IN THIS CHAPTER

- Understanding how simple web pages work
- Incorporating programming into your web page
- Storing content in a database

Chapter 1

Examining the Pieces of Web Programming

At first, diving into web programming can be somewhat overwhelming. You need to know all kinds of things in order to build a web application that not only looks enticing but also works correctly. The trick to learning web programming is to pull the individual pieces apart and tackle them one at a time.

This chapter gets you started on your web design journey by examining the different pieces involved in creating a simple web page. Then it kicks things up a notch and walks you through dynamic web pages. And finally, the chapter ends by explaining how to store your content for use on the web.

Creating a Simple Web Page

Before you can run a marathon, you need to learn how to walk. Likewise, before you can create a fancy website, you need to know the basics of how web pages work.

Nowadays, sharing documents on the Internet is easy, but it wasn't always that way. Back in the early days of the Internet, documents were often created using proprietary word-processing packages and had to be downloaded using the

cumbersome File Transfer Protocol (FTP). To retrieve a document, you had to know exactly what server contained the document, you had to know where it was stored on the server, and you had to be able to log into the server. After all that, you *still* needed to have the correct word-processing software on your computer to view the document. As you can imagine, it wasn't long before a new way of sharing content was required.

To get to where we are today, several different technologies had to be developed:

- A method for linking related documents together
- A way for the document reader to display formatted text the same way in any type of device
- An Internet standard allowing clients to easily retrieve documents from any server
- A standard method of styling and positioning content in documents

This section describes the technology that made viewing documents on the Internet work the way it does today.

Kicking things off with the World Wide Web

In 1989, Tim Berners-Lee developed a method of interconnecting documents to make sharing research information on the Internet easier. His creation, the *World Wide Web,* defined a method for linking documents together in a web structure, so that a researcher could follow the path between related documents, no matter where they were located in the world. Clicking text in one document took you to another document automatically, without your having to manually find and download the related document.

The method Berners-Lee developed for linking documents is called *hypertext.* Hypertext embeds links that are hidden from view in the document and directs the software being used to view the document (known as the *web browser*) to retrieve the referenced document. With hypertext, you just click the link, and the software (the web browser) does all the work of finding and retrieving the related document for you.

Because the document-viewing software does all the hard work, a new type of software had to be developed that was more than just a document viewer. That's where web browsers came into existence. Web browsers display a document on a computer screen and respond to the reader clicking hypertext links to retrieve other specified documents.

To implement hypertext in documents, Berners-Lee had to utilize a text-based document-formatting system. Fortunately for him, a lot of work had already been done on that.

Making sense of markup languages

Markup languages were developed to replace proprietary word-processing packages with a standard way of formatting documents so that they could be read by any type of document viewer on any type of device. This goal is accomplished by embedding *tags* in the text. Each tag indicates a formatting feature, such as headings, bold or italic text, or special margins. What made markup languages different from word-processing packages is that these tags were common text codes instead of proprietary codes, making it generic enough that any device could read and process them.

The first popular markup language was the Generalized Markup Language (GML), developed by IBM in the 1960s. The International Organization for Standardization (ISO) took up the challenge of creating markup languages and produced the Standard Generalized Markup Language (SGML), mainly based on GML, in the 1980s. However, because SGML was developed to cover all types of document formatting on all types of devices, it's extremely complex and it wasn't readily adapted.

Berners-Lee used the ideas developed in SGML to create a simplified markup language that could support his hypertext idea. He called it *Hypertext Markup Language* (HTML). HTML uses the same concept of tags that SGML uses, but it defines fewer of them, making it easier to implement in software.

An example of an HTML tag is `<h1>`. You use this tag to define text that's used as a page heading. Just surround the text with an opening `<h1>` tag, and a corresponding closing `</h1>` tag, like this:

```
<h1>This is my heading</h1>
```

When the browser gets to the `<h1>` tag, it knows to format the text embedded in the opening and closing tags using a different style of formatting, such as a larger font or a bold typeface.

To define a hypertext link to another document, you use the `<a>` tag:

```
<a href="anotherdoc.html">Click here for more info</a>
```

When the reader clicks the *Click here for more info* text, the browser automatically tries to retrieve the document specified in the `<a>` tag. That document can be on the same server or on another server anywhere on the Internet.

HTML development has seen quite a few changes since Berners-Lee created it and turned it over to the World Wide Web Consortium (W3C) to maintain. After many years of faithfully maintaining the HTML standard, unfortunately, it had met with some controversy, as a competing standard, maintained by the Web Hypertext Application Technology Working Group (WHATWG), a consortium of several vendors, emerged. Table 1-1 shows the path the HTML standard has taken.

TABLE 1-1 HTML Versions

Version	Description
HTML 1.0	Formally released in 1989 as the first public version of HTML
HTML 2.0	Released in 1995 to add interactive elements
HTML 3.0	Released in 1996 but never widely adopted
HTML 3.2	Released in 1997, adding support for tables
HTML 4.01	Released in 1999, widely adopted, and remains an often-used standard
XHTML 1.0	Released in 2001, standardizing HTML around the XML document format
XHTML 1.1	Released in 2002, making updates and corrections to XHTML 1.1
HTML 5.0	Released in 2014, adding multimedia features
HTML 5.1	Released in mid-2017, adding form validation and context menus
HTML 5.2	Released in late-2017, adding additional styling features
HTML 5.3	Also released in late-2017, this was the final version released by the W3C

In 2019, the W3C stopped as the sole maintainer of the official HTML standard and joined with the WHATWG consortium to produce a single HTML standard, called the *HTML Living Standard.* This is now considered the official HTML standard, and the standard that this book focuses on. The Living Standard doesn't have specific release versions, but instead, incorporates changes "on the fly" to the HTML specifications once they are approved by their board. You can find the latest HTML features described at the WHATWG website, `html.spec.whatwg.org/multipage/`. The WHATWG documentation refers to the term HTML5 as a buzzword, often used to describe the modern HTML standard.

Retrieving HTML documents

Besides a document-formatting standard, Berners-Lee also developed a method of easily retrieving the HTML documents in a client–server environment. A *web server* software package runs in the background on a server, listening for connection requests from *web clients* (the browser). The browser sends requests to retrieve HTML documents from the server. The request can be sent anonymously (without using a login username), or the browser can send a username and password or certificate to identify the requestor.

These requests and responses are defined in the *Hypertext Transfer Protocol* (HTTP) standard. HTTP defines a set of requests the client can send to the server and a set of responses the server uses to reply back to the client.

This section walks you through the basics of how web servers and web clients use HTTP to interact with each other to move web pages across the Internet.

Web clients

The web client sends requests to the web server on a standard network communication channel. The requests use a common Internet protocol known as Transmission Control Protocol (TCP). Each TCP connection uses a defined port number to communicate between the server application and clients. HTTP uses TCP port 80 as the standard for communications between servers and clients. HTTP uses standard text requests sent to the server, either requesting information from the server or sending information to the server. Table 1-2 shows the basic HTTP client requests available.

TABLE 1-2 HTTP Client Requests

Request	Description
`CONNECT`	Converts the connection into a secure tunnel for sending data
`DELETE`	Deletes the specified resource
`GET`	Requests the specified resource
`HEAD`	Requests the title of the specified resource
`OPTIONS`	Retrieves the HTTP requests that the server supports
`PATCH`	Applies a modification to a resource
`POST`	Sends specified data to the server for processing
`PUT`	Stores specified data at a specified location
`TRACE`	Sends the received request back to the client

As shown in Table 1-2, when you ask to view a web page from your client browser, the browser sends the HTTP `GET` request to the server, specifying the filename of the web page. The server then responds with a response code along with the requested data. If the client doesn't specify a filename in the `GET` request, most servers have a default file with which to respond.

Web servers

With HTTP, the web server must respond to each client request received. If the client sends a request that the server can't process, the server must send some type of error code back to the client indicating that something went wrong.

The first part of the server response is a status code and text that the client uses to determine whether the submitted request was successful. The format of the HTTP response uses a three-digit status code, followed by an optional text message that the browser can display. The three-digit codes are broken down into five categories:

- **1xx:** Informational messages
- **2xx:** Success
- **3xx:** Redirection
- **4xx:** Client error
- **5xx:** Server error

The three-digit status code is crucial to knowing what happened with the response. Many status codes are defined in the HTTP standards, providing some basic information on the status of client requests. Table 1-3 shows just a few of the standard HTTP response codes that you may run into.

As you can see from Table 1-3, a web server can return many possible responses. It's the client's job to parse the response and determine the next action to take.

If the response indicates the request was successful, the server will follow the response code with the data related to the request, such as the contents of an HTML file. The client must then read the returned data and decide what to do with it. For HTML files, the browser will display the requested file, applying the HTML formatting tags to the data.

TIP

Don't worry about trying to memorize all the HTTP status codes. Most of them you'll never run into in your web-programming career. Before long, you'll start to remember a few of the more common ones, and you can always look up any others you run into.

TABLE 1-3 Common HTTP Server Response Status Codes

Status Code	Text Message	Description
100	Continue	The client should send additional information.
101	Switching Protocols	The server is using a different protocol for the request.
102	Processing	The server is working on the response.
200	OK	The server accepted the request and has returned the response.
201	Created	The server created a new resource in response to the request.
202	Accepted	The data sent by the client has been accepted by the server but has not completed processing the data.
206	Partial Content	The response returned by the server is only part of the full data; more will come in another response.
300	Multiple Choices	The request matched multiple possible responses from the server.
301	Moved Permanently	The requested file was moved and is no longer at the requested location.
302	Found	The requested resource was found at a different location.
303	See Other	The requested resource is available at a different location.
304	Not Modified	The requested resource was not modified since the last time the client accessed it.
307	Temporary Redirect	The requested resource was temporarily moved to a different location.
308	Permanent Redirect	The requested resource was permanently moved to a different location.
400	Bad Request	The server cannot process the request.
401	Unauthorized	The resource requires authentication that the client did not provide.
402	Payment Required	The requested resource is not freely available.
403	Forbidden	The resource requires authentication, and the client does not have the proper permission.
404	Not Found	The requested resource was not located on the server.
414	URI Too Long	The Uniform Resource Identifier (URI) describing the location of the resource was longer than the server is able to handle.

(continued)

TABLE 1-3 *(continued)*

Status Code	Text Message	Description
415	Unsupported Media Type	The server does not know how to process the requested resource file.
429	Too Many Requests	The client has sent too many requests within a specific amount of time.
500	Internal Server Error	An unexpected condition occurred on the server while trying to retrieve the requested resource.
501	Not Implemented	The server doesn't recognize the request.
502	Bad Gateway	The server was acting as a proxy to another server but received an invalid response from the other server.
503	Service Unavailable	The server is currently unavailable, often due to maintenance.
505	HTTP Version Not Supported	The server doesn't support the HTTP standard used by the client in the request.
507	Insufficient Storage	The server is unable to store the resource due to lack of storage space.
511	Network Authentication Required	The client is required to authenticate with a network resource to receive the response.

Styling your web content

The HTML standard defines how browsers perform basic formatting of text, but it doesn't really provide a way to tell a browser how to display the text. The `<h1>` tag indicates that the text should be a heading, but nothing tells the browser just how to display the heading to make it different from any other text on the page.

This is where styling comes into play. *Styling* allows you to tell the browser just what fonts, sizes, and colors to use for text, as well as how to position the text in the display. This section explains how styling affects how your web pages appear to your visitors.

Style sheets

There are several ways to define styling for an HTML document. The most basic method is what the browser uses by default. When the browser sees an HTML formatting tag, such as the `<h1>` tag, it has a predefined font, size, and color that the developer of the browser felt was useful.

That's fine, but what if you want to make some headings black and others red? This is possible with *inline styling*. Inline styling allows you to define special styles that apply to only one specific tag in the document. For example, to make one heading red, you'd use the following HTML:

```
<h1 style="color: red">Warning, this is bad</h1>
```

The `style` term is called an *attribute* of the `<h1>` tag. There are a few different attributes you can apply directly to tags within HTML; each one modifies how the browser should handle the tag. The `style` attribute allows you to apply any type of styling to this specific `<h1>` tag in the document. In this example, I chose to change the color of the text.

Now, you're probably thinking that I've just opened another can of worms. What if you want to apply the red color to *all* the <h1> tags in your document? That's a lot of extra code to write! Don't worry, there's a solution for that.

Instead of inserting styles inline, you can create a style definition that applies to the entire document. This method is known as *internal styling.* It defines a set of styles at the top of the HTML document that are applied to the entire document. Internal styling looks like this:

```
<style>
h1 {color: red;}
</style>
```

Now the browser will display all the `<h1>` tags in the document using a red color. But wait, there's more!

Style listings can be somewhat lengthy for large web pages, and placing them at the top of a document can become cumbersome. Also, if you want to apply the same styles to all the web pages in a website, having to retype or copy all that text can be tiring. To solve that problem, you use an external style sheet.

An *external style sheet* allows you to define styles just as the internal method does, but in a separate file, called a *style sheet.* Any web page can reference the same style sheet, and you can apply multiple style sheets to a single web page. You reference the external style sheet using the `<link>` tag, like this:

```
<link rel="stylesheet" href="mystyles.css">
```

When the browser sees this tag, it downloads the external style sheet called `mystyles.css` and applies the defined styles to the document.

This all sounds great, but things just got a lot more complicated! Now there are three different locations from which you can define styles for your HTML document, on top of what the browser itself does. How are you supposed to know which ones take precedence over the others?

The *Cascading Style Sheet* (CSS) standard defines a set of rules that determine just how browsers should apply styles to an HTML document. As the name implies, styles cascade down from a high level to a low level. Styles defined in a higher-level rule override styles defined in a lower-level rule.

The CSS standard defines nine separate levels, which I cover in greater detail in Book 2, Chapter 2, but for now, here are the four most common style levels, in order from highest priority to lowest:

- Styles defined within the element tags
- Styles defined in an internal style sheet
- Styles defined in an external style sheet
- Styles defined by the client's browser defaults

So, any style attributes you set in an element tag override any styles that you set in an internal style sheet, which override any styles you set in an external style sheet, which override any styles the client browser uses by default. This allows you to set an overall style for your web pages using an external style sheet, and then override those settings for individual situations using the standard element tags.


TECHNICAL STUFF

You may be wondering how assistive technology tools work to change the web page display for individuals who are sight impaired. Part of the nine rules that I cover in Book 2, Chapter 2, incorporate any rules defined in the browser for sight-impaired viewing.

CSS standards

The CSS standard defines a core set of styles for basic rendering of an HTML document. The first version of CSS (called CSS1) was released in 1996, and it only defined some very rudimentary styles:

- Font type, size, and color
- Text alignment (such as margins)
- Background colors or images
- Borders

The second version of CSS, called — you guessed it! — CSS2 was released in 1998. It added only a few more styling features:

- More exact positioning of text
- Styles for different output types (such as printers or screens)
- The appearance of browser features such as the cursor and scrollbar

That's still not all that impressive of a list of styles. Needless to say, more was needed to help liven up web pages. To compensate for that, many browser developers started creating their own style definitions, apart from the CSS standards. These style definitions are called *extensions.* The browser extensions covered lots of different fancy styling features, such as applying rounded edges to borders and images, making a smoother layout in the web page.

As you might guess, having different extensions to apply different style features in different browsers just made things more complicated. Instead of coding a single style for an element in an HTML document, you needed to code the same feature several different ways so the web page would look the same in different browsers. This quickly became a nightmare.

When work was started on the CSS3 standard in 1999, one of the topics was to rein in the myriad browser extensions. However, things quickly became complicated because all the different browser developers wanted their own extensions included in the new standard.

To simplify the process, the CSS design committee split the CSS standards into separate modules. Each CSS module covers a specific area of styling, such as colors, media support, and backgrounds. Each module could be voted on and released under a different timeline. The downside to this approach is that now each module has been released as a recommended standard at a different time, making the CSS3 standard somewhat difficult to track and implement.

Quite possibly one of the most anticipated features of CSS3 was the ability to define fonts. Fonts had long been the bane of web programmers. When you define a specific font, that font must be installed on your website visitor's computer in order for the browser to use it. If the font isn't available, the browser picks a default font to use, which often becomes an ugly mess.

Web fonts allow you to define a font on your server so that every client browser can download the font and render text using it. This was a huge accomplishment! No longer were you reliant on your website visitors having specific fonts installed in their web browsers.

Yet another popular feature of CSS3 is the use of shadows and semitransparent colors in text and other web page elements, such as form objects. These features by themselves can transform an ugly HTML form into a masterpiece.

The combination of HTML5 and CSS3 greatly revolutionized the web world, allowing developers to create some pretty amazing websites. However, one thing was still missing: the ability to easily change content on the web page.

Creating a Dynamic Web Page

Static web pages contain information that doesn't change until the web designer or programmer manually changes it. In the early days of the Internet, simply jumping on the Internet bandwagon was important for corporations. It wasn't so important what companies posted on the web, as long as they had an Internet presence where customers could get basic information about the company and its products. Static web pages, consisting solely of HTML and CSS, easily accomplished this function.

But one of the big limitations of static web pages is how much effort it takes to update them. Changing a single element on a static web page requires rebuilding and reloading the entire page, or sometimes even a group of web pages. This process is way too cumbersome for an organization that frequently needs to post real-time information, such as events, awards, or closings. Also, during this process, a developer can accidentally change other items on the page, seriously messing up the information on the web page, or even the entire web page layout!

Dynamic web pages allow you to easily change your content in real time without even touching the coding of the page. That's right: Without manually making any changes to the page itself, the information on the page can change. This means you can keep the content on the page fresh so that what a visitor sees there now may be updated or replaced in a day, an hour, or a minute. The core layout of the web page can remain the same, but the data presented constantly changes.

To successfully create a dynamic web page, you have to know a method for automatically inserting real-time data into the HTML code that gets sent to the client browser. This is where web scripting languages come in.

A *web scripting language* allows you to insert program code inside your web page that dynamically generates HTML that the client browser reads. A processor reads the program code and dynamically generates HTML to display content on the web page, as shown in Figure 1-1.

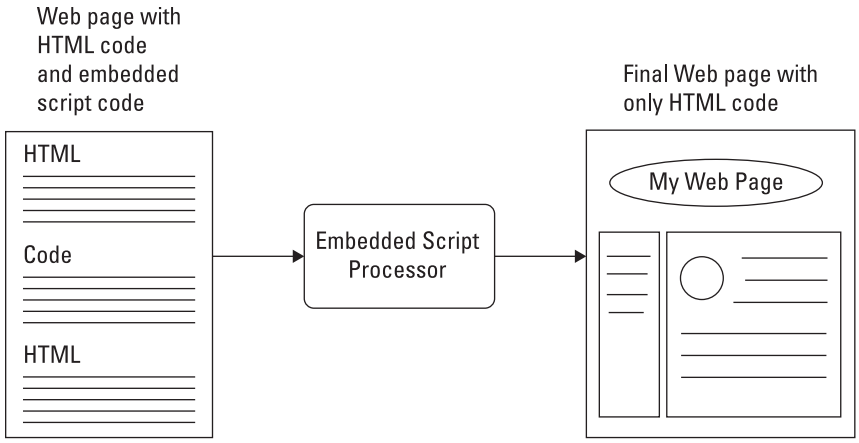

FIGURE 1-1: Program code embedded in a web page.

Now, because programming code is embedded in the web page, something somewhere must run the code to produce the dynamic HTML for the new content. As it turns out, there are two places where the embedded program code can run:

- On the client's computer, after the web browser downloads the web page. This is known as *client-side programming.*
- On the web server before the web page is sent. This is known as *server-side programming.*

This section takes a look at how each of these types of programming differ in creating dynamic content for your website.

Client-side programming

In client-side programming, you embed program code inside the HTML code that the server sends to the client browser with the HTML code. The browser must be able to detect the embedded program code and run it, either inside the browser or as a separate program outside the browser. Figure 1-2 demonstrates this process.

JavaScript

These days, the most popular client-side programming language is JavaScript. JavaScript is a scripting language that you embed inside the normal HTML code in your web page. It runs within the client browser and can utilize features of the browser that are not normally accessible from standard HTML code. JavaScript code is commonly used to produce pop-up messages and dialog boxes that people interact with as they view the page. These are elements that HTML code can't generate.

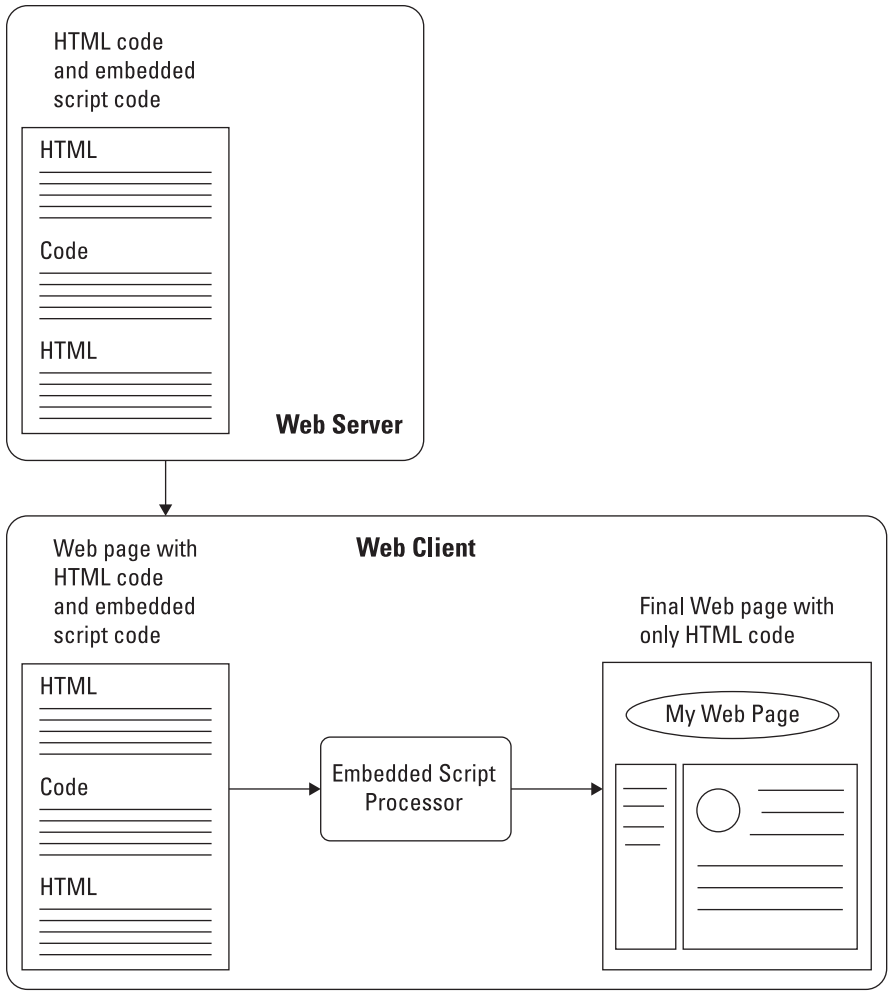

FIGURE 1-2: Using client-side code in a web page.

As shown in Figure 1-2, the entire web page with the embedded JavaScript code is downloaded to the client browser. The client browser detects the embedded JavaScript code and runs it accordingly. It does this while also processing the HTML tags within the document and applying any CSS styles defined. That's a lot for the browser to keep up with!

The downside of JavaScript is that, because it runs in the client browser, you're at the mercy of how the individual web browser interprets the code. Although the HTML language started out as a standard, JavaScript was a little different. In the early days of JavaScript, different browsers would implement different features of JavaScript using different methods. It was not uncommon to run across a web page that worked just fine for one type of browser, but didn't work at all in another type of browser — all because of JavaScript processing inconsistencies.

Eventually, work was done to standardize JavaScript. The JavaScript language was taken up by the Ecma International standards organization, which created the ECMAScript standard, which is what JavaScript is now based on. As the ECMAScript standard evolved, more and more browser developers started seeing the benefits of using a standard client-side programming language and incorporated them in their JavaScript implementations. At the time of this writing, the 15th version of the standard, called ECMAScript 2024, has been finalized and implemented in most browsers.


TECHNICAL STUFF

The name JavaScript was chosen to capitalize on the popularity of the Java programming language for use in web applications. However, it doesn't have any resemblance or relation to the Java programming language.

JavaScript libraries

JavaScript is popular, but one of its downsides is that it can be somewhat complicated to program. With so many different features incorporated by so many different developers, today a JavaScript program can quickly turn into a large endeavor to code.

To help solve this issue, groups of developers banded together to create various sets of libraries to make client-side programming with JavaScript easier. These days, there are several popular JavaScript libraries available, with possibly the most popular being jQuery.

The jQuery libraries are self-contained JavaScript functions that you can reference in your own JavaScript programming to perform common functions, such as finding a location in a web page to display text or retrieve a value entered into an HTML form field.

Instead of having to write lines and lines of JavaScript code, you can just reference one or two jQuery functions to do the work for you. That's a huge time-saver, as well as a great resource for implementing advanced features that you would never have been able to code yourself using just JavaScript.

Server-side programming

The other side of web programming is server-side programming. Server-side programming languages solve the problem of different client code interpreters by running the code on the server. In server-side programming, the web server interprets the embedded programming code before sending the web page to the client's browser. The server then takes any HTML that the programming code

generates and inserts it directly into the web page before sending it out to the client. The server does all the work running the scripting code, so you're guaranteed that every web page will run properly. Figure 1-3 illustrates this process.

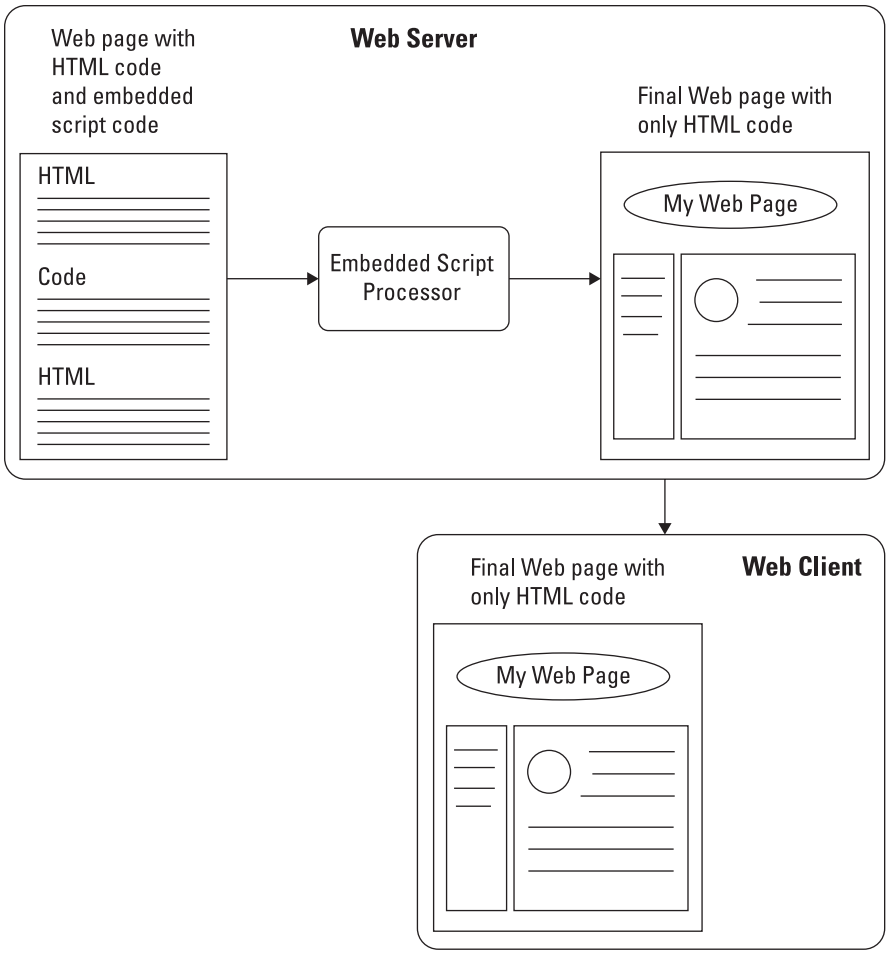

FIGURE 1-3: Using server-side programming to create a web page.

While JavaScript has risen to be the most popular client-side programming language, there are many popular server-side programming languages that are in use these days, each with its own set of pros and cons. This section takes a look at a few of the more popular programming languages.

CGI scripting

One of the first attempts at server-side programming support was the Apache web server's Common Gateway Interface (CGI). The CGI provided an interface between the web server and the underlying server operating system (OS), which was often UNIX-based.

This allowed programmers to embed scripting code commonly used in the UNIX platform to dynamically generate HTML. Two of the most common scripting languages used in the UNIX world and, thus, commonly used in CGI programming are Perl and Python.

Although CGI programming became popular in the early days of the web, it wasn't long before it was exploited. It was all too easy for a novice administrator to apply the wrong permissions to CGI scripts, allowing a resourceful attacker to gain privileged access to the server. Other methods of processing server-side programming code had to be developed.

Java

One of the other early attempts at a controlled server-side programming language was Java. Although the Java programming language became popular as a language for creating standalone applications that could run on any computer platform, it can also run as a server-side programming language in web applications. When used this way, it's called Jakarta Server Pages (JSP).

The JSP language requires that you have a Java compiler embedded with your web server. The web server detects the Java code in the HTML code and then sends the code to the Java compiler for processing. Any output from the Java program is sent to the client browser as part of the HTML document. The most common JSP platform is the open-source Apache Tomcat server.

The Microsoft ASP.NET family

Microsoft's first entry into the server-side programming world — Active Server Pages (ASP) — had a similar look and feel to JSP. ASP programs embedded ASP scripting code inside standard HTML code and required an ASP server to be incorporated with the standard Microsoft Internet Information Services (IIS) web server to process the code.

However, Microsoft developers determined that it wasn't necessary to maintain a separate programming language for server-side web programming, so they combined the server-side programming and Windows desktop programming environments into one technology. With the advent of the .NET family of programming

languages, Microsoft released ASP.NET for the web environment, as an update to the old ASP environment.

With ASP.NET, you can embed any type of Microsoft .NET programming code inside your HTML documents to produce dynamic content. The .NET family of programming languages includes Visual Basic .NET, C#, J#, and even Delphi. NET. This allows you to leverage the same code you use to create Windows desktop applications as you do to create dynamic web pages. You can often use the same Windows features, such as buttons, slide bars, and scrollbars, inside your web applications that you see in Windows applications.

JavaScript

Yes, you read that right. The same JavaScript language that's popular in the client-side programming world is now also popular as a server-side programming language. The Node.js library allows you to interface JavaScript code inside HTML web pages for processing on the server.

The benefit to using Node.js is that you only need to learn one language for both client-side and server-side programming.

PHP

What started out as a simple exercise in tweaking CGI scripts turned into a new server-side programming language that took the world by storm. Rasmus Lerdorf wrote the Personal Home Page (PHP) programming language as a way to improve how his CGI scripts worked. After some encouragement and help, PHP morphed into its own programming language, and a new name, PHP: Hypertext Preprocessor (yes, it uses the acronym inside its name, which is called a *recursive acronym*).

The PHP language developers freely admit that they borrowed many features from other popular languages, such as Perl, Python, C, and even UNIX shell scripting. However, PHP was developed specifically for server-side programming, and it has many features built in that aren't available in other scripting languages. You don't need to wrestle with strange setups or features to get PHP to work in a web environment. It has matured into a complete catalog of advanced features that cover everything from database access to drawing graphics on your web page.

Because of the dedication of the PHP developers to create a first-rate server-side programming language, and because it's free open-source software, PHP quickly became the darling of the Internet world. Many web-hosting companies include PHP as part of their basic hosting packages. If you already have space on a web-hosting server, it's possible that you already have access to PHP!

Combining client-side and server-side programming

Client-side and server-side programming both have pros and cons. Instead of trying to choose one method of creating dynamic web pages, you can instead use both at the same time!

You can easily embed both client-side and server-side programming code into the same web page to run on the server, as shown in Figure 1-4.

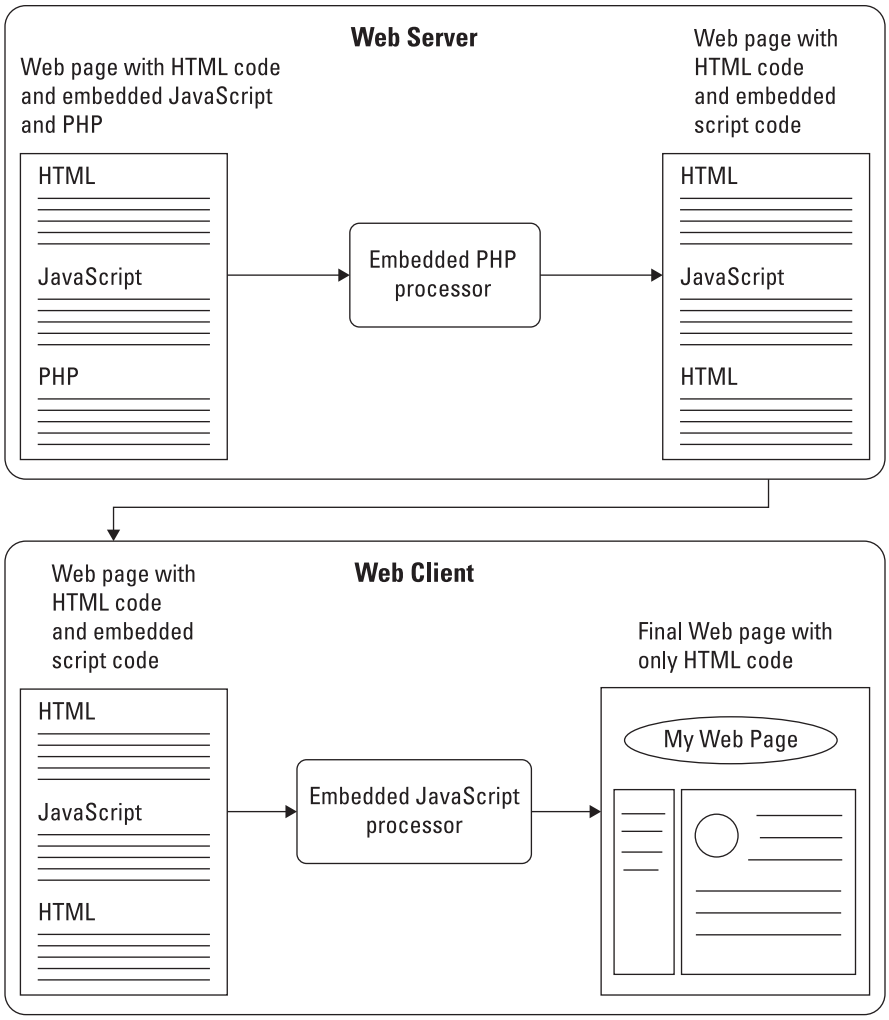

FIGURE 1-4: Combining client-side and server-side programming.

One common use for JavaScript and PHP coding is data validation. When you provide an HTML form for your website visitors to fill out, you have to be careful that they actually fill in the correct type of data for each field. With server-side programming, you can't validate the data until the site visitor completes and submits the form to the server. If a website visitor accidentally skips filling out a single field and the entire form needs to be filled out all over again, that can be a frustrating experience for the user, as well as an inefficient use of server and network resources.

To solve that problem, you can embed JavaScript code into the form to check as the site visitor enters data into the form. If any form fields are empty when the Submit button is clicked, the JavaScript code can block the form submission and point out the empty field. Then, when all the data is completed and the form is successfully submitted, the PHP code on the server can process the data to ensure it's the correct data type and format. Book 2, Chapter 3 covers forms in lots more detail.

Storing Content

The last piece of the dynamic web application puzzle is the actual content. With static web pages, content is already built into the web page code. To change information on a static web page, you have to recode the page. Unfortunately, more often than not, when a web page is updated, the old version is lost.

With dynamic web applications, the content comes from somewhere outside of the web page. But where? The most common place is a database, located either on the local server or in a cloud network.

Databases are an easy way to store and retrieve data. They're quicker than storing data using standard files, and they provide a level of security to protect your data. By storing content in a database, you can also easily archive and reference old content and replace it with new content as needed.

Much like the server-side programming world, the database world has lots of different database software options. Here are some of the more popular ones:

- **Oracle:** Oracle has set the gold standard for databases. It's found in many high-profile commercial environments. Although Oracle is very fast and supports lots of features, it can also be somewhat expensive.

- **Microsoft SQL Server:** Microsoft's entry into the database server world, SQL Server, is geared toward high-end database environments. It's often found in environments that utilize Microsoft Windows Servers. Microsoft also provides the SQL Server Express package as a free stripped-down version of the full SQL Server package for personal use.
- **PostgreSQL:** The PostgreSQL database server is an open-source project that attempts to implement many of the advanced features found in commercial databases. In its early days, PostgreSQL had a reputation for being somewhat slow, but it has made vast improvements. Unfortunately, old reputations are hard to shake, and PostgreSQL still struggles with overcoming them.
- **MySQL:** The MySQL database server is yet another open-source project. Unlike PostgreSQL, it doesn't attempt to match all the features of commercial packages. Instead, it focuses on speed. MySQL has a reputation for being very fast at simple data inserts and queries — perfect for the fast-paced web application world.

  Mainly because of its speed, the MySQL database server has become a popular tool for storing data in dynamic web applications. It also helps that, because it's an open-source project, web-hosting companies can install it for free, which makes it a perfect combination with the PHP server-side programming language for dynamic web applications.