you can use in JavaScript. Although it may seem odd, JavaScript supports the plus sign when working with string values:

```
var value1 = "test";
var value2 = "ing";
var value3 = value1 + value2;
```

The resulting `value3` variable will contain the string value *testing.* The plus sign concatenates the two separate string values into a single string value. This comes in handy when you want to display a string value stored in a variable along with some text, like this:

```
var display = "Welcome, " + name + " to the game!";
```

Notice the spaces at the end of the first string value, and at the beginning of the second string value. These are necessary because the concatenation doesn't add any spaces itself when it joins the variable value to the other strings.

Controlling Program Flow

By default, JavaScript processes statements in a linear fashion, operating on one statement, and then moving on to the next statement in the program code. You may want to alter the behavior of the code based on some type of conditions, events, or variable values.

You can do that using *flow control statements.* Flow control statements alter the flow of the program to make the JavaScript interpreter jump over code to another statement, based on some type of condition. The following sections discuss two popular methods of flow control in JavaScript programming.

Conditional statements

Life is full of conditions. How often do you get up in the morning and say "If it's raining today, I'd better bring my umbrella"? Your actions for the day depend on a specific condition (the weather). JavaScript programs provide the same type of condition checks for your code. These are called *conditional statements.* They process blocks of code depending on a condition that the program can test for.

There are a few different types of conditional statements:

- `if` statements
- `else` statements
- `switch` statements

Each has its own set of nuances and formats that you'll need to become familiar with. This section walks through each type of conditional statement.

`if` statements

The `if...else` statement checks a condition that you specify and runs specific code if that condition occurs or skips the code if the condition doesn't occur. Here's the basic format for an `if` statement:

```
if (condition) {
     code to process
}
```

The JavaScript interpreter evaluates the condition between the parentheses. If the condition evaluates to a `true` value, the interpreter runs the code inside the braces. If the condition evaluates to a `false` value, the interpreter skips all the code between the braces. Let's take a look at an example of this:

```
if (age > 17) {
    alert("You are allowed to play the game");
}
```

The condition in this `if` statement evaluates the value currently stored in the `age` variable. If the value is greater than 17, the interpreter runs the `alert()` function. If the value is not greater than 17, the interpreter skips the `alert()` function.


TIP

The greater-than symbol used in the condition is another type of JavaScript operator, called a *comparison operator.* Comparison operators compare two values to test their equality. Table 1-3 shows the JavaScript comparison operators to use in conditions.

The equal-to operator (==) is possibly the most forgotten operator in JavaScript, even for pros. If you want to check if a variable is equal to a specific value, you must use the equal-to operator:

```
if (counter == 20) {
```

TABLE 1-3 The JavaScript Comparison Operators

Operator	Description
==	Equal to
===	Equal to and the same data type
!=	Not equal to
!==	Not equal to the value or the type
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
?	Ternary operator

The equal-to operator compares the two values, and the interpreter processes the code in the code block only if they're equal. In coding, programmers often get in a hurry and write the following:

```
if (counter = 20) {
```

This uses the assignment operator (=) instead of the comparison operator (==). The assignment operator assigns the value of 20 to the counter variable and then returns a true value if the assignment was successful (which it usually is). This is *not* the same thing as the comparison operator, and it'll produce faulty results in your programs!

Most of the comparison operators are fairly self-explanatory. The ternary operator is somewhat different. It provides a shortcut way of combining an `if...else` statement and an assignment statement:

```
var display = (age > 21) ? "Too old":"Young enough";
```

The ternary operator performs the condition on the left side of the question mark. If the condition evaluates to a true value, it assigns the first value to the variable. If the condition evaluates to a false value, it assigns the second value to the variable.

`else` statements

With the `if` statement, if the condition is not met, the interpreter just skips the code you specify in the code block. The `else` statement allows you to specify code to run if the condition evaluates to a false value. That looks like this:

```
if (age < 18) {
    display = "Sorry, you are not old enough to play";";
    status = "failed";
} else {
    display = "You may begin the game";
    status = "ok";
}
```

Now there are two separate code blocks — one associated with the `if` statement and another associated with the `else` statement. The two are linked. If the condition evaluates to a `true` value, the JavaScript interpreter runs the code in the `if` statement code block. If the condition evaluates to a `false` value, the JavaScript interpreter runs the code contained in the `else` statement code block. The interpreter runs the code in one block or the other. At no time will the interpreter run the code in both code blocks.

More often than not, you'll find yourself needing to test a variable for a range of values. Instead of having to write multiple `if...else` statements, JavaScript allows you to string them together into one long statement by using the `else if` statement. The `else if` statement strings together multiple `if` and `else` statements so that they become one long chain of condition tests:

```
if (age < 10) {
    display = "You are very young";
} else if (age < 20) {
    display = "You are between 10 and 19";
} else if (age < 30) {
    display = "You are between 20 and 29";
} else {
    display = "You are 30 or older";
}
```

When stringing together multiple `if` and `else` statements, the interpreter goes through the list in order from the first condition test to the last condition test. When the first condition evaluates to a true value, the interpreter runs the code in the code block and exits the statements.

`switch` statements

Using the `else if` statement provides an easy way to check for a value within a range, but it's still somewhat clunky to code. JavaScript makes that test easier for us by providing the `switch` statement.

The `switch` statement performs a similar function to the `else if` statement, but using a cleaner format:

```
switch (expression) {
    case match1:
        statements
        break;
    case match2:
        statements
        break;
    default:
        statements
}
```

With the `switch` statement, the JavaScript interpreter evaluates the *expression* specified and then compares the result with one or more `case` statements. Each `case` statement specifies a different possible result of the expression. If the result matches, the interpreter runs the statements contained in that section. The `break` statement is used to force the interpreter to then skip over the remaining `case` statement sections to the end of the switch code block. If none of the case results matches, the interpreter runs the statements under the `default` statement.

This sounds complicated, but it actually makes your life much easier when coding to check for variable values. Here's an example of using a switch statement:

```
switch (counter) {
    case 0:
        alert("You have four lives left");
        break;
    case 1:
        alert("You have three lives left");
        break;
    case 2:
        alert("You have two lives left");
        break;
    case 3:
        alert("Careful, you only have one life left");
        break;
    case 4:
        alert("Sorry, you are out of lives");
}
```

The expression to evaluate is the value stored in the `counter` variable. Depending on the value, the program displays a different alert message and then breaks out of the `switch` statement code block.

Checking for a range of values is a little tricky, but doable with the switch statement. Follow these steps to experiment with using the `switch` statement in a JavaScript program:

1. **Open your favorite text editor, program editor, or IDE package.**
2. **Enter the following code:**

```
<!DOCTYPE html>
<html>
<head>
<title>Testing the switch statement</title>
<script>
var age = prompt("How old are you?");
</script>
</head>
<body>
<script>
switch (true) {
   case (age < 18):
       alert("Sorry, you are too young to play");
       break;
   case (age < 50):
       alert("Welcome to the game!");
        break;
    case (age >= 50):
        alert("Sorry, you are too old to play");
}
</script>
</body>
</html>
```

3. **Save the code as `switchtest.html` in the `DocumentRoot` folder of your web server.**

   That's `c:\xampp\htdocs` for XAMPP on Windows or `/Applications/XAMPP/htdocs` for XAMPP on Mac macOS.

4. **Open the XAMPP Control Panel and start the Apache web server.**

5. **Open your browser, and enter the following URL:**

```
http://localhost:8080/switchtest.html
```

You may need to change the TCP port based on your web server.

6. **Stop the Apache web server and close the XAMPP Control Panel.**

The `switchtest.html` code has two separate switch elements. The first one is in the head element of the web page. It uses the JavaScript `prompt()` function to prompt the site visitor for an age:

```
var age = prompt("How old are you?");
```

It stores the value in the `age` variable. Figure 1-3 shows what the prompt looks like using the Microsoft Edge browser.

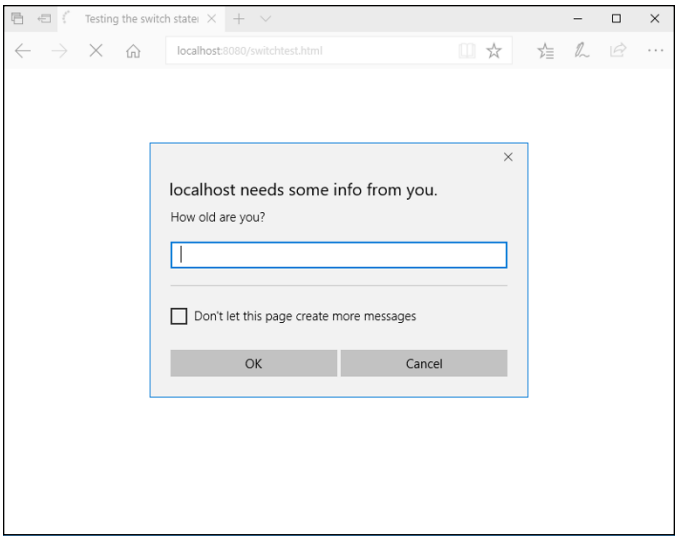

FIGURE 1-3: The JavaScript `prompt()` function as displayed by the Microsoft Edge browser.

Notice that the Edge browser provides some additional information in the prompt dialog box, such as the host that has produced the prompt. This can be helpful to prevent security issues with unwanted pop-up prompts from dangerous websites.

The second script element retrieves the `age` variable and uses it in a `switch` statement. The odd thing is that the switch statement just has a true value for the expression. This means the expression will always evaluate to a true condition, so the individual `case` statements test the condition.

The interpreter will run the first `case` statement that matches the age range. Figure 1-4 shows the `alert()` message that displays for entering an age of 15.

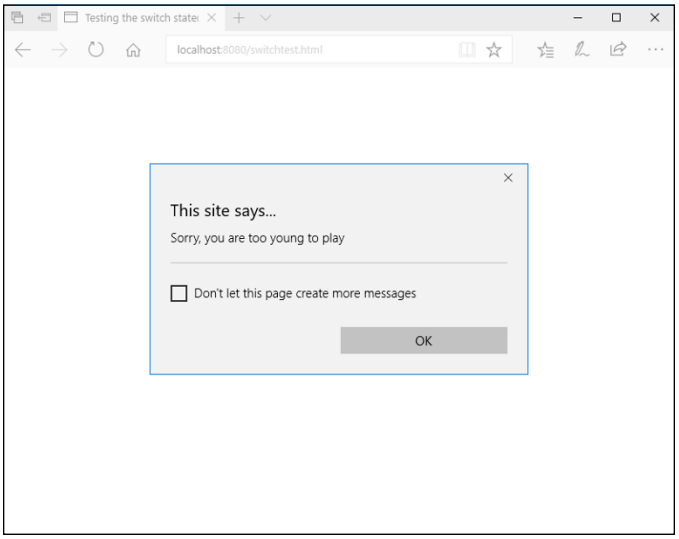

FIGURE 1-4: The JavaScript `alert()` function response as displayed by the Microsoft Edge browser.

TIP

Notice, in this example, that the code sets the `age` variable in one script element and uses that value in another script element. JavaScript maintains variables and their values between multiple script elements contained in the same web page.

Loops

Often, when you're writing programs, you'll run into situations where you need to run the same block of code multiple times, called a *loop*. Usually, in a loop, one or more variables changes values in each iteration of the loop. The loop exits when the variable reaches a specific value.

WARNING

Loops that contain variables that never change values are called *endless loops*. If your program gets stuck in an endless loop, the browser will never show the web page as loading completely.

JavaScript supports a few different types of loop statements, as shown in Table 1-4.

Each of these loop types is useful and comes in handy in different environments. The following sections walk through how to use each type of JavaScript loop statement.

TABLE 1-4 JavaScript Looping Statements

Statement	Description
`do..,while`	Executes a block of statements and checks a condition at the end
`for`	Checks a condition, executes a block of statements, and then alters a specified variable
`for...in`	Executes a block of statements for each element contained in an array
`while`	Checks a condition and then executes a block of statements

The `do...while` loop

The `do...while` loop executes a block of statements and then at the end of the block tests a condition to determine if the block should be repeated:

```
var side1 = 1;
var side2 = 5;
do {
   area = side1 * side2;
   alert(side1 + "x" + side2 + " = " + area);
side1 = side1 + 1;
} while (side1 <= 10)
```

The `do...while` loop ensures that the code in the loop executes at least once before the condition is evaluated.

The `while` loop

The `while` loop is the opposite of the `do...while` loop:

```
var side1 = 1;
var side2 = 5;
while (side1 <= 10) {
area = side1 * side2;
   alert(side1 + "x" + side2 + " = " + area);
side1 = side1 + 1;
}
```

Because the condition is checked before the interpreter executes the code in the code block, it's possible that the condition will fail before the first loop and none of the code will be executed.

The `for` statement

The `for` statement is similar to the `while` loop, but it provides three features in one statement:

- It sets the initial values of one or more variables going into the loop.
- It defines the condition to evaluate before each iteration.
- It defines how a variable should be changed at the end of each iteration.

The basic format of the `for` statement is:

```
for(statement1; condition; statement2) {
    statements
}
```

The *`statement1`* statement is executed before the loop starts. The interpreter then evaluates the *`condition`* to determine whether to execute the statements in the code block. At the end of executing the statements in the code block, the interpreter executes the *`statement2`* statement.

This provides a compact way of creating loops for your programs. Try the following steps to test using the `for` statement to calculate the factorial of a value:

1. **Open your favorite text editor, program editor, or IDE package.**
2. **Enter the following code:**

```
<!DOCTYPE html>
<html>
<head>
<title>Calculating the Factorial</title>
<script>
var number = prompt("Please enter a number:");
</script>
</head>
<body>
<script>
var factorial = 1;
for (counter = 1; counter <= number; counter++) {
    factorial = factorial * counter;
}
```

```
var output = "The factorial of " + number + " is " + factorial;
alert(output);
</script>
</body>
</html>
```

3. **Save the file as `factorial.html` in the `DocumentRoot` folder for your web server.**
4. **Start the XAMPP Control panel and start the Apache web server.**
5. **Open your browser and enter the following URL:**

```
http://localhost:8080/factorial.html
```

You may need to change the TCP port to match your web server setup.

6. **Stop the Apache web server and close the XAMPP Control Panel.**

The `factorial.html` code uses the `prompt()` function to prompt for a value at the start of the program. This is shown in Figure 1-5.

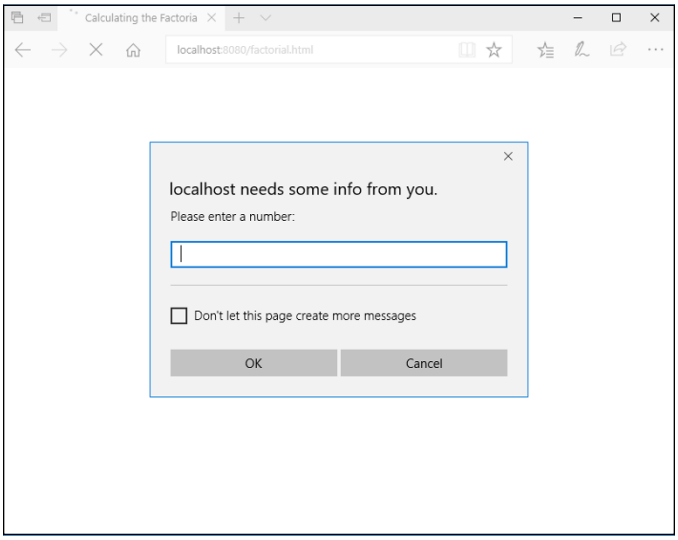

FIGURE 1-5: Prompting for the factorial number.

You find the factorial of a number by multiplying the series of numbers up to and including the number you want. So the factorial of 5 is $1 \times 2 \times 3 \times 4 \times 5$, which is 120. To calculate the factorial, you set up a `for` statement to iterate through the numbers starting at 1, up to the number entered into the prompt dialog box:

```
for(counter = 1; counter <= number, counter++)
```

The counter variable keeps track of how many iterations of the for loop have taken place. After each iteration, you add 1 to the counter variable by using the incrementor operator (++). At the start of each new iteration, the interpreter checks the condition statement, which evaluates whether the counter variable value is less than or equal to the number value. If this is true, the interpreter continues with the next loop iteration.

The for...in statement

The for...in statement comes in handy when you need to iterate through the data elements stored in an array variable. Often, you don't know how many values are stored in the array, so you can't just use a for loop to loop a specific number of times.

The for...in statement allows you to extract individual data values from the array and then stop when the array runs out of data elements. That code looks like this:

```
var scores = [100, 110, 105];
for (index in scores) {
    output = "One bowling score was " + scores[index];
    alert(output);
}
```

The first time the for...in statement runs, the index variable contains the value of 0, or the first index number from the array. For the next iteration, the index variable contains the value of 1, and for the final iteration, it contains the value 2. You can then use that index value to reference the individual data values stored in the array.

Working with Functions

As you write more complex JavaScript code, you'll find yourself reusing parts of code that perform specific tasks. Sometimes it's something simple, such as displaying a prompt and retrieving a response from the site visitor. Other times it's a complicated calculation that's used multiple times in your program.

In each of these situations, writing the same blocks of code over and over again can get tiresome. It would be nice to just write the block of code once, and then be able to refer to that block of code in the other places it's needed.

JavaScript provides a feature that lets you do just that. Functions are blocks of code to which you assign names; then you can reuse them anywhere in your code. Any

time you need to use the block of code in your program, you simply use the name you assigned to the function. This is referred to as *calling the function.* The following sections describe how to create and use functions in your JavaScript code.

Creating a function

To create a function in JavaScript you use the `function` statement:

```
function name(parameter1, parameter2, ...) {
    function code
    return value;
}
```

The *name* that you assign to the function must be unique within your program code. The function can take parameters that the calling program passes to it. You can use the parameter variables within the function code.

WARNING

Functions are intended to be self-contained. The only data they work with are the values passed as the function parameters. This allows you to use the function in any program that requires that function task. Because of that, you can't directly access variables defined in the main program from inside the function code.

At the end of the function, you can opt to have it return a single value back to the calling program by including the `return` statement. If no data needs to be returned back to the calling program, you can leave out the `return` statement.

Here's an example of writing a function to calculate the factorial value of a number passed to the function:

```
function factorial(number) {
    var factorial = 1;
    for(counter = 1; counter <= number; counter++) {
        fact = fact * counter;
    }
    return fact;
}
```

The `factorial()` function requires a single parameter, assigned to the `number` variable. Inside the `factorial()` function the code uses the `number` variable to calculate the factorial. When the `for` loop completes, the answer is stored in the `fact` variable. The `return` statement returns the value of the `fact` variable back to the calling program.

Using a function

To call a function from inside the JavaScript program, you just reference it by name, and include any parameters you need to pass. If the function returns a value, you can assign the output of the function to a variable:

```
var result = factorial(5);
```

The return value from the `factorial()` function is assigned to the `result` variable. You can use the `factorial()` function as many times as necessary in your program code.


TIP

Before you can use the function though, you must ensure that it gets defined. Because of this, it's common to define JavaScript functions at the start of the head element section of the web page.

IN THIS CHAPTER

- Getting acquainted with the Document Object Model
- Working with the Document Object Model
- Reading data from your web page
- Writing to your web page

Chapter 2
Advanced JavaScript Coding

In the previous chapter, I explain the basics of how to incorporate JavaScript code into a web page. If you read that chapter, you ran a couple of simple JavaScript programs, using the `prompt()` function for input and the `alert()` function for output. That was a great start, but the whole point of using JavaScript is to dynamically alter the data and/or appearance of web pages. This chapter explains how JavaScript interfaces with your web pages and shows you how to write JavaScript code to dynamically add, delete, or change content in your website.

Understanding the Document Object Model

In order for JavaScript to have access to the elements in your web page, it needs to know how to find them. The Document Object Model (DOM) provides a standard way of accessing objects placed within a web page. It creates a tree structure that contains every element, attribute, content text, and even CSS3 style contained within the web page. It treats each of these items as objects that the browser (or your program code) can manipulate. Finding any of these items is just a matter of walking through the tree with your JavaScript code.

The browser defines every web page as a set of DOM objects that the web page contains. Just as your family has a family tree that you can trace back to find relatives, every web page has its own DOM tree of the objects contained within the web page. With JavaScript, you can peruse through the DOM tree and make modifications along the way.

The Document Object Model tree

Every family tree has a head, and for the DOM tree, the head is the html element that starts out the web page. Just as parents have children, the `html` object in the DOM tree has two child objects: the `head` object and the `body` object, shown in Figure 2-1.

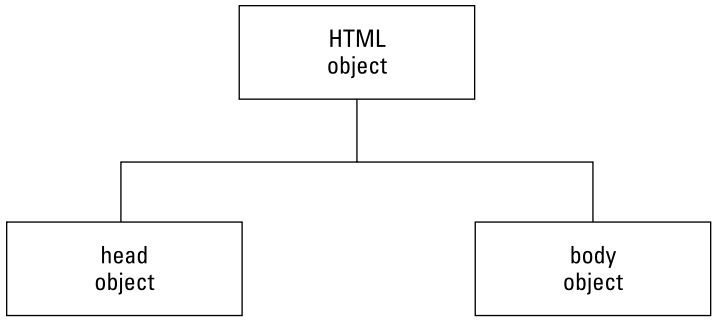

FIGURE 2-1: The `html` object and its two child objects.

The head and body elements in the HTML code are called *child objects* of the `html` object in the DOM tree. Because it comes first in the code, the `head` object is called the "first child object," while the `body` object is the "last child object." This terminology is important when working with DOM objects.

As you continue down the DOM tree, the browser places each object in the web page under its parent object. Let's take a look at a simple example of this principle. I'll use this sample web page for the demo:

```
<!DOCTYPE html>
<html>
<head>
<title>Sample DOM web page</title>
</head>
<body>
<h1>This is the heading of the web page</h1>
<p>This is sample text</p>
</body>
</html>
```

From this sample HTML5 code, the browser creates DOM objects from each element, and places them in a DOM tree layout it keeps in memory, as shown in Figure 2-2.

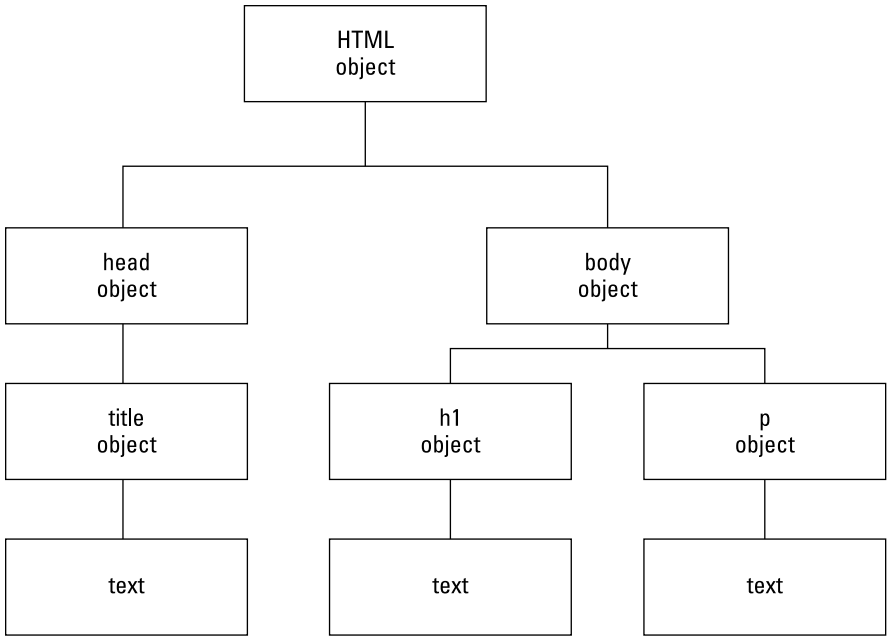

FIGURE 2-2: The simple DOM tree.

In the example shown in Figure 2-2, the `html` object contains the two child objects — `head` and `body` — but now each of those objects has child objects as well. The `head` object contains just one child object, the `title` object. The `title` object also has one child object, which may seem odd at first glance, because the `title` object doesn't contain any additional objects.

One of the more confusing features of the DOM tree is how it handles text inside elements. It treats the text inside an element as a separate DOM object that has its own features. So, in this example, the `title` object contains a single child object, which is the `text` object for the title text.

The `body` object has two child objects. The `h1` object is the first child of the `body` object, and it, too, contains a `text` child object. The `p` object is the last child of the `body` object, and it has a `text` object in it as well.

This simple example shows the basics of DOM. Working out the DOM tree for a large web page with lots of different types of elements can be complicated, but it uses the same principle. Fortunately, JavaScript has some features that help make things a little easier for you.

JavaScript and the Document Object Model

So far, you've seen that the browser breaks every web page down into a DOM tree of objects. The browser uses the DOM tree to keep track of all the HTML5 elements, their content, and the styles that appear on the web page. However, because JavaScript programs run in the browser (remember the whole client-side programming thing?), they have full access to the DOM tree created by the browser.

That means your JavaScript programs can interact directly with the DOM tree that the browser follows to create the web page. And not only that, but your JavaScript programs can add, change, and even remove objects in the DOM tree. As your JavaScript program modifies the DOM tree, the browser automatically updates the web page window with the new information. How cool is that? This is the key to client-side dynamic web programming.

Just like the DOM tree, JavaScript treats each element contained in a web page as an object. In JavaScript, objects have two features:

- **Properties:** Properties define information about the object.
- **Methods:** Methods are actions to take with the objects.

JavaScript assigns a special object named `document` to represent the entire web page DOM tree. You can reference many of the DOM objects directly from the `document` object, as well as add or remove objects. Table 2-1 lists some of the document properties available in JavaScript.

To reference a document property, you use the format *`document.property`*, like this:

```
var myurl = document.URL;
```

The same applies to using the document methods. Table 2-2 shows a list of the more popular document methods used in JavaScript.

TABLE 2-1 JavaScript Document Properties

Property	Description
`activeElement`	Returns the element that currently has the focus of the web page window
`anchors`	Returns a list of all the anchor elements on the web page
`body`	Sets or retrieves the body element of the web page
`cookie`	Returns all cookie names and values set in the web page
`characterSet`	Returns the character set defined for the web page
`documentElement`	Returns the DOM object for the html element of the web page
`documentMode`	Returns the mode used by the browser to display the web page
`domain`	Returns the domain name of the server used to send the document
`embeds`	Returns a list of all the embed elements in the web page
`forms`	Returns a list of all the form elements in the web page
`head`	Returns the head element for the web page
`images`	Returns a list of all the img elements in the web page
`lastModified`	Returns the time and date the web page was last modified
`links`	Returns a list of all the anchor and area elements in the web page
`title`	Sets or retrieves the title of the web page
`URL`	Returns the full URL for the web page

TABLE 2-2 JavaScript Document Methods

Method	Description
`createElement()`	Adds a new element object
`createTextNode()`	Adds a new text object
`getElementbyId(`*`id`*`)`	Returns an element object with the specified id value
`getElementsByClass Name(`*`class`*`)`	Returns a list of elements with the specified class name
`getElementsByTagname(`*`tag`*`)`	Returns a list of elements of the specified element type
`hasFocus()`	Returns a true value if the web page has the window focus
`write(`*`text`*`)`	Sends the specified text to the web page
`writeln(`*`text`*`)`	Sends the specified text to the web page, followed by a new line character

Let's run a quick test to see how this works. Follow these steps to test using the `write()` method for a web page document:

1. **Open your favorite text editor, program editor, or integrated development environment (IDE) package.**
2. **Enter the following code:**

```
<!DOCTYPE html>
<html>
<head>
<title>DOM Test</title>
<script>
document.write("<h1>This is a test of the DOM</h1>");
</script>
</head>
<body>
</body>
</html>
```

3. **Save the file as `domtest.html` in the `DocumentRoot` folder for your web server.**

   If you're using XAMPP in Windows, that's the `c:\xampp\htdocs` folder; for XAMPP in macOS, it's `/Applications/XAMPP/htdocs`.
4. **Open the XAMPP Control Panel and then start the Apache Web server.**
5. **Open your browser and enter the following URL:**

```
http://localhost:8080/domtest.html
```

   You may need to change the TCP port in the URL to match your Apache web server.
6. **Close the browser.**

When you examine the code in the `domtest.html` file, you'll notice that there's nothing in the body element, so you may not expect to see anything on the resulting web page. However, when you run the program, you should see the output shown in Figure 2-3.

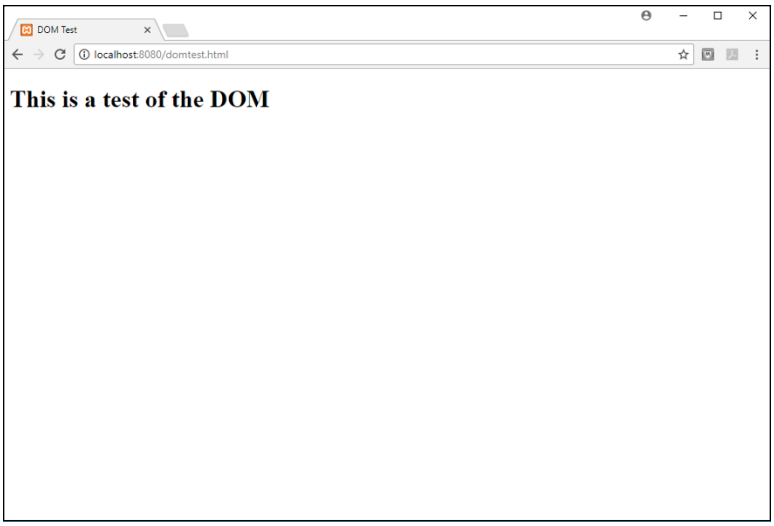

FIGURE 2-3: The output from the `domtest.html` program.

The `document.write()` function runs the `write()` method from the `document` object to dynamically place the h1 element in the web page for us!

WARNING

The `write()` method is an easy way to dynamically place text in the web page, but it can be somewhat dangerous to use. The `write()` method overwrites everything that was originally in the web page. In this example, I ran it from the head element, so it placed the output at the top of the web page, before any elements defined in the body element. However, if you use the `write()` function from within the body element, it'll remove any elements that were previously on the web page. I'll show you some better methods for doing this later in this chapter.

Besides the document properties and methods, JavaScript also has properties and methods that apply to each element object in the document. The following sections detail how to use those properties and methods.

JavaScript DOM object properties

Now that you have access to the objects contained in the web page, you can use JavaScript to manipulate them. Each DOM object contains one or more properties that define the actual object. There are lots of object properties JavaScript uses with objects. Table 2-3 shows a list of the more popular JavaScript DOM object properties you'll use.

TABLE 2-3 JavaScript DOM Object Properties

Property	Description
attributes	Returns a list of the object's attributes
childElementCount	Returns a list of the number of child objects the object has
childNodes	Returns a list of the object's child nodes, including text and comments
children	Returns a list of only the object's child element object nodes
classList	Returns a list of the class name attributes of an object
className	Sets or returns the value of a class attribute of an object
firstChild	Returns the first child object for the object
id	Sets or returns the id value of the object
innerHTML	Sets or returns the HTML content of the object
lastChild	Returns the last child object for the object
nodeName	Returns the name of the object
nodeType	Returns the element type of the object
nodeValue	Sets or returns the value for the object
nextSibling	Returns the next object at the same level in the tree as the object
parentNode	Returns the parent object for the object
previousSibling	Returns the previous object at the same level in the tree as the object
style	Sets or returns the value of the style property for the object

Besides these standard properties, each attribute that you assign to an HTML5 element and each CSS style property that you apply to an element becomes an object property of the DOM object as well.

Follow these steps to experiment with accessing the DOM object properties for our sample web page:

1. **Open your favorite text editor, program editor, or IDE package.**
2. **Enter the following code:**

```
<!DOCTYPE html>
<html>
<head>
<title>Testing DOM properties</title>
</head>
<body>
<body>
<h1>This is the heading of the web page</h1>
<p>This is sample text</p>
<br>
<button type="button" onclick="changeme('red')">Change background to
   red</button>
<button type="button" onclick="changeme('white')">Change background to
   white</button>
<script>
function changeme(color) {
    document.body.style.backgroundColor = color;
}
</script>
</body>
</html>
```

3. **Save the file as** `domproperties.html` **in the** `DocumentRoot` **folder for your web server.**
4. **Open the XAMPP Control Panel and start the Apache web server if it's not already running.**
5. **Open your browser and enter the following URL:**

```
http://localhost:8080/domproperties.html
```

   You may need to change the TCP port to match your Apache web server.

6. **Click the buttons to change the background color of the web page.**
7. **Close the browser window.**

The `domproperties.html` code uses two buttons to trigger the `changeme()` function. (I talk more about how to do that in Book 3, Chapter 4.) The `changeme()` function uses the `document.body` object to reference the body element in the web page. It then uses the `style` object property to reference the CSS3 styles applied to the body element.

You may be wondering why the `backgroundColor` style property isn't `background-color`, because that's how CSS3 defines that property. Unfortunately, the DOM standard doesn't like using dashes in property names. So, instead, wherever there's a dash in a CSS3 property name (such as in `background-color`), it removes the dash and capitalizes the first letter of the next word. That's how we

get `backgroundColor` as the DOM property to change the `background-color` CSS3 property on the web page.

JavaScript DOM object methods

Besides properties, JavaScript objects also contain methods. The methods provide actions to interact with the object. You've already seen a demonstration of using the `write()` method of a DOM object in JavaScript. There are plenty more object methods for you to use in your JavaScript programs to help you retrieve information about the DOM objects, modify existing DOM objects, or even add new DOM objects to your web page. Table 2-4 shows some of the more popular DOM object methods that you'll use.

TABLE 2-4 JavaScript DOM Object Methods

Method	Description
`appendChild(`*object*`)`	Adds a new child object to an existing object
`blur()`	Removes the page focus from an object
`click()`	Simulates a mouse click on the object
`cloneNode`	Duplicates an object in the DOM
`contains(`*object*`)`	Returns a true value if the object contains the specified object
`focus()`	Places the window focus on the object
`getAttribute(`*attr*`)`	Returns the value for the specified object attribute
`getElementsByClassName(`*class*`)`	Returns a list of objects with the specified class name
`getElementsByTagName(`*tag*`)`	Returns a list of objects with the specified tag name
`hasAttribute(`*attr*`)`	Returns `true` if the object contains the specified attribute
`hasAttributes()`	Returns `true` if the object contains any attributes
`hasChildNodes()`	Returns `true` of the object contains any child objects
`insertBefore(`*object*`)`	Inserts the specified object before the object
`removeAttribute(`*attr*`)`	Removes the specified attribute from the object
`removeChild(`*object*`)`	Removes the specified child object from the parent object
`replaceChild(`*object*`)`	Replaces the child object with the specified object
`setAttribute(`*attr*`)`	Sets the specified attribute of the object to the specified value
`toString()`	Converts the object to a string value

As you can see, there are quite a few different methods available for you to use when you reference a specific element in the web page. However, part of the problem with using JavaScript to dynamically change elements is finding them in the first place. The next section covers how to do that.

Finding Your Elements

As your web pages become more complicated, they'll contain dozens, hundreds, and possibly even thousands of different elements. Trying to find a specific element within that mess so you can dynamically change it can be a challenge.

There are basically two different ways to find a specific element buried within the HTML5 code in your web page:

- Using a unique feature assigned to the element to jump directly to it
- Walking the DOM tree to navigate your way down to the element's object from a specific point in the DOM tree

Both methods have their own pros and cons for using them. Obviously, if you can use a unique feature of an element (such as an `id` attribute) to reference a specific element that's the easiest way to go. However, that's not always possible, so it helps to know how to get there the hard way. The following sections describe how to use both methods for referencing element objects within the DOM tree.

Getting to the point

The easiest way to uniquely identify an element in your web page is to assign it a unique `id` attribute value. When you assign the `id` attribute to elements, you can then reference them in your JavaScript code by using the `getElementById()` method.

The `getElementById()` method returns a pointer to the DOM object with the specified `id` value. When you have the pointer to the element object, you can use any of the DOM object properties or methods to work with the element.

Follow these steps to test this out:

1. **Open your favorite text editor, program editor, or IDE package.**
2. **Enter the following code:**

```
<!DOCTYPE html>
<html>
<head>
<title>Finding an Element</title>
<script>
function changeit() {
    var answer = prompt("Enter some new text");
    var spot = document.getElementById("here");
    spot.innerHTML = answer;
}
</script>
</head>
<body>
<h1>Trying to find an element</h1>
<button type="button" onclick="changeit()">
Click to change
</button>
<p id="here">This is the original text</p>
</body>
</html>
```

3. **Save the file as `findtest.html` in the `DocumentRoot` folder for your web server.**
4. **Start the Apache web server if it's not currently running.**
5. **Open your browser and enter the following URL:**

```
http://localhost:8080/findtest.html
```

6. **Note the text that appears in the web page below the button.**
7. **Click the button and then enter some new text at the prompt dialog box.**
8. **Click OK in the dialog box.**
9. **Note the text that now appears on the web page.**
10. **Close the browser window when you're done playing.**

The `findtest.html` code defines the `changeit()` JavaScript function in the head section. The `changeit()` function uses a `prompt()` function to retrieve some text from the site visitor and then attempts to replace the text in the `p` DOM object with the new text.

To do that, it uses the `getElementById()` document method to create a pointer to the `p` object in the web page, identified by the id attribute value of `here`. After it retrieves the pointer to the `p` object, it uses the `innerHTML` object property to change the text that appears inside the `p` object.

When you run the program, you should see the heading, the same text, and a button. When you click the button, a prompt dialog box should appear, prompting you to enter some text (see Figure 2-4).

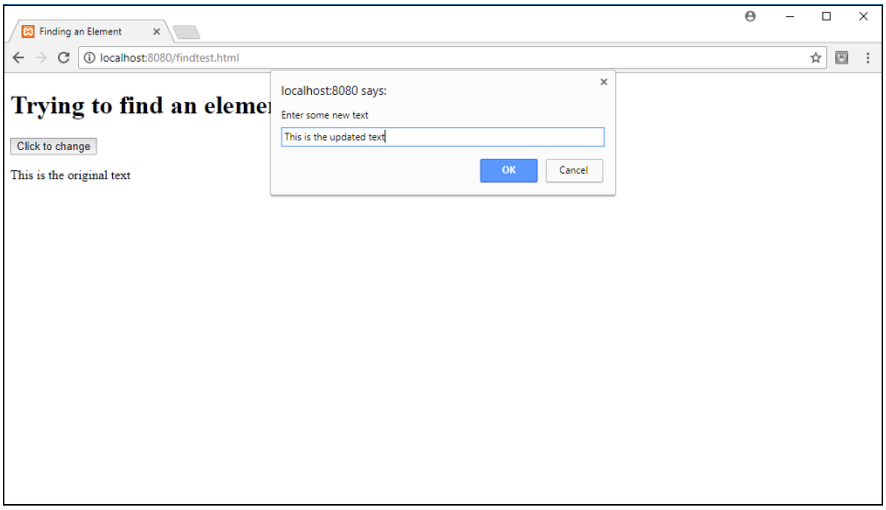

FIGURE 2-4: The initial page and dialog box for the `findtest.html` web page.

Type some text and then click OK. The browser will automatically change the content of the p element to show the text you entered into the dialog box!

You can continue doing that for as long as you like. Each time you enter new text, it'll appear in the web page automatically!

Walking the tree

Finding the DOM object for a specific HTML5 element in the DOM tree by using its `id` attribute is the preferred method, but that's not always available. Sometimes you need to find an element within the document to use as a base, and then use the element properties to find child and sibling objects:

- Use the `firstChild` property to find the first element in a group.
- Use the `nextSibling` property to find the related elements within the group.

You can then alternate between `firstChild`, `lastChild`, `nextSibling`, or `previousSibling` properties to work your way down to where you want to be in the DOM tree.

That can be tedious work, especially for large web pages. You need to be aware of exactly how all the elements appear and fit together in the web page.

Follow these steps to try this method out:

1. **Open your favorite text editor, program editor, or IDE package.**
2. **Enter the following code:**

```
<!DOCTYPE html>
<html>
<head>
<title>Walking Test</title>
<script>
function changeit() {
    var spot = document.getElementById("mylist");
    var item1 = spot.firstChild;
    var item2 = item1.nextSibling;
    var item3 = item2.nextSibling;
    var item4 = item3.nextSibling;
    item1.innerHTML = "Cake";
    item2.innerHTML = "Ice Cream";
    item3.innerHTML = "Cookies";
    item4.innerHTML = "Fudge";
}
</script>
</head>
<body>
<h1>Changing elements by walking</h1>
<h2>Here's a list of food to buy</h2>
<ul id="mylist"><li>Carrots</li><li>Brussel Sprouts</li><li>Eggplant
   </li><li>Tofu</li></ul>
<button type="button" onclick="changeit()">
Change the list
</button>
</body>
</html>
```

3. **Save the file as `walkingtest.html` in the `DocumentRoot` folder for your Apache web server.**
4. **Start the Apache web server if it's not already running.**

5. **Open your browser and enter the following URL:**

```
http://localhost:8080/walkingtest.html
```

6. **Examine the items in the list.**
7. **Click the button.**
8. **Note the new items in the list.**
9. **Close the browser window when you're done.**

The `walkingtest.html` code defines an `id` attribute for the unordered list element, but each of the items within the list isn't uniquely identified. In order to reference them, the code uses the `firstChild` and `nextSibling` object property values to walk its way through the list of items. When you click the button, all the items in the list are replaced, as shown in Figure 2-5.

FIGURE 2-5: The `walkingtest.html` results.

The code finds the `ul` object by using the `id` attribute value of the ul element. It assigns that object to the variable `spot`. Then the code can reference the individual list items based on that location in the DOM tree. The first child of the `ul` object is the first `li` object for the list. The `firstChild` property returns a pointer to that object, which the code stores in the `item1` variable. Next, the code uses the `nextSibling` property of the `item1` variable, which returns a pointer to the next item in the list and is stored in the `item2` variable. That continues on, using the `nextSibling` property for each item to find the next item in the list. After the code retrieves pointers to all the list items, it uses the `innerHTML` property to change the text for each item.

WARNING

Be careful how you create the list in the code. If you place each list item on a separate line, the code won't work! That's because the browser assigns any white space between elements as a text object in the DOM. So the `nextSibling` property will point to the new line character text object and not the next li object in the list! It's important to remember that when working with the positional properties of objects.

Working with Document Object Model Form Data

When you use HTML5 forms in your web pages, you usually incorporate quite a few different elements — text boxes, text areas, check boxes, and radio buttons. Your JavaScript code can use the DOM tree objects to manipulate all these elements. The following sections show you how to use the DOM tree to work with different types of form elements.

Text boxes

Handling data in a text input element is a little different from what I did with the p element. Because the input element is a one-sided tag, there's no `innerHTML` property to store the text that's inside the text box.

Instead, you need to use the `value` attribute of the object to read any text that may already be in the text box (whether placed there by the `value` attribute or typed by the site visitor). To do that, you use the `value` object property:

```
var textbox = document.getElementById("test");
var data = textbox.value;
```

You can also use the `value` property to write data to the text box. That code looks like this:

```
var textbox = document.getElementById("test");
var answer = prompt("Enter text to change");
textbox.value = answer;
```

This provides for an easy way to create a message area on your web page for displaying short messages, such as status messages. Just place a `textbox` input element near the bottom of the web page, and change the `value` property of it with any message you need to display.

There are also a few other DOM object properties associated with `textbox` objects that can come in handy. Table 2-5 shows the DOM `textbox` properties available.

TABLE 2-5 The `textbox` DOM Properties

Property	Description
`autocomplete`	Sets or retrieves the value of the autocomplete attribute
`autofocus`	Sets or retrieves whether the text box gets the window focus when the web page loads
`defaultValue`	Sets or retrieves the default value assigned to the text box
`disabled`	Sets or retrieves whether the text box is disabled in the form
`form`	Retrieves the parent form the text box belongs to
`list`	Retrieves the data list associated with the text box
`maxLength`	Sets or retrieves the maximum length of the text box
`name`	Sets or retrieves the name attribute for the text box
`pattern`	Sets or retrieves the pattern attribute for the text box
`placeholder`	Sets or retrieves the placeholder attribute for the text box
`readOnly`	Sets or retrieves whether the text box is read only
`required`	Sets or retrieves whether the text box is a required field in the form
`size`	Sets or retrieves the value of the size attribute for the text box
`type`	Retrieves the type of element the text box is
`value`	Sets or retrieves the value attribute for the text box

With these few properties, you have full control to dynamically modify any text box that appears on the web page.

Text areas

The `textarea` DOM object works similar to the `textbox` object. Instead of the `innerHTML` property, you use the `value` attribute to retrieve any text from the text area or place any new text into the text area.

There are a few other properties that are unique to the `textarea` object:

- `cols`: Sets or retrieves the number of columns assigned to the text area
- `rows`: Sets or retrieves the number of rows assigned to the text area
- `wrap`: Sets or retrieves whether text can auto-wrap within the text area

As you can tell, you can dynamically change the size of the text area in a web page using JavaScript and the DOM object properties. That can create quite an effect as your site visitor is filling out the form.

Check boxes

The `checkbox` object is another oddity in the DOM. A check box in a form provides for a yes/no type of answer — either the visitor checks the check box or the box is unchecked. You can test for that condition using the DOM `checked` property:

```
var pizza = document.getElementById("pizzabox");
if (pizza.checked) {
   alert("your pizza will be delivered shortly");
}
```

You can also set whether the check box is checked by assigning the property a `true` or `false` value:

```
pizza.checked = true;
```

Table 2-6 shows all the DOM object properties that are supported when using check boxes.

TABLE 2-6 The `checkbox` DOM Properties

Property	Description
`autofocus`	Sets or retrieves whether the check box gets the focus when the web page loads
`checked`	Sets or retrieves the state of the check box
`defaultChecked`	Retrieves the default state of the check box
`defaultValue`	Retrieves the default value assigned to the check box
`disabled`	Sets or retrieves whether the check box is disabled

Property	Description
`form`	Retrieves the parent form the check box belongs to
`intermediate`	Sets or retrieves the intermediate state of the check box
`name`	Sets or retrieves the name assigned to the check box element
`required`	Sets retrieves whether the check box must be checked before submitting the form
`type`	Retrieves the type of element the check box is
`value`	Sets or retrieves the value associated with the check box

That gives you full control over how the check boxes behave in your web page.

Radio buttons

Working with radio buttons is always a complicated matter. All the radio buttons in the same group use the same name property, so the browser can handle them as a group. Remember, only one radio button in the group can be selected at any time.

Handling data from a radio button requires using the checked and value object properties, just like the `checkbox` object. Because all the radio buttons use the same name, the value attribute is crucial in determining if you're working with the correct radio button in the form.

IN THIS CHAPTER

- Loading the jQuery library
- Using jQuery in your web pages
- Finding elements
- Replacing data
- Changing styles
- Adding nodes
- Using animation

Chapter 3
Using jQuery

As you code dynamic web applications using JavaScript, you'll find yourself using the same statements and features over and over again to create dynamic effects on your web pages. As it turns out, JavaScript developers around the world use the same statements and features to implement the same effects on their web pages, too!

Because of that, lots of work has been done by developers in trying to create a standard JavaScript library of useful functions. Instead of having to write the same JavaScript statements over and over, you just run a simple function from a prebuilt library. That makes life for the JavaScript programmer much easier!

By far the most common JavaScript library used around the world today is the jQuery library. The jQuery library was written to simplify five main functions that JavaScript is commonly used for:

- Finding content in an HTML5 document
- Changing content in an HTML5 document
- Creating animations using CSS
- Listening for web page events (see Book 3, Chapter 4)
- Communicating with remote servers (see Book 6, Chapter 3)

I cover how to use the first three features of jQuery in this chapter. In the next chapter, I show you how to use jQuery to simplify listening to actions your site visitors take while on your web pages. Then in Book 6, Chapter 3, I show how to use jQuery to simplify communicating with PHP programs running on the server from inside your web pages. But for now, let's examine how to use the basics of jQuery.

Loading the jQuery Library

Before you can use the jQuery library functions in your web page, you need to load the library functions. The jQuery library is nothing more than a standard external JavaScript program that defines lots of handy functions for us. The project freely provides the JavaScript code library for use in any application, whether it's commercial or open source.

The main website for the jQuery project is `www.jquery.com`. From there, you can find documentation on jQuery, as well as the software download packages. There are two main versions of jQuery:

- The latest production version (at the time of this writing, 3.2.1)
- The latest development version (which isn't assigned a version number)

For all your website work, you'll want to use the latest production version of the jQuery library. The development package is for testing cutting-edge features and isn't guaranteed to work correctly at all times in all situations. This could lead to issues in your dynamic web application.

After you decide to use the latest production version of the jQuery library, there are actually four different versions of the library that you can use in your application:

- **Uncompressed:** The full jQuery library in an uncompressed file
- **Minified:** The full jQuery library in a compressed file
- **Slim:** Everything except support for animation and Ajax in an uncompressed file
- **Slim minified:** Everything except support for animation and Ajax in a compressed file

For most purposes, you'll be fine using the minified version of the file. This contains all the jQuery features, but in a compressed file so that it will load faster for your site visitors.

You've decided to use the minified version of the latest production version of the jQuery library software (because you trust me completely), but there's still one more decision for you to make. Because the jQuery file is a JavaScript library, your application needs to load it for each web page that contains jQuery code. This can get somewhat tedious for large applications.

You can either download the jQuery library file to your own server to host, making it easier for your website visitors to access it along with the rest of your application files, or you can point your site visitors' browsers to download the jQuery library file from a content delivery network (CDN) server. The following sections walk through how to use both options.

Option 1: Downloading the library file to your server

Sometimes it's better to have your website visitors download all the files necessary for your dynamic web application from one place — your own server. To do that, you need to have the jQuery library file installed on your web server, in the `DocumentRoot` folder so that your site visitors can access it.

Downloading the file from the main jQuery web page and installing it on your web server is a fairly easy process. Just follow these steps:

1. **Open your web browser and go to** `www.jquery.com/download`.
2. **Click the link to the compressed production version of jQuery.**

   At the time of this writing, it's version 3.2.1.

   Your browser downloads the file to the default download folder. It should have a name something like `jquery-3.2.1.min.js`. (The numbers will be different if the version number has changed since this book was written.)
3. **Copy the file to the** `DocumentRoot` **folder for your web server.**

   If you're using XAMPP in Windows, that's `c:\xampp\htdocs`; for XAMPP in macOS, it's `/Applications/XAMPP/htdocs`.

To load the jQuery library in your web application, you'll need to include a `<script>` tag in the head element of your web page. The `<script>` tag should point to the jQuery library file that you downloaded:

```
<script src="jquery-3.2.1.min.js"></script>
```

WARNING

It's important that the browser loads the jQuery library file before you use any jQuery functions in your application. Place the `<script>` tag near the top of the head element section, after the `<title>` tag.

Option 2: Using a content delivery network

One downside to hosting the jQuery library file on your own server is that all your site visitors will need to download it directly from your server, creating an additional load on your server. To prevent that, you can point the `<script>` tag to load the jQuery library file from a CDN.

A CDN provides content for applications from a common server or group of servers. Your website visitors can download the jQuery library file from the nearest CDN to their location, which may speed up the time it takes to load your web page.

The jQuery project runs its own CDN to host the latest jQuery library file and provides the `<script>` tag formats required for each of the different library file options. At the time of this writing, they host that at `https://code.jquery.com`. Here's the current `<script>` tag to use to load the jQuery library from the jQuery CDN website:

```
<script src="https://code.jquery.com/jquery-3.2.1.min.js" integrity="sha256-hwg4
   gsxgFZhOsEEamdOYGBf13FyQuiTwlAQgxVSNgt4=" crossorigin="anonymous"></script>
```

The `integrity` and `crossorigin` attributes are for the Subresource Integrity (SRI) feature, which helps the browser ensure the downloaded file hasn't been tampered with. This is a nice feature to add to ensure your site visitors aren't using a hacked library file.

TIP

Besides the jQuery CDN website, Google and Microsoft also host the jQuery library files on their own CDN websites. If your site visitors are geographically disbursed throughout the world, it may be faster for them to download the jQuery library file from a Google or Microsoft server. The `https://code.jquery.com` website provides instructions on how to use the Google and Microsoft CDN websites.

Using jQuery Functions

After you have the jQuery library file downloaded to the site visitor's browser, you're ready to start using jQuery functions in your dynamic web application.

All jQuery functions must be embedded in the special `jQuery()` function. This signals to the browser that it must use the jQuery library to process the functions used in the code. The general format for embedding jQuery code into your web page thus looks like this:

```
<script>
    jQuery(code);
</script>
```

Because jQuery is still JavaScript code, you must surround the jQuery code using the standard script HTML5 element. The actual jQuery code itself must also be embedded within the `jQuery()` function.


TIP

If things are starting to look complicated, don't worry — this is a standard format. After you've written a few jQuery programs, you'll feel right at home using it. That said, there is a shortcut that can come in handy. Instead of using the `jQuery()` function name, you can use the `$()` shortcut function name.

Finding Elements

One of the main features of jQuery is to help you find HTML5 elements in the web page to manipulate. In Book 3, Chapter 2, I show you how you need to use the JavaScript `getElementById()`, `getElementByClassName()`, or `getElementByTagName()` functions to find elements in the web page. Needless to say, that gets fairly complicated when you start working with large web pages.

The jQuery library greatly simplifies this process. It incorporates the same selector method that CSS uses to apply styles to HTML5 elements. For example, to find the h1 element in a web page, you just use the jQuery code:

```
$("h1");
```

Now that's easy! If you want to find an element based on an `id` attribute value, you use the same format as in CSS:

```
$("#warning");
```

Likewise with class names:

```
$(".warning");
```

After you've found the HTML5 element that you're looking for, jQuery allows you to easily apply lots of different functions to modify the element — but more on that later in this chapter.

Follow these steps to experiment with retrieving text from an HTML5 p element using your new jQuery skills:

1. **Open your favorite text editor, program editor, or integrated development environment (IDE) package.**
2. **Type the following code:**

```
<!DOCTYPE html>
<html>
<head>
<title>Testing jQuery</title>
<script src="jquery-3.2.1.min.js"></script>
</head>
<body>
<h1>This is my heading</h1>
<p>This is some content on my web page</p>
<script>
    var data = $("p").text();
    alert(data);
</script>
</body>
</html>
```

3. **Save the file as `jquery1.html` in the `DocumentRoot` folder of your web server, where you also saved the jQuery library file.**
4. **Open the XAMPP Control Panel and start the Apache web server.**
5. **Open your browser and enter the following URL:**

```
http://localhost:8080/jquery1.html
```

   You may need to modify the TCP port used in the URL to match your web server.

The jQuery code finds the p element in the web page, retrieves the text that it contains, and then stores it in the `data` JavaScript variable. You can then use that as any other JavaScript variable, including displaying it using an `alert()` function, as shown in the code. When you run the program, you should see the output shown in Figure 3-1.

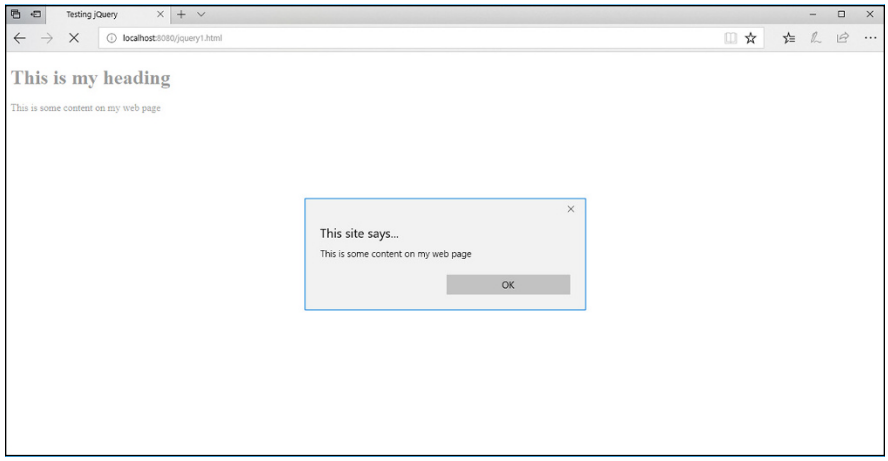

FIGURE 3-1: The output from the `jquery1.html` program.

Congratulations on running a successful jQuery program! However, one little issue remains: I placed the script element that contained the jQuery code at the very end of the body element section. There's a reason for that. The jQuery find feature can only find elements that have already been processed by the browser into the Document Object Model (DOM) tree. If you try to run a jQuery find operation before the browser completes building the DOM tree, it won't find the elements. That can be a huge problem with a dynamic web application, but there's a way around that: the `.ready()` function.

The `.ready()` function causes jQuery to wait until the browser has completely loaded the DOM tree, and all the HTML5 elements contained in the web page are available. After that's done, the `.ready()` function runs whatever jQuery code you place inside of it.

To do that, you need to embed your jQuery code into a code block that now looks like this:

```
<script>
jQuery(document).ready(function() {
     code
});
</script>
```

The `.ready()` function uses an anonymous function that it runs when the browser fully loads the DOM tree. You just embed your jQuery code inside the anonymous function, and you're guaranteed it won't run until the DOM tree is ready. When you use this method, you can place your jQuery script element anywhere in the web page code, including the head element section. That makes it much easier to spot the embedded jQuery code, instead of having to go hunting all around the web page code file for it.

Using this method, the code that you created earlier would look like this:

```
<script>
    jQuery(document).ready(function() {
        var data = $("p").text();
        alert(data);
    });
</script>
```

Now you can place this script element block into the head element of the web page and it'll work just fine!

WARNING

Although you can now place your jQuery code anywhere in the head element section, you still need to place the script element that loads the jQuery library before any jQuery code.

Replacing Data

After you find the HTML5 elements in your web page, the next step is to modify the content of the web page. Fortunately, jQuery makes that step easier, too. This section shows how jQuery allows you to change the text, HTML code, and even attributes of the HTML5 elements contained in your web pages.

Working with text

As shown in the previous code example, adding the `.text()` function to the jQuery object retrieves the text contained in the object. You can use the same `.text()` function to replace that text. Just place the text you want to use as a parameter to the `.text()` function:

```
$("p").text("This has changed");
```

Follow these steps to test this out:

1. **Open your favorite text editor, program editor, or IDE package.**
2. **Type the following code:**

```
<!DOCTYPE html>
<html>
<head>
```

```
<title>Testing jQuery Replacing Text</title>
<script src="jquery-3.2.1.min.js"></script>
<script>
    jQuery(document).ready(function() {
        $("button").click( function() {
            $("p").text("This has changed!");
        });
    });
</script>
</head>
<body>
<h1>This is my heading</h1>
<p>This is some content on my web page</p>
<button>Test button</button>
</body>
</html>
```

3. **Save the file as** `jquery2.html` **in the** `DocumentRoot` **folder of your web server.**

4. **Make sure the Apache web server is running, open your browser, and enter the following URL:**

```
http://localhost:8080/jquery2.html
```

5. **Click the button on the web page and watch the text on the page.**

There are a couple of new things I threw into this example, so let me explain a bit:

- **I added a button element at the bottom of the web page.** Notice that I didn't need to add the `onclick` attribute for the button as I used in the preceding chapter with JavaScript. The jQuery library is kind enough to do that for us!
- **In the jQuery code I added the following line:**

```
$("button").click( function() {
```

The first part you should recognize — it finds the first button element on the web page. The code then applies the `.click()` function to that object. The browser runs this function when it detects that the site visitor clicks the referenced button. In this case, when the button gets clicked, the code triggers another anonymous function. The code in the anonymous function is

```
$("p").text("This has changed!");
```

When you click the button, you should see the content of the p element change to the text you specify in the jQuery code, right before your eyes, as shown in Figure 3-2!

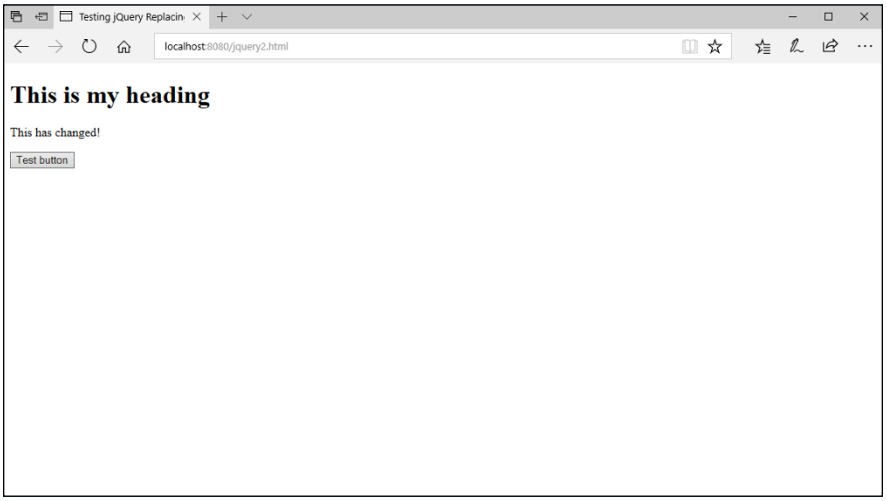

FIGURE 3-2: The result of the `jquery2.html` program.

You use this method to change any type of text content in any type of block element.

Working with HTML

The `.text()` function allows you to change the text contained within an element, but it doesn't change the HTML5 code for the element. You can do that by using the `.html()` function:

```
$("p").html("<h1>This changed to a heading</h1>");
```

Notice that you need to supply the full HTML5 element tags along with the text that you want to appear in the element.

If you replace the original `.text()` function in the example code with this line, when you click the button the p element turns into an h1 element, and the browser styles it accordingly, as shown in Figure 3-3!

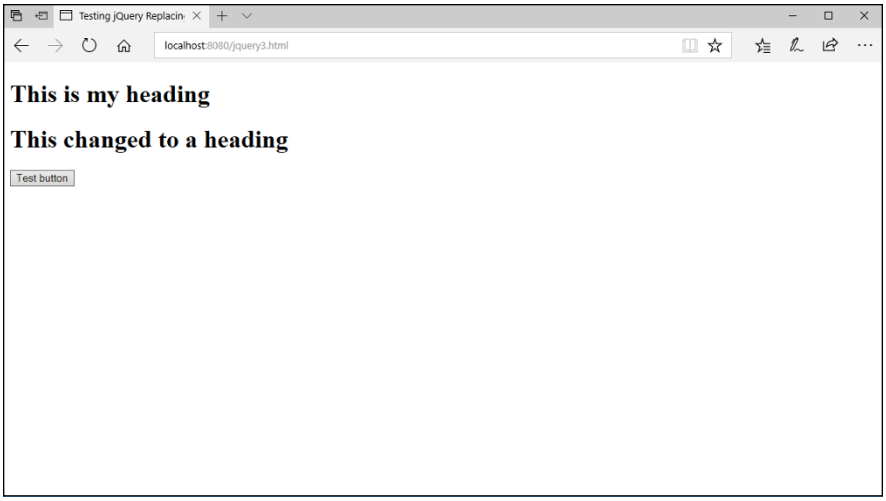

FIGURE 3-3: Changing the element using the `.html()` function.

Working with attributes

Not only can you modify the text and HTML code in an element, but you can also retrieve and set attributes for the element using the `.attr()` function. To retrieve an attribute value, use the following format:

```
$(selector).attr("attribute");
```

To set a value associated with an attribute of the element, the format is as follows:

```
$(selector).attr("attribute", "value");
```

This allows you to change the appearance of an element by modifying the attributes as needed from your jQuery code.

Working with form values

One of the greatest features of the jQuery library is the ability to dynamically read and modify data in HTML5 forms. This feature comes in handy if you need to validate form data as your site visitors are typing it into the form, before it even leaves their workstations!

The `.val()` function provides access to the value attribute for input elements:

```
var data = $("input").val();
```

The `value` HTML5 attribute also allows you to set the default value that appears in the input form. So by adding a value to the `.val()` function, you can control what text appears in the form field as well:

```
$("input").val("Enter your last name");
```

The next chapter shows you how you can trigger the `.val()` function as your site visitor presses each key as she's typing in the form fields. With that feature, you can create dynamic search results as the visitor is typing!

Changing Styles

The jQuery library can do more than just change the content that appears in the web page. It also contains functions that help you dynamically change the styles that the browser applies to elements on the web page.

This section discusses how you can access the CSS properties assigned to an object, as well as modify them on the fly as your site visitor interacts with the web page.

Playing with properties

Book 2, Chapter 2, shows how you can apply CSS3 styles to elements to not only style them but also position them on the web page. Style rules defined in an internal or external style sheet determine just how the browser displays and positions the element on the web page.

The jQuery library provides some functions for you to use to help manipulate the CSS3 properties that the browser applies to elements. The first one I talk about is the `.css()` function.

The `.css()` function allows you to retrieve and set individual properties or a group of properties for any element in the web page. To retrieve the current value assigned to a CSS3 property, you use the following format, where *`selector`* is the CSS-style selector for finding the element and *`property`* is the CSS3 property name you want to retrieve:

```
$(selector).css(property);
```

For example, to determine the background color applied to a div element, you'd use the following:

```
var color = $("div").css("background-color");
```

When you use JavaScript to set CSS3 properties, you have to use different names, because JavaScript doesn't support the dash in the property names. Notice that with jQuery you use the actual CSS property name that you're already used to using — nothing new to learn!

Then, as you can probably guess by now, to set the CSS3 property for an element, you just add the value as the second parameter to the function call:

```
$("div").css("background-color", "red");
```

Follow these steps to test this out:

1. **Open your favorite text editor, program editor, or IDE package.**
2. **Type the following code:**

```
<!DOCTYPE html>
<html>
<head>
<title>Changing Properties with jQuery</title>
</head>
<style>
   div {
       background-color:yellow;
   }
</style>
<script src="jquery-3.2.1.min.js"></script>
<script>
    jQuery(document).ready(function() {
        $("button").click(function() {
            $("p").css("background-color", "red");
            $("p").css("font-size", "50px");
      });
   });
</script>
<body>
<div id="container">
<h1>This is my heading</h1>
<p>This is some content on my web page</p>
<button>Test button</button>
</div>
</body>
</html>
```

3. **Save the file as `jquery4.html` in the `DocumentRoot` folder of your web server.**

4. **Make sure the Apache web server is running, and then open your browser and enter the following URL:**

```
http://localhost:8080/jquery4.html
```

5. **Click the button and watch the content from the p element on the web page.**

This version of our example program uses a short CSS internal style sheet to set the background color of the div element around the elements on the web page. When jQuery detects the button click, it applies two new styles to the p element — changing the background color to red and increasing the font size to 50 pixels. Figure 3-4 shows what this looks like after you click the button.

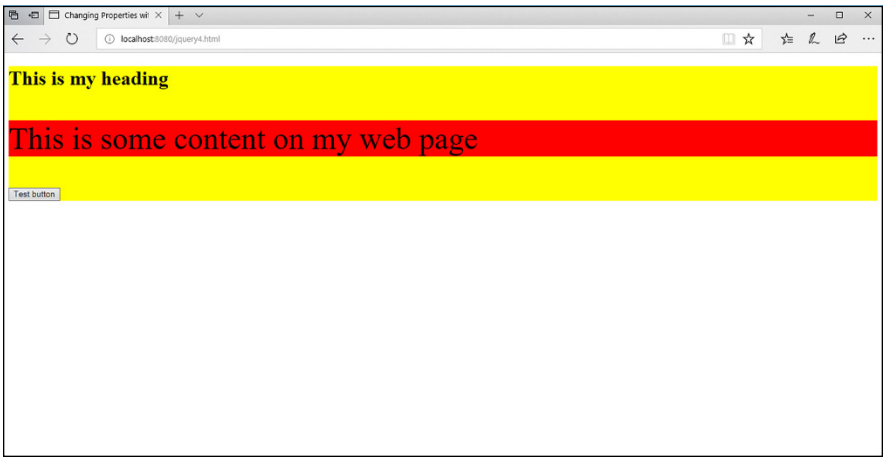

FIGURE 3-4: The result of the `jquery4.html` program.

You can apply as many styles to as many elements as you need within the event trigger. However, the more styles you apply, the messier it gets. But fortunately, there are a couple of solutions to that problem.

Using CSS objects

Instead of piling multiple `.css()` function lines on top of each other, trying to change lots of different style properties, you can create a *style object* in jQuery. The style object allows you to specify styles just as you do in the CSS3 style sheet and provide multiple styles in a single `.css()` function.

The style object has the following format:

```
{"property1":"value1", "property2":"value2"...}
```

Now things are really starting to look familiar — very similar to how CSS rules define properties and their values! Using this format, you can combine the two style changes in the `jquery4.html` example to one line:

```
$("p").css({"background-color":"red", "font-size":"50px"});
```

This is a great way to make dramatic changes to the web page layout and style dynamically in response to events that your site visitor triggers, such as changing the background color of text boxes as data is entered, or changing the location of important content that may be missed.

Using CSS classes

Things are starting to get pretty fancy with your jQuery style coding, and you're starting to introduce another issue to your program. Now you're embedding styles inside your jQuery code, separate from the rest of the styles defined in the CSS3 style sheets. That can make things somewhat confusing when you're trying to troubleshoot a problem, or even if you're trying to go back over your own code several months later!

To solve that issue, the brilliant jQuery developers added another set of functions that interact with CSS3 class rules. With the jQuery class functions, you can add, remove, or even toggle a class to an element. Table 3-1 shows the classes available for you.

TABLE 3-1 The jQuery Class Functions

Function	Description
`.addClass(`*`class`*`)`	Adds the specified class to the element
`.hasClass(`*`class`*`)`	Returns a `true` value if the element contains the specified class attribute
`.removeClass(`*`class`*`)`	Removes the specified class from the element
`.toggleClass(`*`class`*`)`	Alternately adds and removes the specified class each time it's called

Now all you need to do is place the group of style properties you need into a class rule in your CSS style sheet definitions, and then add, remove, or even toggle the class for the element. The `.hasClass()` function allows you to check what class is currently assigned to the element.

Follow these steps to try this feature out:

1. **Open the `jquery4.html` file from the previous example in your editor.**
2. **Modify the code so that it looks like this:**

```
<!DOCTYPE html>
<html>
<head>
<title>Changing Properties with jQuery</title>
</head>
<style>
   div {
       background-color:yellow;
   }

   .changeit {
       background-color:red;
       font-size:50px;
   }
</style>
<script src="jquery-3.2.1.min.js"></script>
<script>
   jQuery(document).ready(function() {
       $("button").click(function() {
            $("p").toggleClass("changeit");
        });
   });
</script>
<body>
<div id="container">
<h1>This is my heading</h1>
<p id="content">This is some content on my web page</p>
<button>Test button</button>
</div>
</body>
</html>
```

3. **Save the file as `jquery5.html` in the `DocumentRoot` folder of the Apache web server.**

4. **Ensure that the Apache web server is running, and then open your browser and enter the following URL:**

```
http://localhost:8080/jquery5.html
```

5. **Click the button multiple times to toggle the style effects on and off.**

In the updated code, I added a new class rule to the internal style sheet:

```
.changeit {
    background-color:red;
    font-size:50px;
}
```

And I used the `.toggleClass()` function to apply it to the p element:

```
$("p").toggleClass("changeit");
```


TIP

For even more fun, you can use the `.show()` and `.hide()` jQuery functions, which pretty much do what they say. They change the `display` CSS3 property of the element to `block` (for `.show()`) or `none` (for `.hide()`).

Changing the Document Object Model

Not only can you use jQuery to modify content and styles of the existing elements in your Web page, you can also use it to add or remove entire elements! There are a handful of different jQuery functions available for manipulating the element nodes contained in the DOM tree.

Adding a node

You can use jQuery to add a new node to the DOM tree to display additional content as needed. Table 3-2 shows the functions available for adding new nodes.

TABLE 3-2 The jQuery Functions to add DOM Nodes

Function	Description
`.after()`	Adds a node after an existing node
`.append()`	Adds a node to the end of an existing node
`.appendTo()`	Adds a new node to the end of an existing node
`.before()`	Adds a node before an existing node
`.insertAfter()`	Adds a new node after an existing node
`.insertBefore()`	Adds a new node before an existing node
`.prepend()`	Adds a node to the beginning of an existing node
`.prependTo()`	Adds a new node to the beginning of an existing node

Note the subtle difference between the `.after()` and `.append()` functions. The `.append()` function adds the new node to the end of the existing node, so it becomes a child node of the existing node in the DOM. The `.after()` function, on the other hand, adds a new sibling node after the existing node in the DOM. Likewise for the `.before()` and `.prepend()` functions.

For example, you can add a new p element to the existing p element in your example program by adding the following code:

```
$("p").after("<p>This is a new node</p>");
```

As you would expect, when you run one of these functions, the new node immediately appears in the web page.

Removing a node

The jQuery library provides two functions for you to remove existing nodes from the DOM:

- `.empty()`: Removes all child nodes from the specified node
- `.remove()`: Removes the specified node

It's important to note that the `.empty()` function doesn't remove the specified node — it just removes any child nodes associated with the node.

Playing with Animation

When you run the `jquery4.html` example code to change the background color and font size, the changes occur almost immediately after you click the button. That's a pretty stark effect, which can be toned down some.

One of the cooler features of jQuery is the ability to animate style changes. With the `.animate()` function, you can specify an endpoint style for the content, and jQuery will slowly work its way to that endpoint from the current style. This slow morphing process causes the web page to look like it's animated!

This is a hard feature to explain without actually viewing it, so follow these steps to try it out:

1. **Open the `jquery4.html` file in your editor.**
2. **Change the line that sets the `font-size` style property to use the `.animate()` function.**

   Look for the following line:

   ```
   $("p").css("font-size", "20px");
   ```

   And change it to this:

   ```
   $("p").animate({"font-size": "50px"});
   ```

3. **Save the new file as `jquery6.html` in the `DocumentRoot` folder for the web server.**
4. **Ensure the Apache web server is running and then open your browser and enter the following URL:**

   ```
   http://localhost:8080/jquery6.html
   ```

5. **Click the button and watch the animation.**

The `.animate()` function requires a CSS object, so even if you just specify one property to change, you must use the object format. When you click the button, instead of an instant change in font size, you see the text "grow" to get to the font size. You can change the rate of animation by adding a second parameter to the `.animate()` function — the milliseconds it takes to get to the final endpoint value. The default is 400 milliseconds (ms).

IN THIS CHAPTER

- Exploring web page events
- Using events with JavaScript
- Working with jQuery and events

Chapter 4

Reacting to Events with JavaScript and jQuery

In the previous chapters in this minibook, I explain how to incorporate both JavaScript and jQuery into your HTML5 code to help create a dynamic web application. The trick to using JavaScript and jQuery, though, is knowing when to use them. How are you supposed to know when your site visitor is hovering the mouse pointer over a product in your catalog to pop up more information? Fortunately, your web page is talking to you, telling you what your website visitors are doing at all times. All you need to do is listen to your web page and direct your JavaScript or jQuery code accordingly. That's exactly what this chapter shows you how to do.

Understanding Events

The world is full of events. There are birthday events, holiday events, school events, all types of events competing for your time. Your world is loaded with events, and it's your job to determine which events to participate in (your birthday) and which ones to ignore (Talk Like a Pirate Day?).

The same is true with your web application. There are lots of events that your site visitor generates as she interacts with your web page. Each time your site visitor

moves the mouse, that's an event. Each time she types text into a form field, that's an event. And of course, each time she clicks the mouse on a link or button, those are events, too. The key to successful dynamic web applications is to detect the events you need and ignore the ones you don't need.

Event-driven programming

Most of the JavaScript code earlier in this minibook uses *procedural programming.* In procedural programming, the browser follows your JavaScript code line by line, processing each statement as it appears in the program.

There's another way to write programs, called *event-driven programming.* With event-driven programming, your program centers around events that occur in the web page. You must define a list of events to monitor, and if one of those events occurs, the browser runs the JavaScript function you've defined for the event.

With event-driven programming, you need to know what events to watch for. This section details the events that are generated by the browser on the different activities that occur while your site visitor views your web page.

Watching the mouse

No, I'm not talking about Mickey. I'm talking about paying attention to what your site visitor is doing with the mouse device on his or her workstation. Believe it or not, your browser tracks every single move and action your mouse takes. You can tap into that wealth of information with your JavaScript or jQuery programs.

As you can imagine, there are many different events that the mouse generates as you move it around. Table 4-1 shows a list of the different mouse event names generated by the browser as defined in HTML5 and JavaScript. Later on, I show you the jQuery version of the event names.

As you can tell from the list in Table 4-1, you can watch exactly what your site visitors are doing while viewing your web page. (Scary!) Although this information can be useful, it can also result in information overload. The key to successful mouse watching is to only watch for the important events, such as when the site visitor clicks the primary mouse button on an object in the web page or when the mouse is hovering over an object.

WARNING

It's not a good idea to write code that watches the `onmousemove` event, because that event triggers for every pixel the mouse pointer moves to on the screen, generating thousands of events at a time!

TABLE 4-1 Mouse Events

Event	Description
`onclick`	The primary mouse button has been clicked.
`oncontextmenu`	The secondary mouse button has been clicked.
`ondblclick`	The primary mouse button has been double-clicked.
`onmousedown`	The primary mouse button has been depressed.
`onmouseenter`	The mouse pointer has entered a specific area in the window.
`onmouseleave`	The mouse pointer has left a specific area in the window.
`onmousemove`	The mouse pointer is moving.
`onmouseover`	The mouse pointer is hovering over an object.
`onmouseout`	The mouse pointer has left a specific area in the window.
`onmouseup`	The primary mouse button has been released.

Listening for keystrokes

The keyboard talks to the browser, too. You can watch for key events in your JavaScript or jQuery programs just as you watch the mouse. Unlike the long list of mouse events, there are only three keyboard events for you to work with:

- `onkeydown`: A key is being pressed down.
- `onkeypress`: A key has been pressed and released.
- `onkeyup`: A key has been released.

Notice the subtle difference between the three events. The `onkeydown` event only triggers while the site visitor is pressing the key. Both the `onkeypress` and `onkeyup` events trigger when the site visitor releases the key. Granted, for most typing situations, the difference is very small, but for some applications (for example, games), it can be useful to know how long a key is being pressed, which you can only get from the `onkeydown` event.

WARNING

The term *keystroke* may be misleading. There are some keys on the standard keyboard that don't generate a keystroke themselves, such as the Shift, Alt, and Ctrl keys on a Windows keyboard. These keys are modifiers for other keys that generate the keystrokes.

Paying attention to the page itself

Even the web page itself has events that your JavaScript and jQuery programs can listen for. Before HTML5, there were only a handful of page events that you could tap into. The newer HTML5 standard has defined a lot more page events to work with. Table 4-2 lists the more common HTML events that you may run into.

TABLE 4-2 Page Events

Event	Description
`onafterprint`	Triggers after the site visitor prints the web page
`onbeforeprint`	Triggers before the site visitor prints the web page
`onbeforeunload`	Triggers just before the web page is removed from the browser window
`onerror`	Triggers when there is an error in loading a required file for the web page
`onhaschange`	Triggers when the server address of the URL has changed
`onload`	Triggers when the body of the web page loads
`onmessage`	Triggers when a message is sent to the browser window
`onoffline`	Triggers when the site visitor sets the browser to view the web page offline
`ononline`	Triggers when the site visitor sets the browser to view the web page online
`onpagehide`	Triggers when the site visitor navigates away from the web page
`onpageshow`	Triggers when the web page appears in the browser window
`onpopstate`	Triggers when the browser's history changes
`onresize`	Triggers when your site visitor resizes the browser window
`onstorage`	Triggers when a web storage area is updated
`onscroll`	Triggers when the site visitor moves the scrollbar in the browser window
`onunload`	Triggers when the web page is removed from the browser window

The web page events allow you to track when your web page first appears in the site visitor's browser and when it leaves (and even just before it leaves). This gives you the opportunity to load things right up front when the page appears, or perform some operation as the page is about to disappear from the browser window.

Focusing on JavaScript and Events

JavaScript and HTML5 team up to provide a way for your program to listen for events and perform some type of action when they occur. The HTML5 element code registers a JavaScript function for the browser to run when a specific element event occurs.

Different HTML5 elements generate different events based on how they interact with the site visitor on the web page. The following sections walk you through how to set up a JavaScript event monitor for different HTML5 elements.

Saying hello and goodbye

The page events allow you to monitor when the web page loads and unloads from the site visitor's browser. You use these in the `<body>` tag of the web page to specify any `onload` or `onunload` event functions you need to run:

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

In this example, the browser runs the `welcome()` JavaScript function when the web page first loads into the browser window, as shown in Figure 4-1.

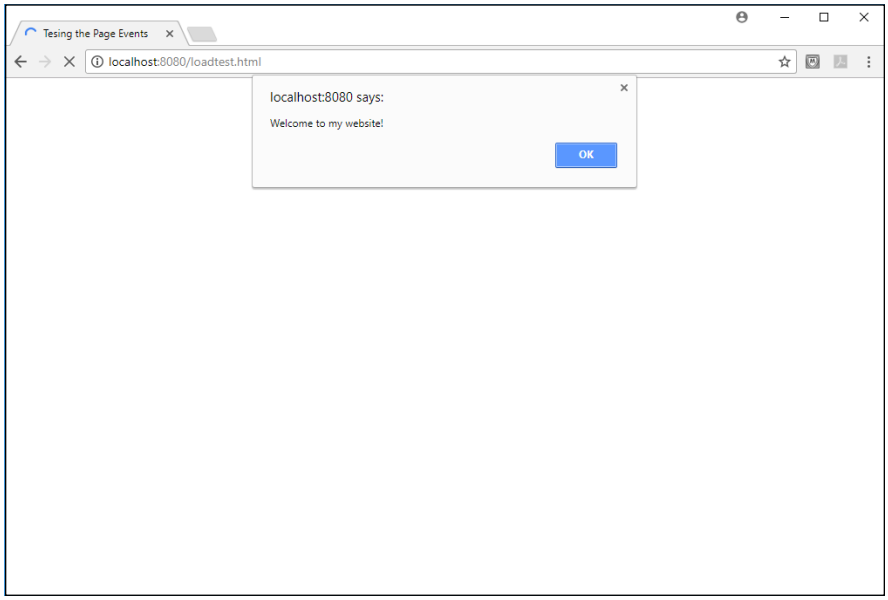

FIGURE 4-1: Running a function when the web page loads in the Chrome browser.

WARNING

There's some controversy as to just what the term *loads* means for the `onload` event. Some browsers trigger the `onload` event as the first thing before processing any of the HTML5 elements into the Document Object Model (DOM), while others wait until all the HTML5 elements have been processed before triggering the event. Because of this, it's not recommended to try to access any of the web page elements from a function triggered by the `onload` event — there's no guarantee that they'll be there yet.

You can test the `onload` event out in your own browsers by following these steps:

1. **Open your favorite text editor, program editor, or integrated development environment (IDE) package.**
2. **Type the following code into the editor window:**

```
<!DOCTYPE html>
<html>
<head>
<title>Testing the Page Events</title>
<script>
   function welcome() {
      alert("Welcome to my website!");
   }
</script>
</head>
<body onload="welcome()">
<h1>This is the main web page</h1>
<p>This is some content on the web page</p>
</body>
</html>
```

3. **Save the file as `loadtest.html` in the `DocumentRoot` folder of your web server.**

   For XAMPP on Windows, that's `c:\xampp\htdocs`; for XAMPP on macOS, that's `/Applications/XAMPP/htdocs`.
4. **Open the XAMPP Control Panel, and start the Apache web server.**
5. **Open your browser and enter the following URL:**

```
http://localhost:8080/loadtest.html
```

   You may need to change the TCP port to match your web server.

   You should see the welcome alert message, but you may or may not see the HTML code behind it on the web page.

6. **Try different browsers to see if they behave any differently.**

Figure 4-1 show the results from running the test using the Chrome browser. The `alert()` message appears from the `onload` event, but no content appears in the web page yet. Figure 4-2 shows running the same test using the Microsoft Edge browser.

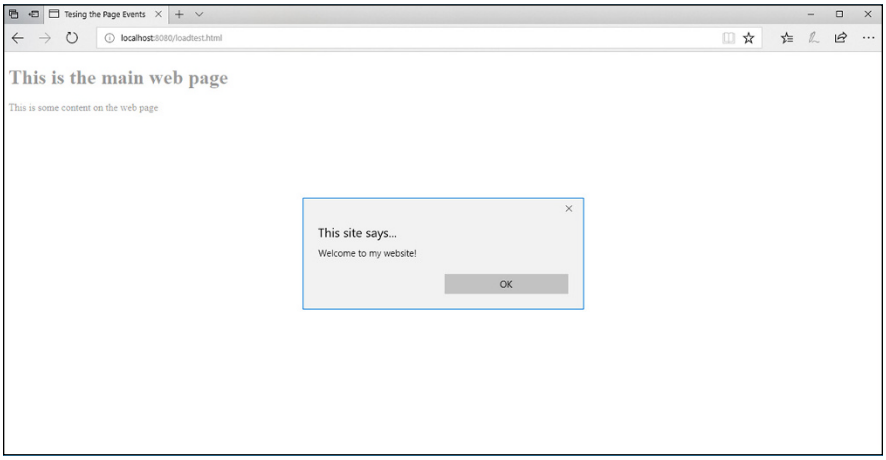

FIGURE 4-2: Running the `onload` test using the Microsoft Edge browser.

The Edge browser displays the elements on the web page and then triggers the `onload` event to run the `alert()` function!

Using the `onunload` and `onbeforeunload` events can be even more problematic. Most browsers won't allow you to use the `alert()` function after the browser window has already closed, so don't try to use that in the `onunload` event. Usually you can still access the DOM tree objects during the unload process, but even that's not guaranteed. It's common practice to only use the `onunload` and `onbeforeunload` events to trigger functions that ensure any application data is safely stored before the application closes out the web page.

Listening for mouse events

To trigger a JavaScript function for mouse events, you need to define the events as attributes in the HTML5 elements. This section shows you how to do that for a few different mouse events.

Clicking the button

When your website visitor clicks the primary mouse button anywhere on your web page, that triggers an `onclick` event. To capture that event for individual elements, you must use add the `onclick` attribute to the element opening tag and specify the JavaScript function you want the browser to run when the event triggers. For example:

```
<button onclick="myfunction()">
```

If you have more than one button on your web page, you can pass a parameter to the JavaScript function identifying which button was selected:

```
<button onclick="func('buy')">Buy</button>
<button onclick="func('browse')">Browse</button>
<button onclick="func('help')">Help</button>
```


TIP

Notice that to pass a string value inside the attribute value you must use single quotes around the string value if you use double quotes around the HTML attribute. If you use double quotes, the browser will confuse them with the double quotes used to delimit the attribute value.

Follow these steps to test out listening for button clicks:

1. **Open your favorite editor.**
2. **Type the following code:**

```
<!DOCTYPE html>
<html>
<head>
<title>Testing Button Events</title>
<script>
   function clickme(name) {
      if (name == "help") {
         alert("Do you need some help?");
      } else if (name == "buy") {
         alert("What would you like to buy?");
      } else if (name == "browse") {
         alert("You can browse our catalog");
      }
   }
</script>
</head>
<body>
```

```
<h1>Store Menu</h1>
<p>Here are the current options:</p>
<button onclick="clickme('buy')">Buy a product</button>
<button onclick="clickme('browse')">Browse our catalog</button>
<button onclick="clickme('help')">Get Help</button>
</body>
</html>
```

3. **Save the file as `buttontest.html` in the `DocumentRoot` folder of your web server.**
4. **Ensure that the Apache web server is still running.**
5. **Open your browser and enter the following URL:**

```
http://localhost:8080/buttontest.html
```

6. **Click each of the buttons that appears on the web page.**
7. **Close the browser window when you're done testing.**

As you click each button, a different alert dialog box should appear, as shown in Figure 4-3.

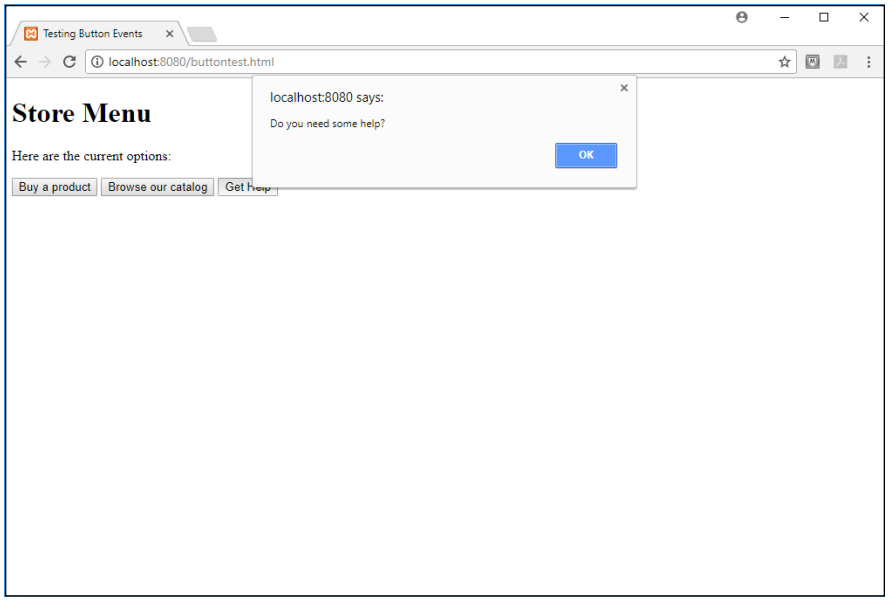

FIGURE 4-3: The Help alert dialog box appearing from the `buttontest.html` application.

If you prefer, you can also use a unique ID attribute for each button to help identify it in the event function code.

Hovering the pointer

It may seem odd, but the `onmouseover` and `onmouseout` events allow you to alter the appearance of many types of elements as your website visitors hover their mouse pointers over them. You're not limited to using these events on only buttons; you can work with the mouse events from inside any standard block element, such as paragraph and heading elements within your web page. Follow these steps to try that out:

1. **Open your editor.**
2. **Type the following code:**

```
<!DOCTYPE html>
<html>
<head>
<title>Testing Mouse Events</title>
<style>
   #test {
      background-color: yellow;
      width:400px;
   }
</style>
<script>
    function changeit(state) {
      if (state == "in") {
        document.getElementById("test").style.backgroundColor="red";
      } else if (state == "out") {
        document.getElementById("test").style.backgroundColor="yellow";
}
    }
</script>
</head>
<body>
<h1>This is a test of the mouse events</h1>
<p id="test" onmouseover="changeit('in')" onmouseout="changeit('out')">
   This is some content that will change color!</p>
</body>
</html>
```

3. **Save the file as** `hovertest.html` **in the** `DocumentRoot` **folder of your web server.**
4. **Ensure that the Apache web server is running.**
5. **Open your browser and enter the following URL:**

```
http://localhost:8080/hovertest.html
```

6. **Move your mouse pointer around through the text in the paragraph and observe what happens.**

   The background color of the p element text should change when your mouse pointer hovers over it.

7. **Close your browser window when you're done testing.**

The `onmouseover` event triggers the `changeit()` JavaScript function, passing the text `in`, while the `onmouseout` event triggers the same `changeit()` JavaScript function, but passes the text `out`. The JavaScript code detects the value passed to the `changeit()` function and sets the `background-color` style property of the p element accordingly.

Listening for keystrokes

Elements that accept data entry, such as text boxes and text areas, can trigger the keystroke events as your site visitors type. This allows you to monitor just what data your site visitors enter into the form fields as they type.

You'll often find yourself in situations where you need to count characters entered into a text box or text area in a form. You can use the `onkeyup` event to trigger a counter that counts the keystrokes.

Follow these steps to create a small program to demonstrate this feature using JavaScript and the `onkeyup` event:

1. **Open your favorite editor.**
2. **Type the following code:**

```
<!DOCTYPE html>
<html>
<head>
<title>Testing Keystroke Events</title>
```

```
<script>
    function gotkey() {
        var count =document.getElementById("text").value.length;
var output = "Character count: " + count;
        document.getElementById("status").innerHTML=output;
}
</script>
</head>
<body>
<h1>Testing for keystrokes</h1>
<p>Please enter some text into the text area</p>
<textarea id="text" cols="50" rows="20" onkeyup="gotkey()"></textarea><br>
<p id="status"></p>
</body>
</html>
```

3. **Save the file as `keytest.html` in the `DocumentRoot` folder of your web server.**
4. **Ensure that the Apache web server is running.**
5. **Open your browser and enter the following URL:**

```
http://localhost:8080/keytest.html
```

6. **Start typing some text in the text area that appears on the page.**

   You should see the character count appear under the text area and be able to keep track of the characters that appear.

7. **Close the browser to exit the program.**

The `gotkey()` function uses the `length` property of the `value` attribute of the element. By stringing them all together into the same statement, you can easily return the number of characters that are currently in the text area:

```
var count = document.getElementById("text").value.length;
```

The p element after the text area starts out empty, but for each triggering of the `gotkey()` function, it changes the `innerHTML` property to the string that was stored in the `output` variable. Figure 4-4 shows what the result will look like as you type text into the text area.

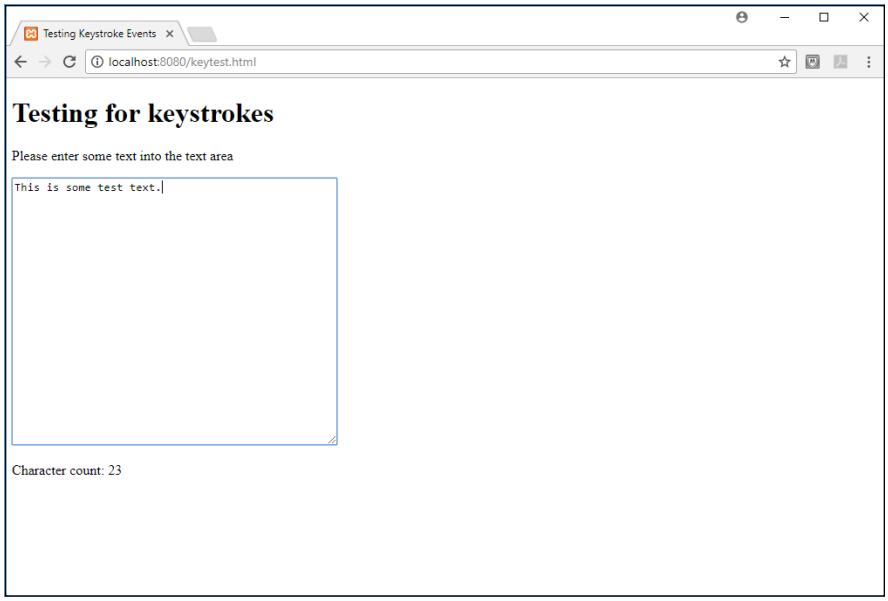

FIGURE 4-4: Counting keystrokes in the `keytest.html` program.

Now you can provide an interface that tells your site visitors how many characters they've typed into a text box or text area! You can take this feature one step further by disabling the text area if they've entered too many characters:

```
function gotkey() {
    var count = document.getElementById("text").value.length;
    if (count > 20) {
        var output = "Sorry, that's too many characters";
       document.getElementById("text").disabled="disabled";
    } else {
        var output = "Character count: " + count;
    }
    document.getElementById("status").innerHTML=output;
   }
```

Now things are really starting to get fancy!

Event listeners

JavaScript provides one more way to assign events to elements. You use the `.addEventListener()` function to dynamically assign events to monitor the elements on your web page. That looks like this:

```
document.getElementById("button1").addEventListener("click", clickbuy);
```

The first parameter of the `.addEventListener()` function defines the event to monitor (note the missing `on` as part of the event name). The second parameter specifies the function to call when the event is triggered. (Also note the missing parentheses in the function name.)

Just as you can dynamically add an event listener to an element, you can remove it using the `.removeEventListener()` function.

TIP

You can assign two or more functions to the same event trigger for an element. The JavaScript interpreter will trigger each function when the event occurs.

Looking at jQuery and Events

The jQuery library uses a slightly different approach to handling events. Instead of relying on the HTML5 event attributes in elements, it monitors the events in the browser and allows you to tap into them directly. This helps simplify things, because you don't need to split the event code between the HTML5 code and the jQuery code. Everything you need is in the jQuery code.

jQuery event functions

The jQuery library provides functions for handling all the HTML5 events that you've seen. The benefit of using the jQuery event model is that you don't need to specify the event attribute in the HTML5 code — the jQuery function does all the work for you!

For example, to monitor for the `onclick` event for a button, you just simply use the following:

```
$("button").click(function() {
    code
});
```

This creates an anonymous function to run whenever the site visitor clicks the button. The actual HTML5 button element would look like this:

```
<button>Click here</button>
```

And that's all you need! The benefit of this method is that you do all the event coding in the JavaScript code — there's nothing in the HTML5 code.

For the most part, the jQuery event functions mirror the HTML5 event attributes, but leave off the `on` part in the event name. There are, however, a couple of extra

handy event functions available. Table 4-3 shows a list of the jQuery events that you're most likely to use.

TABLE 4-3 The jQuery Event Functions

Event	Description
`blur()`	Triggers when the element loses the window focus
`change()`	Triggers when the element changes
`click()`	Triggers when the primary mouse button clicks on the element
`dblclick()`	Triggers when the primary mouse button is double-clicked on the event
`focus()`	Triggers when the element gains the window focus
`focusin()`	Triggers when the element or a child element gains the window focus
`focusout()`	Triggers when the element or a child element loses the window focus
`hover()`	Defines two functions — one for when the mouse pointer is over the element and another one for when it leaves
`keydown()`	Triggers when a key is held down
`keypress()`	Triggers when a key is pressed and released
`keyup()`	Triggers when a key is released
`mousedown()`	Triggers when the primary mouse button is held down
`mouseenter()`	Triggers when the mouse pointer enters the element area
`mouseleave()`	Triggers when the mouse pointer leaves the element area
`mousemove()`	Triggers when the mouse pointer moves
`mouseout()`	Triggers when the mouse pointer leaves the element area
`mouseover()`	Triggers when the mouse pointer is over the element area
`mouseup()`	Triggers when the primary mouse button is released
`ready()`	Triggers when the DOM tree is fully populated
`resize()`	Triggers when the browser window has been resized
`scroll()`	Triggers when the site visitor uses the scrollbar
`select()`	Triggers when an item is selected
`submit()`	Triggers when a submit button has been clicked

An extremely handy addition is the `hover()` function. It allows you to define two separate functions at the same time — one for when the mouse is hovering over the element and another for when it's not. Follow these steps to test this feature out.

1. **Open your favorite editor.**
2. **Type the following code into the editor window:**

```
<!DOCTYPE html>
<html>
<head>
<title>Testing Mouse Events</title>
<style>
    .yellow {
       background-color: yellow;
       width: 400px;
    }

    .red {
        background-color: red;
        width: 400px;
    }
</style>
<script src="jquery-3.2.1.min.js"></script>
<script>
    jQuery(document).ready( function() {
        $("p").hover( function() {
            $(this).addClass("red"); },
        function() {
            $(this).removeClass("red"); });
    });
</script>
</head>
<body>
<h1>This is a test of the mouse events</h1>
<p class="yellow">This is some content that will change color!</p>
<p>This is some content that will change color, too!</p>
</body>
</html>
```

3. **Save the file as `jhovertest.html` in the `DocumentRoot` folder for your web server.**

4. **Ensure that the Apache web server is running.**

5. **Open your browser and enter the following URL:**

```
http://localhost:8080/jhovertest.html
```

6. **Move the mouse pointer around to hover over the p element sections and watch what happens.**

   Each p element should get the red background only when you hover over it; the other p element should stay the same.

7. **Close out the browser to end the test.**

In the code for this example, everything happens in the jQuery code:

```
jQuery(document).ready( function() {
    $("p").hover( function() {
        $(this).addClass("red"); },
    function() {
        $(this).removeClass("red"); });
});
```

You should recognize the first line, which tells jQuery to wait until the browser loads the document before running the function code. The function code selects all p elements and then assigns the `hover()` event function to them. In this example, I created two p elements to show another neat feature in jQuery.

When you hover over each p element, only that p element changes background color! The key to that is the `$(this)` object in jQuery. The `$(this)` object represents the currently selected object. Using that, whichever p element triggered the event is the one that the `addClass()` function applies to, while the other p element is ignored. That saves us a whole lot of code from having to uniquely identify each p element on the web page! Figure 4-5 shows the result of the program in action.

This example shows just how easy it is to code events with jQuery. One of the primary goals of jQuery is to make coding for handling events easier, and I'd say they met their goals!

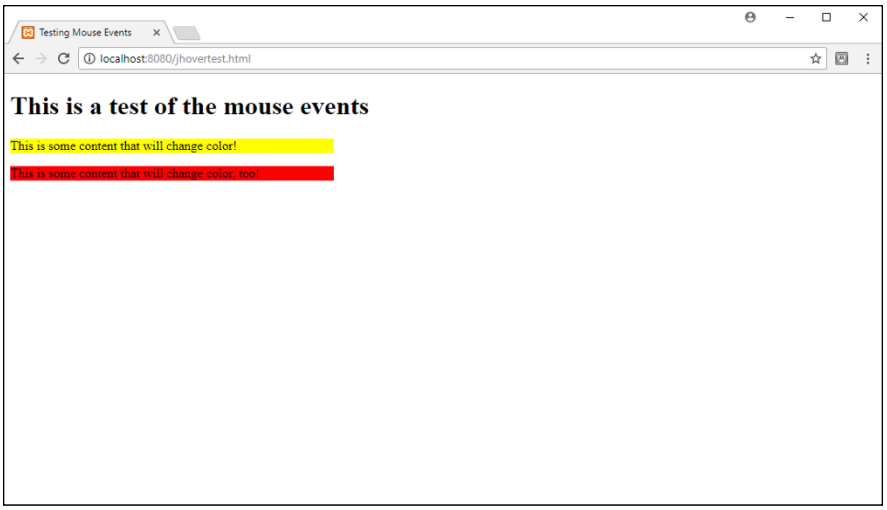

FIGURE 4-5: The `jhover.html` code test only changes one p element at a time.

The jQuery event handler

The jQuery library also provides a way for you to code event handlers. With jQuery, the event handler function is called `on()`. Here's the format for the `on()` function:

```
$(selector).on("event", "filter", data, function() {
   code
});
```

The *`selector`* part you should be familiar with now. It determines which element(s) the event handler is attached to. The `event` parameter defines the jQuery event to attach to the element(s). The *`filter`* parameter is a little different. It defines a child selector to the main selector you specify. For example, if you only want to capture click events on buttons within an article element section, you'd use the following:

```
$("article").on("click", "button", function() {
```

To test this feature out, follow these steps to convert the `keytest.html` JavaScript code you worked on earlier to use jQuery instead:

1. **Open your favorite editor.**
2. **Type the following code into the editor window:**

```
<!DOCTYPE html>
<html>
<head>
<title>Testing jQuery Keystroke Events</title>
```

```
<script src="jquery-3.2.1.min.js"></script>
<script>
    jQuery(document).ready( function() {
        $("textarea").on("keyup", function() {
            var count = $(this).val().length;
            var output = "Character count: " + count;
            $("#status").text(output);
        });
    });
</script>
</head>
<body>
<h1>Testing for keystrokes</h1>
<p>Please enter some text into the text area</p>
<textarea cols="50" rows="20"></textarea><br>
<p id="status"></p>
</body>
</html>
```

3. **Save the file as** `jkeytest.html` **in the** `DocumentRoot` **folder of your web server.**
4. **Ensure that the web server is running.**
5. **Open your browser and enter the following URL:**

```
http://localhost:8080/jkeytest.html
```

6. **Start typing text in the text area.**

   You should see the count message appear in the status area, showing the accurate count of how many characters are in the text area.

7. **Close the browser window when you're finished.**
8. **Stop the web server.**

One thing you have to say about jQuery code: It's a lot cleaner looking than the JavaScript version! Notice that now you don't need to define an event attribute in the `<textarea>` tag. jQuery takes care of that for you.

The jQuery code itself is fairly clean and uncomplicated:

```
jQuery(document).ready( function() {
    $("textarea").on("keyup", function() {
        var count = $(this).val().length;
        var output = "Character count: " + count;
        $("#status").text(output);
    });
});
```

It starts out as usual, waiting for the document DOM to load and then assigns the event handler to the text area element on the web page. The event handler looks for the `keyup` event; when it's detected, the handler function retrieves the length of the text in the text area (again, using the `$(this)` selector) and then outputs it to the status p element area. Figure 4-6 shows how this looks.

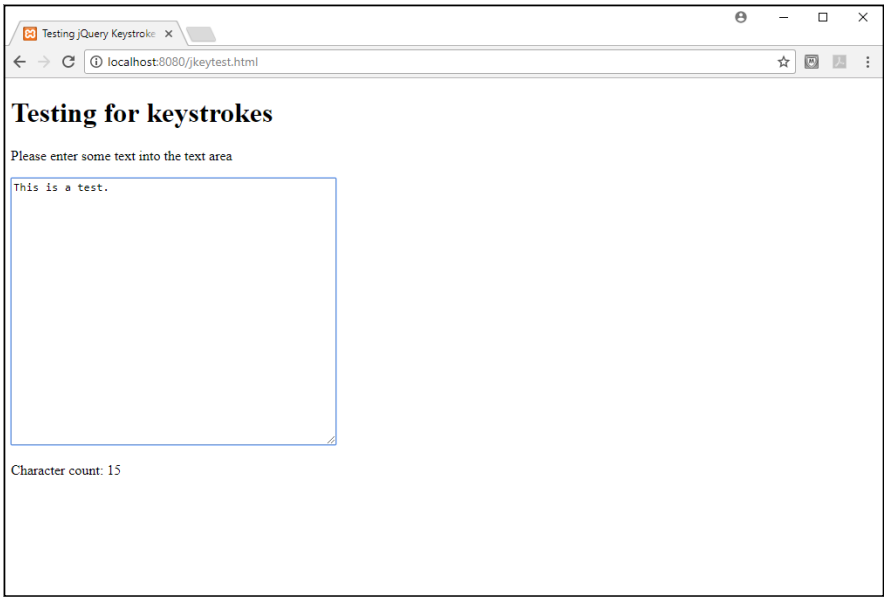

FIGURE 4-6: The output of the `jkeytest.html` program in action.

The results are the same as the JavaScript version, but with a lot less coding!

TIP

To define an event handler that only triggers once and goes away, use the `one()` function instead of the `on()` function. To remove an event handler that you've defined for a selector, use the `off()` function.